SKI Report 93:06

CRACKER – A Program Coupling Chemistry and Transport Version 92-11

Allan Emrén

December 1992

ISSN 1104–1374
ISRN SKI-R-93/6-SE

CRACKER - A program coupling chemistry and transport
Version 92-11

Allan Emrén

SKI TR 93:6

Department of Nuclear Chemistry
Chalmers University of Technology
S-412 96 Göteborg

December 1992

gotab 97197, Stockholm 1993

C R A C K E R

A program coupling chemistry and transport

Version 92-11

Allan T. Emrén

Department of Nuclear Chemistry
Chalmers University of Technology
S-412 96 Göteborg

December 1992

C R A C K E R
A program coupling chemistry and transport

Version 92-11

CONTENTS

PREFACE

The development of CRACKER has been going on for a number of years. The program has now grown to a complexity which makes a convenient user interface necessary. Thus the program structure has been considerably modified. The goal has been that a user should be able to learn usage of the program in less than an hour. Further it should be possible to set up and start a new simulation in less than two minutes.

To make this possible, the program makes use of data blocks which may be accepted or modified by the user. It makes use of some 25 screens and it always suggests a value for a requested parameter. Several default values may be accepted by one key stroke.

The program is available from the author on a 3.5" DOS diskette. Anybody is free to distribute copies of the program on a non commercial basis. All programs and routines in the source code are freely available for anybody to use in their own programs.

Reports on problems in connection with the program, as well as suggestions on improvements in the code or documentation will be most appreciated.

Göteborg in December 1992

Allan Emrén
Dept Nuclear Chemistry
CTH
S-41296 Göteborg
Sweden

SUMMARY

CRACKER is a program coupling chemistry and transport. It simulates chemical reactions of groundwater flowing through a plane fracture. Properties like initial composition of the water, mineralogical composition of the rock and temperature gradients and flow velocity of the water serve as input for the modelling.

The program is designed to handle heterogeneous rock properties, like redox fronts, regions with different mineralogy etc. It is even able handle the common situation of a rock violating the phase rule.

In the CRACKER model, a rock is formed by a more or less random distribution of minerals across the surfaces of a fracture. Water moves along the fracture (in present versions at a constant velocity). Diffusion perpendicular to the flow direction is taken care of by a mixing process. No diffusion parallel to the flow direction is simulated.

The program is supplied with a user interface which makes use of some 25 screens to guide the user through the procedure of setting up a simulation. The properties of a system to be simulated consists of packages and sub packages of data. The packages may be used as are or they may be modified according to the specific situation. As often as this is possible, the program suggests an answer to be accepted or modified.

CRACKER is a package of several programs, most of them written in C. Chemical equilibrium calculations are mostly performed by the well-known geochemical program PHREEQE. The main program, CRACKER, manages information flow and determines which subprograms to use for specific tasks. Further it is responsible for the user interface. Essentially a simulation proceeds by alternate calls to the HACKER and PHREEQE subprograms. HACKER is responsible for generating the rock, water propagation, mixing of waters and sampling the results. PHREEQE is used to solve the chemical equilibrium equations.

This report is supplied with a tutorial example and a guide on how to use CRACKER. In the tutorial example, a simple simulation is set up and run. In this test example, the chemical changes of rain water flowing through a tiny fracture in a Granitic rock are simulated. In the user guide, there are descriptions on how to perform different kinds of simulations.

The directory structure and the data structures used by CRACKER are described in separate sections. Further, the different sub programs are described with respect to

purposes and methods used to handle the problems. The purpose of each first level function in the sub programs is described. For more detailed descriptions, the listings in the appendix may be consulted.

SAMMANFATTNING

CRACKER är ett program som kopplar kemi och transport. Det simulerar kemiska reaktioner mellan grundvatten som flyter genom en spricka och mineral i sprickans väggar. Vattnets kemiska sammansättning när det går in i sprickan, sprickväggens mineralogiska sammansättning och vattnets flödeshastighet tjänstgör som indata för simuleringen.

Programmet kan hantera heterogena egenskaper hos berget, såsom redoxfronter, områden med olika mineralogi etc. Det har till och med möjlighet att hantera den vanligt förekommande situationen att mineralen i berget inte uppfyller Gibb's fasregel.

I CRACKER-modellen simuleras berget genom att mineral fördelas mer eller mindre slumpmässigt över en sprickyta. Vatten rör sig genom sprickan (med konstant hastighet i den nuvarande programversionen). Diffusion vinkelrätt mot flödesriktningen hanteras med hjälp av en blandningsprocess. Ingen diffusion parallellt med flödesriktningen simuleras av den nuvarande programversionen.

Programmet är försett med ett användargränssnitt som utnyttjar ca 25 skärmbilder vilka hjälper användaren att definiera det system som skall simuleras. Systemets egenskaper beskrivs i paket och underpaket med data. Paketen kan accepteras som de är, eller deras innehåll kan ändras av användaren i enlighet med det system som önskas simulerat. Närhelst detta är möjligt, föreslår programmet ett svar som kan accepteras eller modifieras.

CRACKER är ett programpaket vars flesta komponenter är skrivna i C. Kemiska jämviktsberäkningar utförs i huvudsak av det välkända geokemiska programmet PHREEQE. Huvudprogrammet CRACKER administrerar informationsflödet och bestämmer vilka underprogram som skall användas för att lösa de olika uppgifterna. Vidare är det ansvarigt för användargränssnittet. I huvudsak sker en simulering genom att CRACKER omväxlande använder HACKER och PHREEQE. HACKER har ansvar för generering av berget, förflyttning av vattnet, blandning av olika vatten samt tillvaratagande av simuleringsresultat. PHREEQE används för att lösa de kemiska jämviktsekvationerna.

I denna rapport finns ett övningsexempel och en handledning i användandet av CRACKER. I övningsexemplet genomförs en enkel simulering. Det system som

betraktas är regnvatten som passerar en kort sträcka genom en spricka i granit. I handledningen beskrivs hur olika typer av simuleringar kan utföras.

Directorystrukturen och de datastrukturer som används av CRACKER finns beskrivna i varsitt kapitel. Vidare finns översiktliga beskrivningar av de olika delprogrammen med avseende på ändamål och metoder som används för att genomföra programmens uppgifter. Inom varje delprogramsbeskrivning finns också en motsvarande beskrivning av ändamål och arbetsmetod för de första ordningens funktioner som ingår i programmet. För mera detaljerad information hänvisas till programlistningarna i appendix.

PURPOSE OF THE CRACKER PROGRAM

CRACKER is a program coupling chemistry and transport, using elaborate chemical modelling in combination with a simplified transport model. It simulates chemical reactions of groundwater flowing through a plane fracture. The program is even able to handle the common situation of a rock violating the phase rule. Properties like initial composition of the water, mineralogical composition of the rock and temperature gradients and flow velocity of the water serve as input for the modelling.

Particularly, CRACKER is designed to handle heterogeneous rock properties, like redox fronts, regions with different mineralogy etc.

INSTALLATION

Put the distribution disk in drive A or B.

Give the command

```
A:INSTALL A
```

if you are using drive A, else use the command

```
B:INSTALL B
```

Do not forget the blank space between INSTALL and A or B

* To get started with the program, use the tutorial example (next few pages).

* To start the program, give the command

```
C:\>CRACKER
```

from the ROOT directory of the hard disk. You may also change directory and use the same command:

```
C:\CRACKDIR\WORK> CRACKER
```

* Operative system: DOS or DOS running under WINDOWS

* Hardware requirements:

IBM PC or compatible.

EGA or VGA graphics.

80286 or higher processor

8087 or higher math coprocessor or 80486 D processor

If your system does not fulfil the requirements, the source code has to be recompiled.

TUTORIAL EXAMPLE

In the tutorial example a simulation is performed, in which rain water enters a tiny fracture in Granitic rock. The water is followed a few centimetres down the fracture, and the chemical changes are shown upon the screen. When the simulation is finished, the chemical changes may be followed as diagrams. Further, it is shown how to simulate sorption of an element upon the rock surface.

The program has to be installed before it is used. Installation procedure is described on the previous page. To start the program, change to the ROOT directory and give the command

```
C:\>CRACKER
```

To avoid mistakes, read ALL text on each screen.

Screen 1: INTRODUCTION

Action: Accept screen.

Comment: If a new project is started, all results from the last project are lost unless they are saved.

Screen 2: CHOICE OF PROJECT

Action: Choose NEW

Comment: You may start a new project, or extend an old project, which has reached its earlier end point. Thus, as an example, you may let the ground water enter a new kind of rock.

Screen 3: PROJECT NAME

Action: prgtest

Comment: A project must have a name (max eight characters), for creation of files in which the information needed is stored.

Screen 4: PROJECT PROTOTYPE

Action: Choose DEFAULT

Comment: The CRACKER program may need hundreds of data values to know how to perform a simulation. Normally, you will see only a minor part of the data. The reason is that data sets used earlier are reused. You just have to handle differences between new and old projects. By choosing a prototype project, you get a package of properties to modify.

Screen 5: PROJECT GEOMETRY

Action: Set length to 10 and width to 4

Comment: By the settings now chosen, you will simulate a small fracture, two cm wide and five cm long. The water is flowing along the X direction. If you want to start a new project or restart an old project from the initial starting point, current X and Y steps should be 1. Otherwise, the values of current step should NOT be changed.

Screen 6: PROJECT DESCRIPTION

Action: Choose KBS-3

Comment: The default water (KBS-3 groundwater) will be replaced by something else in the new simulation.

Screen 7: WATER DATA FILES

Action: Choose RAIN

Comment: A package of indata is chosen by a name. You may create new indata sets by choosing NEW, choose a prototype and modify that data set.

Screen 8: PROJECT DESCRIPTION

Action: Choose FRACTURE

Comment: The rock data package FRACTURE will be replaced by another data set.

Screen 9: AVAILABLE ROCK DATA FILES

Action: Choose GRANITE

Comment: The data file GRANITE does NOT contain a correct description of Granitic rock. It is included for demonstration purposes only.

Screen 10: PROJECT DESCRIPTION

Action: Accept entire screen by choosing OK.

Comment: Most data for the simulation are now defined.

Screen 11: COMPONENT PROPERTIES

Action: Choose Rock properties

Comment: You may now modify properties of the data sets chosen in the previous screen.

Screen 12: COMMENT ON THE ROCK

Action: Accept.

Comment: The comment line is not used by the program. It is intended for your own notes on purpose etc.

Screen 13: MINERAL SET

Action: Add Hematite

Comment: You must not remove all minerals. Upper or lower case may be used. The number of minerals has to be < 50.

Screen 14: MINERAL ABUNDANCES

Action: Change to 30 % quartz and 10 % hematite

Comment: The sum of percentages has to add up to at least 100. This version of the program does not check that the requirement is fulfilled.

Screen 15: COMMENTS ON THE INDIVIDUAL MINERALS

Action: Accept the screen.

Comment: Here you may write notes for yourself.

Screen 16: COMPONENT PROPERTIES

Action: Accept the screen.

Comment: No more changes will be made during this session.

Screen 17: DESCRIPTION

Action: Choose Water chemistry simulation

Comment: The CRACKER simulation starts. Water is propagated along the fracture and reacts with minerals along its route.

Screen 18: SIMULATION PROGRESS

Action: Wait for message: End of simulation.

Press ENTER to continue.

Comment: During the simulation, local results are displayed upon the screen. In the present program version, PHREEQE writes:

'Stop - Program terminated'

and HACKER writes

'Null pointer assignment'

in each step. The messages should be ignored.

Screen 19: CHOICE OF ACTIONS

Action: Choose 'Display the simulation result'

Screen 20: DISPLAY MODE

Action: Choose 'Log concentrations vs. position'

Comment: In most display modes, the program displays properties along a central slice parallel to the water flow direction. An exception is the pH-pE diagram, which is sampled across the entire fracture surface.

Screen 21: HARDCOPY?

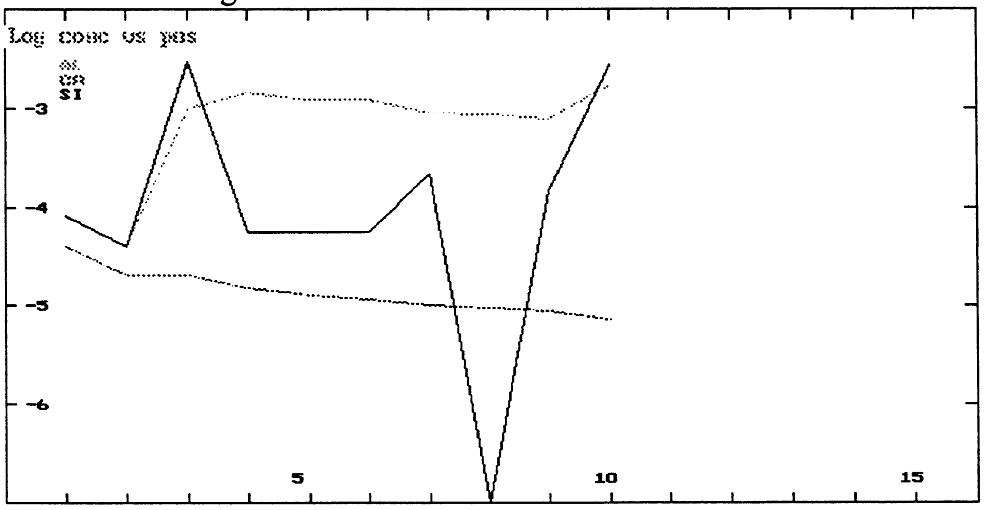
Action: Choose 'n'

Comment: The hard copy option creates a data file, which may be read by the PRINTPIC program and printed with an Epson compatible printer.

Screen 22: PICTURE

Action: To leave the picture press ENTER

Comment: Diagram as below. Each element has its own colour.



Screen 23: DISPLAY MODE

Action: Choose 'Quit'

Screen 24: CHOICE OF ACTIONS

Action: Choose 'Sorption simulation'

Comment: Sorption model data are stored in the directory SORPDATA. Note that the data in some cases do NOT represent reality. Some of them are included for demonstration purpose only. Before you use any data in this directory, you will have to look through the data files to check if they are usable for your purpose.

Screen 25: HARDCOPY?

Action: Choose 'n'

Comment: See screen 21.

Screen 26: PICTURE

Action: To leave the picture press ENTER

Comment: Each curve represents a snapshot of concentration along a slice of the fracture. Five curves should appear, representing 2, 4, 6, 8 and 10 time steps. Normally, much larger fractures as well as larger time intervals are used in CRACKER simulations.

Screen 27: CHOICE OF ACTIONS

Action: Choose 'Quit'

Comment: You are now leaving the program. Normally, you should first save the project, but since this is merely a test example, it will not be saved. A project which is not saved at this stage, may be saved when the CRACKER program is started again. A project which is not saved, will be overwritten.

THE CRACKER MODEL

Introduction

A sufficiently accurate understanding of groundwater chemistry and interactions between minerals and groundwater is of fundamental importance for performance assessments of repositories for radioactive wastes. It is a well-known fact that the heterogeneity of rock on both small and large scales puts a limit to what can be achieved in this respect. By necessity, the groundwater chemistry varies according to the mineral composition along a fracture system. This difficulty has to be handled in various stages of the complete safety analysis of a repository, from the interpretation of site specific hydrogeochemical data, and to the detailed modelling of radionuclide behaviour in the geosphere. Other areas where similar problems are evident are in the interpretation of data from natural analogues and in predictions of influence of environmental changes upon the groundwater properties.

A traditional way around these difficulties has been to assume an average composition of groundwater and mineralogy for the major parts of the system, or possibly, by letting these properties change according to observed or predicted processes. Examples of the latter approach are the modelling of redox fronts and weathering. This approach might be reliable in most cases for description of processes on a larger scale and for behaviour of macro components.

For micro components, notably radio nuclides, the situation is not that convincing, however. Important retardation mechanisms such as sorption and limited solubilities are strongly dependent on the groundwater chemistry (pH, redox potential, complexant concentrations etc.) as well as the mineralogy. In addition, there is a strive for increased sophistication in sorption models (e.g. ion exchange, surface complexation). Still these efforts rely on an assumption of averaging the chemical properties of the system, although to varying extent depending on the aim of the study. The validity of that approach could be questioned, since, in fact, there are good reasons to assume that the chemical properties fluctuate considerably along the transport path.

Conceptual model

In the CRACKER model, a rock is formed by a more or less random distribution of minerals across the surfaces of a fracture. Water moves along the fracture. In the present version it is assumed that the surfaces are plane paralell the water moving at a constant vlocity. Diffusion between different cells is taken care of by the mixing process described below.

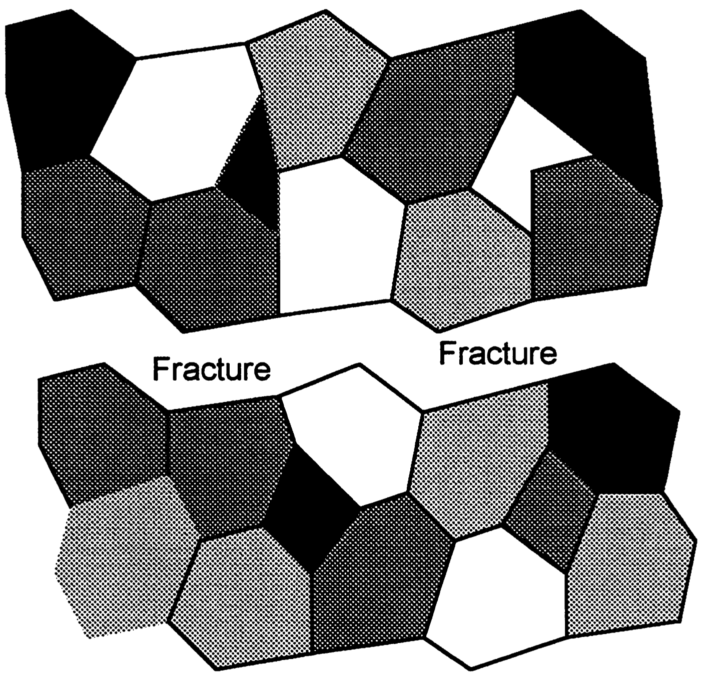


Figure 2. A fracture is surrounded by a more or less random set of mineral grains.

A key concept is that of diffusion cells. That is, a region that is chemically connected by diffusion. Different diffusion cells are assumed to be chemically independent of each other, while reactions may take place between water, dissolved species and minerals in the same diffusion cell. In each time step, the water moves from one diffusion cell to the next.

To understand the concept of diffusion cells, one has to realise that the time required for diffusion to make a water packet homogeneous is proportional to the linear size squared. Thus, if reactions with a mineral grain change the water composition, the change in composition spreads through the water with a velocity, which is initially great, but rapidly decreases. Sooner or later, the velocity becomes less than the flow velocity of the water. At that time, the region influenced has reached a certain size, defining a diffusion cell. Any water package greater than that size will be unable to reach equilibrium during the time it is contact with a mineral grain.

The size of diffusion cells depends upon the water flow velocity and the degree of homogenisation required. As an example, let us assume that the water flow velocity is 0.80 m / year (0.091 mm / hour), the diffusion coefficient is 10^{-10} m^2/s and that a deviation of 20 percent in concentration between different parts of the water package is an acceptable degree of homogeneousness. Integration of Fick's second law then gives a size of 5.3 mm for the diffusion cell.

In each diffusion cell, CRACKER makes use of PHREEQE [1] to solve the equilibrium equations for that particular cell. In most coupled codes, non-compatible mineral sets cause problems, and one often has to select a sub-set of minerals in the simulations. With the approach described here, incompatible minerals do not cause any trouble, as far as each diffusion cell (a few mm^3) contains a mineral set that is internally compatible. Since each cell normally contains only one to three minerals, this property is easily achieved.

Two dimensional fracture model

The fracture model used here consists of a slit between two plane parallel rock surfaces. Minerals are randomly distributed across the surfaces. Each mineral grain is considered to cover a hexagonal area of the rock surface. Occasionally, other grain geometries are formed by the random allocation process, since the same kind of mineral may be placed in several neighbouring hexagons. The size of a hexagon is the same as that of a diffusion cell.

With the water velocities typical for deep Swedish rock (0.5 - 1 m/year) the size of a diffusion cell is a few millimetres. Although the water flow is supposed to be laminar, there is no need to consider the variation of water velocity as a function of the distance to the wall. The reason is that due to diffusion, all water in the same diffusion cell has the same composition. As the water propagates through the system, water from two diffusion cells is transferred in the model to the next cell and mixed. To avoid boundary effects, periodic boundaries are used in the simulations.

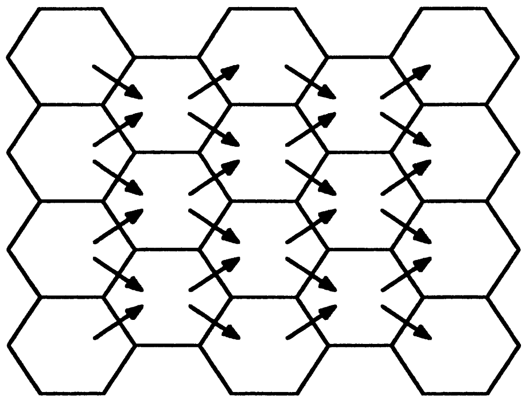


Figure 3. Water from different diffusion cells is mixed as the water flows through the fracture.

At each time step, PHREEQE is used to first equilibrate the mixture from the previous step and then to equilibrate the resulting solution with the mineral (or minerals) present in each diffusion cell. Since a Gaussian is a limiting function for the binomial function it follows that if the size of the diffusion cells is chosen in a proper way, the mixture process gives a good approximation to diffusion perpendicular to the flow direction of the water. Diffusion in the flow direction or backwards is neglected in the present program version (except for diffusion within the cells).

Results from the time steps are stored in a result file containing information about water composition, pH, pE and minerals present in each diffusion cell.

Handling of sorption

The program makes use of a sorption model based upon the K_d concept. K_d is changed into K_V (= K_d $*$ rock density). The program makes use of the equation

$$K_d = \frac{(C_0 - C_{sol})V_{sol}}{C_{sol}M_{rock}} = \frac{n_{rock}}{C_{sol}V_{rock}\text{Rockdensity}} \quad \text{and thus}$$

$$K_V = K_d\text{Rockdensity} = \frac{n_{rock}V_{sol}}{n_{sol}V_{rock}}$$

The rock volume is area multiplied by thickness of sorption layer assumed in the K_d measurement. Experiments by Allard [2] indicate an approximately linear relationship between the logarithm of K_V and pH up to a saturation limit, above which K_V is approximately constant. A simlar relation is supposed to exist for K_V and pE, with the saturation occurring at low pE values. This results in a relation of the shape

$$K_V = K_{V,std} * 10^{(\,pE + 2.0\,) * pE\_coeff\ * (pH - 9.2\,) * pH\_coeff}$$

The cut off values for pH and pE have been more or less arbitrarily chosen to 9.2 and -2.0 respectively. Grain sizes in Allard's experiments have been 0.05 mm. Assuming spheres, this results in apparent K_d which are 3 - 4 times too great for plane surfaces.

Std K_V(pH = 9.2, pE = - 2.0) = K_d(pH = 9.2, oxidising) * 500 / 4 * rockdensity

Sorption properties are described in mineral files in the directory WORK\SORPTION. The files are supposed to be self explanatory.

To change sorption model, you have to modify the SORPTION.C program in the following ways:

Modify the global structure SorptionData, the functions Sorption() and Read_sorption_data().

The functions are small and easily changed. They do not interfere with any other functions in the program. Their only actions are to decrease the concentrations of elements for which sorption data are known.

PHREEQE instabilities

It is well-known [3] that PHREEQE sometimes has difficulties in finding a result when the solutions involved are complex. In particular, this is the case when pH and pE are determined by the reactions rather than kept at constant values. Since simulations as those performed by CRACKER have to allow for the reactions involved to change pH and pE, there are risks for PHREEQE problems.

If PHREEQE fails to converge, CRACKER tries to adjust the convergence parameters of PHREEQE. In most cases this is successful and the program is able to proceed.

Sometimes the problem is not cured and in that case CRACKER is interrupted. Then the user has to fix the problem manually. This may be performed by further manipulation of the convergence parameters or by treating the mineral reactions as "REACTION" rather than as "MINERALS" (in the PHREEQE command language). If no success is achieved, it may be necessary to temporarily replace the mineral causing trouble with something else. When PHREEQE has converged, the result file (RESULT.DAT) should be modified to mimic an ordinary PHREEQE outdata file with mixture of two solutions and equilibrium with one or more minerals. Then the command line

```
>HACKER
```

should be entered. Finally, the CRACKER program may be started by the command line

```
>CRACKER
```

The simulation then continues where it was interrupted.

Warnings

PHREEQE sometimes converges but gives totally wrong results without any kind of warning. The present version of CRACKER has no way of detecting the problem. This kind of problem has been observed in cases where charge balance has been maintained by automatic addition of ions to the solution, and in cases with very low redox buffering capacities of the solutions. To avoid this kind of problem, the header ADJ-PH should be used in the main part of the simulations. For charge balance purposes, the headers NACL or NASO4 may be used in an initial simulation with merely one step. The quantity of ions added by PHREEQE may then be manually added to the initial solution and the ADJ-PH header could be used without any pH stability problems.

If the redox buffering capacity of a solution is small (as is often the case with ground water), numerical instabilities may cause the PHREEQE results to be useless. As an example it may be mentioned that mixing of two identical solutions caused PHREEQE to predict pH and pE to change of several units in the mixture. To avoid this kind of problem, the data base has been supplied with a "redox element" Rx, of which a small quantity (about 10^{-10} mole) may be added to the solution if there is any reason to suspect that numerical errors disturb the results.

Kinetic effects

The present version of CRACKER is an equilibrium code, and does not take kinetics into account. Often minerals with slow kinetics are excluded from simulations. In the long run this approach causes errors in the predictions, since eventually even a slow reaction will proceed enough to be of significance. In CRACKER another approach is available as well. The data base is supplied with an "inert element" It and a corresponding mineral Inert. This mineral may be used to "dilute" the reactions of one or more minerals to make them appear to be slow. Let us assume that a certain quantity q of a mineral will have reacted at equilibrium, but that merely 0.05q is estimated to react during the contact time (a couple of days). Further let us assume the rock consists of p percent of that mineral. Then the rock composition should be declared as 0.05p of the actual mineral and 0.95p of Inert. This approach does not treat the kinetics in a perfect way, but at least it could be expected to give results closer to reality than the approaches to neglect the corresponding reaction or to let it reach equilibrium.

HOW TO USE CRACKER

CRACKER is intended to simulate groundwater flowing through a fracture. In the simplest case, each diffusion cell contains merely one mineral. This case is illustrated in the tutorial example. In the sections below it is described how to perform simulations that deviate from the basic case.

Continuing an old simulation

It is possible to run a short test simulation and to continue later if the results look interesting. Thus when a simulation has reached its end, it may be continued. The same is possible if it is interrupted by some error, a power failure or some mistake. If the result has not been saved, this has to be done. Otherwise the information needed to continue will be destroyed.

A continued simulation may be a pure continuation of an old project or it may be started as a new project with the old project to be continued as prototype. A project may be continued just by a change in the fracture length. Then everything else is left unchanged. It is also possible to let the water enter another kind of rock or to modify the rock properties. Changing the water properties is impossible unless in a continued simulation. Any attempts to perform such changes are ignored by the program.

NOTE! The fracture width or the present position must not be changed as a project is continued. Such modifications will result in unreadable result files. It is not recommended that the fracture thickness and water speed are changed in a continued simulation. The programs will be able to read the result files, but they will be misinterpreted, since the result handling programs will assume the most recent values to be valid for the entire simulation. Should changes of the kind mentioned be desired, the project should be continued by using the final result as input for a new simulation starting at step (1, 1).

Restarting an old simulation

If a simulation has been unsuccessful it may be started from the beginning, perhaps with a slightly modified water or another kind of rock. It is also possible to study the influence of random fluctuations by restarting a simulation with a new value of the seed for the random number generator. The seed should be an integer within the interval (1, 32767). This will result in a different mineral distribution across the rock surface.

Further, one may desire to perform exactly the same simulation with a more or less modified data base. Then the same seed should be used again. Further the original first mineral should be restored. This ensures that exactly the same rock surface will be generated.

Saving results

The simulation results are not automatically saved. The reason is that when a result is saved, possible old results of the same project are overwritten. It should be noticed however, that an unsaved result will be destroyed as a new simulation is started or an old simulation restarted.

Several minerals in the same diffusion cell

If mineral grains are considerably smaller than the size of a diffusion cell, the water is normally in contact with more than one mineral grain at a time. To handle this situation, the program has the ability to have several minerals in the same diffusion cell. This is performed by creating pseudo minerals in the <MINERALS> directory. Each mineral has a file with the name extension .MIN. Thus a mineral like quartz is described in the file QUARTZ.MIN, the content of which is

```
QUARTZ      2   0.0       -4.006    6.22          0
  13 1.0          3 -2.0
```

The two lines have been copied from the PHREEQE data base. Similarly pyrite is described in the file PYRITE.MIN:

```
PYRITE      4   0.00      -18.48     11.30         0                .000
   1  -2.000       2  -2.000      8  1.000      42  2.000
```

To create a pseudo mineral, qp, consisting of quartz and pyrite grains the files are simply added to form a new file QP.MIN:

```
QUARTZ      2   0.0       -4.006    6.22          0
  13 1.0          3 -2.0
PYRITE      4   0.00      -18.48     11.30         0                 .000
   1  -2.000       2  -2.000      8  1.000      42  2.000
```

When this file has been created, qp may be used as an ordinary rock component. The limitations are that only mineral sets fulfilling the phase rule are allowed to appear in the same pseudo mineral. Further the number of minerals in one pseudo mineral must not exceed 20.

If the mineral is being used in sorption simulations, a .SRP file has to be written. The format of such a file is described under the headline DATA STRUCTURE. The coefficients have to be calculated from those of the mineral components. Further, the relative abundances of the different mineral components have to be taken into account.

Charge balancing

Charge imbalances in the water may be adjusted by a change in pH. This is the normal procedure since it makes PHREEQE perform in the best way. To use this option, one has to manually perform charge balancing. There are two predefined headers that may be used for this purpose, NACL and NASO4. If CRACKER is run with one of those options for just one step (endpoint = startpoint), the result will be a charge balanced solution. The composition of that solution is used to modify the initial solution to be used together with the ADJ-PH header. The two headers mentioned above adjusts the charge balance by adding Na^+ ions to increase the charge and Cl^- or SO_4^{2-} respectively to decrease the charge. New options may be created by the user. They are stored in the <HEADERS> directory and are used by HACKER to build the first few lines of the indata file for PHREEQE. Thus, it is possible to modify the way in which pE is treated, the species used for charge balancing etc. Further, new lines may be added, to calculate SUMS etc.

One dimensional simulation

Normally, the CRACKER simulations are two dimensional. If a one dimensional simulation is desired, the fracture width should be put to one step.

Air saturated water

Among the minerals, there is one pseudo mineral named air. This mineral consists of oxygen and carbon dioxide at approximately the partial pressures of clean air. This pseudo mineral may be used as one of the rock components in simulations where the water is supposed to be in contact with air (e.g. fractures near the surface). It should be noted that the air composition in the ground may deviate considerably from that above the surface. In particular, there is often a much higher partial pressure of carbon dioxide in the ground. Thus it may be necessary to create a new pseudo mineral, e.g. SOILAIR.MIN to take care of the deviating composition. This is easily made by a change in the SI parameters of the AIR.MIN file.

Changing the data base

The data base supplied with the program, LIBRARY.DAT, has been made by combining values from several other data bases and by estimating a few missing values (in particular the enthalpy of formation for amorphous $Fe(OH)_3$). It is the responsibility of the user to make sure that data used are correct. To use another data base, one simply has to make sure that the number of species etc. in the new data base are within the range limits of the used PHREEQE version. Further, if the number of an element or a species is changed, one has to change the corresponding numbers in all .MIN files (in the directory <MINERALS>).

PROGRAM STRUCTURE

CRACKER is a package of several programs, most of them written in C. Chemical equilibrium calculations are mostly performed by PHREEQE. The main program, CRACKER, manages information flow and determines which subprograms to use for specific tasks. Further it is responsible for the user interface.

All subprograms are able to run as self consistent units, but since the general data structure of CRACKER is rather complex (as a result of the complicated situations to be modelled), it is usually more convenient to use the full CRACKER program even for a simple task, like running a few PHREEQE calculations.

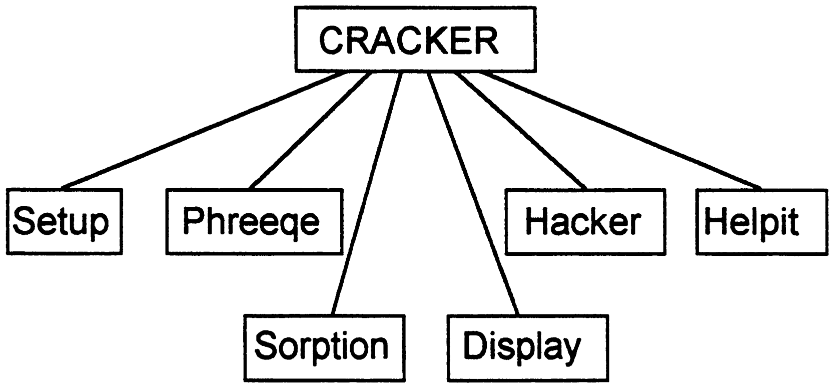


Figure 4. Hierarchical structure of CRACKER.

The general structure of CRACKER is shown in figure 4. The program starts by a call to the Setup subprogram. In this program, all data needed to describe a simulation are entered and stored in a number of files. As the control returns to the main program, a couple of files are manipulated to create an environment for the simulations to be performed later.

When the general data are ready, the main menu is presented to the user. At this point the CRACKER program is able to branch in several directions depending upon the choices of the user. The main branches are

Water chemistry simulation
Sorption
Display of results
Save results
New or modified simulation
Quit

In "Water chemistry simulation" the program walks across the surface in the Y direction (perpendicular to the water flow). When it reaches the edge of the simulated fracture, it moves one step in the X direction (parallel to the water flow) and the Y steps are repeated. In each step (diffusion cell), a mineral (or set of minerals) is chosen and an indata file for PHREEQE is created by the Hacker sub program.

PHREEQE is then called and the result is analysed by Hacker. If PHREEQE failed, the Helpit program is called to create an indata file with adjusted values of the convergence parameters for PHREEQE. When a successful PHREEQE result is present, Hacker extracts water properties and stores the result in result files. This procedure continues until the end of the fracture is reached.

From the main menu, one may now choose "Display results" to get graphs describing the variation of different water properties across the fracture surface. Properties displayed are pH, pE or a desired number of concentrations as a function of position.

Another possible route is to perform a sorption simulation, which means that the Sorption subprogram is called. This program reads the global result file of CRACKER to find mineral distribution and water properties across the surface (steady state is assumed). The sorbing element is entered at the entrance of the fracture. In each time step the simulation starts at the outlet and proceeds towards the entrance. The sorbed quantity and concentration in the liquid phase are calculated for each diffusion cell. A diagram at the screen makes it possible to follow the simulation as the time propagates.

Child processes

There are two levels of child processes in CRACKER. The highest level is shown in figure 4. A simulation always starts by a call to SETUP, for creation of the indata files necessary for the simulation.

Child processes are created by two different methods. In both cases the parent process is waiting for the child process to be ready before it is allowed to continue. To speed things up, this may be changed in future versions for parallel computer systems.

If the sub program generates a simple result like SUCCESS or FAILURE the spawn command is used. In this case the keyboard works undisturbed. Program interrupts by CTRL/BREAK are the possible.

If no simple results have to be transferred, the system command is used. Often the system command is used in combination with redirection of input or output. In this case attempts to interrupt the program from the keyboard may be ignored.

Screen handling

In the include file SCREEN.H a number of routines for input from the screen are defined. Each of them makes use of background text created by a text editor. The background text contains specific information on choices available. General information on how to enter properties are supplied by the screen functions. As an example, the "Simulation geometry" screen background is shown below.

The static information is displayed each time that particular screen has to be used. It serves as background for the suggested values displayed by the screen routine. The values may be highlighted and edited with the aid of arrow keys.

```
      S I M U L A T I O N   G E O M E T R Y
If you want to perform a one dimensional simulation,
make the fracture width = 1 step

Do not change width or stepnumbers in a continued project.
Start a new project at length- and width step = 1

Fracture length (steps)
Fracture width (steps)
Current length step
Current width step
LastRandom
Next mineral
Temperature (  C )
Temperature gradient (K/m)
Water speed (m/year)
Step length (m)
```

All changing information is transferred to the program via the global structure Inscreen. The use of this structure is described below:

```
Inscreen.x = 50;      /* X-position of input fields      */
Inscreen.y = 10;      /* Y-position of first input field */
Inscreen.items = 10; /* Number of input fields          */
```

All variable information that might be modified by the user is transformed to ASCII form.

```
sprintf(Inscreen.text[ 0 ],"%d",Geom.length);
sprintf(Inscreen.text[ 1 ],"%d",Geom.width);

Screen_input("..\\screens\\geometry");  /* Name of screen picture */
```

When a screen is accepted the result has to be transferred back to the variables asked for.

```
Geom.length = Inscreen.number[ 0 ]);
Geom.width  = Inscreen.number[ 1 ]);
```

Each result appears in three shapes, as a text string (.text), as an integer (.number) and as a floating point variable (.real). The functions always try to transform the result to all three kinds of shapes. It is the programmer's responsibility to use the correct shape. If the wrong shape is used, the result may become meaningless.

Screen functions are available for input of text and values, for choice of files and for choice of branches.

Communication

Although the sub programs are able to work as stand alone units, they have to exchange considerable amounts of information when they are working together to fulfil a simulation task. As has been mentioned above, simple messages like "SUCCESS" or "I_NEED_HELP" are transferred via exit codes. At the transmitting end it may be coded like

```
exit( FAILURE );
```
The message is received in the calling statement like
```
message = spawnl( P_WAIT, HACKER.EXE, HACKER.EXE, NULL );
```

More complex messages are transferred via files. Examples of such files are the geometry file, the rock description file, indata file for PHREEQE, PHREEQE result file and the global result file.

Project management

When a new project is started up or an old project reactivated, the files describing the project are copied to the work directory (CRACKDIR\WORK). A project consists of about ten files. One file, CURRENT.PRJ, contains the name of the current project. The other project files have names with the continuation .DAT. Most of them contain general information on the project, like rock properties, diagram appearance or geometry. A few files contain temporary messages to the sub programs that may be called during the simulation.

When a new project is started the files describing the old project are overwritten. To prevent the information from being destroyed, there is a possibility for the user to first save the old project. The information is then distributed among several directories, some of which are <PROJECTS>, <WATERS> and <ROCKS>. Later, one may continue the same project (possibly with some new properties), or bring parts of it into another new or continued project.

The project is not saved unless the user chooses that option. The reason is that when a project is saved, the old version of the same project is overwritten. In some cases the user may want to undo a continuation of an old simulation, and that would be impossible if the old results were automatically overwritten.

Simulation progress

The main simulation is the water chemistry simulation. Here the subprogram units PHREEQE and HACKER are used to perform the simulation. PHREEQE is used to perform equilibrium calculations.

The HACKER subprogram starts by reading the chemical data base (LIBRARY.DAT) and the latest PHREEQE result file (RESULT). It checks that PHREEQE has converged. If this is not the case, HACKER calls for help from CRACKER, which tries to solve the problem by adjusting the convergence parameters of PHREEQE.

If the result appears OK, HACKER continues by reading information on geometry and current position (POSITION.DAT), rock properties (ROCK.DAT) and previous results (OLDSLICE.DAT). The current PHREEQE result is appended to the file NEWSLICE.DAT.

CRACKER slices the fracture perpendicular to the water flow direction. In each slice the program propagates from one side to the other. In practise, this stepping is performed by HACKER. When the far end of a slice is reached, OLDSLICE.DAT is appended to the global result file RESULT.DAT, NEWSLICE.DAT is copied to overwrite the OLDSLICE.DAT file and an (almost) empty NEWSLICE.DAT file is created.

HACKER makes a random choice of mineral(s) for the current diffusion cell and transfers the contents of the neighbouring cells in OLDSLICE.DAT to solution 1 and 2 to be mixed as they enter the current cell. A PHREEQE indata file is created, which describes the mixing and equilibration with the minerals.

Finally HACKER updates the POSITION.DAT file and checks if the endpoint of the simulation has been reached. With the proper message it quits and CRACKER takes over the control.

Portability

The programs are written in C. The source code may be compiled with the Borland Turbo C++ compiler. System specific commands used in the programs are system( ) and spawn( ). Those functions may have to be replaced in other environments. In most cases they may be replaced by fork( ) statements.

Besides the system specific functions mentioned above, all system specific code is gathered in the include files ALLAN.H and SCREEN.H. Some of the functions found in these files may have to be changed if a different environment is to be used. The modifications should not cause much trouble, since the functions performing complex tasks are using a number of very elementary system specific functions.

USER INTERFACE

The most important part of the user interface system is the screen handling routines described above. The properties of a system to be simulated consist of packages and sub packages of data. The packages may be used as are or they may be modified according to the specific situation. As often as this is possible, the program suggests an answer to be accepted or modified. The programs do not distinguish between upper and lower case characters. Thus, Fe, FE and fe are interpreted to be the same element. The use of both upper and lower case characters is however encouraged, since this increases the readability and decreases the risk of mistakes.

In some cases the program asks a simple question. Possible answers are shown together with the question.

While a water chemistry simulation is going on, it may be followed on the screen as a condensed result output. The sorption simulation is followed in a diagram, which is completed during the simulation.

DIRECTORY STRUCTURE

The programs and data needed to run CRACKER are stored in a number of directories. They are created by the installation program. The purposes of the different directories are shown below as well as the hierarchy of sub directories.

CRACKDIR			The main directory of CRACKER
	SCREENS		Text files to be displayed as background for the screen input functions.
	SOURCE		Source code for all programs (except .bat files). No source code for PHREEQE is supplied.
	WORK		Executable codes and most of the data needed for a particular simulation.
		PROJECTS	Files describing building blocks used to define the different projects as well as their geometry.
		HEADERS	The first few lines of indata files for PHREEQE (options, STEPS, NEUTRAL etc.) are stored in files used to build complete indata files for the program.
		DIAGRAMS	Descriptions of diagram properties (what to display, axes, number of curves, intervals etc.).
		WATERS	Descriptions of initial compositions and other properties of waters entering a fracture.
		ROCKS	Descriptions of mineral compositions in rocks.
		MINERALS	Chemical properties of minerals described according to the PHREEQE MINERALS format. Pseudo minerals consisting of several real minerals may be used to simulate more than one mineral per diffusion cell.
		SORPDATA	Files describing sorption data for elements on a number of minerals.
		LIBS	PHREEQE library files.
		RESULTS	Result files generated by simulations. Contain information on rock and water properties as a function of position in x and y direction.

DATA STRUCTURE

Internally, structures are used by CRACKER and most of the sub programs. Names of variables (and functions) are chosen to be self explaining as far as possible. Externally, data are stored in files, usually with one value per line, and often with explaining text that makes the files more readable. The idea is illustrated by the geometry file shown below.

```
Fracture length (steps)
100
Fracture width (steps)
1
Length StepNo
101
Width StepNo
1
LastRandom
32093
Next mineral
biotite
Temperature (C)
25.000000
Temperature gradient (K/m)
0.000000
Water speed (m/year)
0.800000
Step length (m)
0.005000
```

Projects

The project description file contains names of other files to be used in the simulation. It may look like this:

```
ADJ-PH
RAIN
DEFAULT
GRANITE
DEFAULT
DEFAULT
```

The first DEFAULT is the name of a file describing sorption properties of a hypothetical element similar to Np. It is recommended that more descriptive names are used. The two bottom lines are names of files describing diagrams.

Unlike most other files describing a project, this file is not copied to the work directory. In the work directory merely the project name is to be found (in the file CURRENT.PRJ). The project file itself is in the PROJECTS directory with the extension .PRJ. The geometry files are in the same directory and are supplied with the extension .POS.

Waters

In the WATERS directory, two kinds of files are found. They are supplied with the extensions .WAT (in CRACKER format) and .PHR (in PHREEQE format) respectively. Since it is not foreseen that the user has any need for editing a .WAT file, it is not supplied with comments. Such a file may look in the following way:

```
Standard rain water
muscovit
7.505000          pH
15.000000         pE
10.000000         T (C)
3                 Number of elements
NA 1.223000e-04
CL 1.890000e-04
C  2.330000e-04
```

Rocks

In the ROCKS directory, files describing rock properties are found. Since they may be used to run HACKER manually in case of convergence problems, they are supplied with description lines. If a rock file is modified manually, it is essential that each mineral group starts with an empty line. The nucleation probability is not used in the present version of CRACKER. An example of a (shortened) rock file is shown below.

```
Granite for test purposes
Number of minerals
6

Comment
SiO2
PHREEQE name
Quartz
Nucleation prob
0.000000
Frequency
35.000000

Comment
K-Al-Si-O
PHREEQE name
muscovit
Nucleation prob
0.000000
Frequency
20.000000
.
.
.
.
etc.
```

Sorpdata

In the sorpdata directory, coefficients for the sorption relation are stored. In this directory there is one file for each mineral. By mineral is meant a real mineral or a pseudo mineral consisting of several minerals as described above. In the file, the coefficients are stored together with explaining text. All elements for which sorption data are known should be described in that file. The file name extension is .SRP. If the SORPTION subprogram does not find data for a certain combination of element and mineral, it assumes that the element is not sorbed by the mineral.

As an example, the file HEMATITE.SRP is shown below. The explaining texts are not used by the program. The are there to make the file understandable. The text lines may be replaced by empty lines if desired.

```
Sorption on HEMATITE
Number of elements with known sorption data
1
Element 1:
NP
Thickness of sorption layer (meter)
2.5E-5
KV ( m^3 / m^3 ) at pE = - 2.0 and pH = 9.2:
1.3E+3
pE-coeff:  KV(pE) = KV(-2.0) / ( 10 ^ ( (pE+6) * pE_Coeff ) )
0.26
pH-coeff:  KV(pH) = KV( 9.2 ) * ( 10 ^ ( (pH-9.2) * pH_Coeff ) )
0.75
```

SUB PROGRAMS

A brief description of the sub programs is given below. The two include files are used by most of the C programs to make the programs themselves as system independent as possible. Further they are used to shorten the program source codes.

Include file allan.h

SCREENWIDTH, SCREENHEIGHT are parameters describing the desired number of pixels to use for graphics display. They have to match the physical screen.
Open_screen() sets up a graphic screen according to default screen description.

Coordinate(float x_left, float y_down, float x_right, float y_up) sets up a logical screen with coordinates ranging from (x_left, y_down) to (x_right, y_up). X-values are increasing to the right and y-values upwards. All graphic functions with float arguments listed below make use of this coordinate system.

Coordinate_system(float x_left, float y_down, float x_right, float y_up) sets up a logical screen and draws a frame along the limits of the specified display area.

Locate(int line, int column) moves the text cursor to a specified line and column on the physical screen. It does not move the logical pencil.

Move_to(float x, float y) repositions the logical pencil to a specified position on the logical screen.

Line_to(float x, float y) draws a line from the current logical pencil position to (x, y) on the logical screen. The line is displayed on the physical screen and the logical pencil is left at the new position.

Setpixel(float x, float y) sets a pixel on the logical screen and displays it on the physical screen. The logical pencil does not move.

Color(int number) changes the colour of the logical pencil. The argument has to be an integer. The numbers may be replaced by standard colour macros (WHITE, BLUE, BLACK etc.).

Boarder_color(int number) defines the colour to be assumed as boarder for an flood filled area. If the function is not called, the colour of the logical pencil is assumed to be boarder colour.

Background_color(int number) sets the screen background to a desired colour.

Clear_screen() sets the screen to the background colour (default: BLACK).

Read_pixel(float x, float y) returns the number of the colour displayed at (x, y) on the logical screen.

void Paint(float x, float y) flood fills an area on the screen with the colour of the logical pencil. The painting stops when a closed curve in colour "boarder colour" is met.

Print(char *string) prints a text string at the position of the logical pencil.

Printret(char *string) prints a text string at the position of the logical pencil and moves the logical pencil to the start of next line.

Get_image(float left, float down, float right, float up) allocates memory and stores a rectangular area from the screen in that memory region. The region may be dumped to disk or copied to the screen later. May be used for animation purposes.

Put_image(float left, float up, mode) displays an image saved by the Get_image function. The coordinates define the position of the upper left corner of the image. The drawing mode may be defined by one of the macros COPY_PUT, XOR_PUT, OR_PUT, AND_PUT, NOT_PUT . With the aid of drawing modes, transparency and similar properties may be achieved.

Dump_screen(char *filename) puts a bitmap image of the present screen in the specified file. Colour information is lost.

Read_screen(char *filename) displays an image stored upon the disk. Any colour that is not background is displayed as white.

Close_screen() releases the memory used for graphics and makes the system return to an earlier screen configuration.

Random_float() generates a random number in the interval (0, 1).

float Fileinput(FILE *source) reads a line of text from a file. If the line starts with a numerical value, this value is returned. Otherwise, the value 0 is returned.

float input(question) asks a question and waits for input of a numerical value from the keyboard. Example:
meters = input("Length in cm? ") / 100.0;

Wait_key() causes the program to wait until a key is pressed. The function returns the code value of that key.
Inkey() checks if a key has been pressed. If so, the code value is returned.

Trim(char *string) takes away RETURN and LINE FEED characters from a text string.

Trim_space(char *string) takes away RETURN, LINE FEED and trailing SPACE characters from a text string.

char *Filetext(char *string, short length, FILE *source) reads and Trims a text string from an open file. Returns NULL if end of file is hit.

Include file screen.h

Screen_input(char *screen_name) Makes it possible for the user to enter several values by modifying values chosen with the arrow keys. Text, integer and float values may be mixed.

Screen_msg(char *screen_name) writes a message on the screen and waits for the user to press a key.

Screen_list(char *screen_name) similar to Screen_input, but two columns are used, of which one column may be reached by the arrow keys. The content of the unchangeable column is defined in the string variables Inscreen.description[i].

Screen_choice(char *screen_name) displays a number of alternatives, which may be highlighted by the arrow keys and chosen by the ENTER key.

char *Screen_file(char *path) searches a specified directory for files fulfilling desired criteria, takes away file name extensions, displays the file name list and gives the user a possibility to choose one of the names or to define a new name. If the alternative NEW is chosen, an old file may be chosen as prototype for the new one.

Edit_string(short y, short x, char *string) displays the string at line y, column x in highlighted form and supplied with a cursor. The user may walk with arrow keys, modify the text and exit with the RETURN key.

Highlight(int x, int y, char *position) writes a specified character in reverse colours at specified line and column.
Lowlight(int x, int y, char *position) writes a specified character in normal colours at specified line and column.

Mark(int x, int y, char *string) displays a string in reverse colour at a specified position on the screen.

Unmark(int x, int y, char *string) displays a string in normal colour at a specified position on the screen.

char *Skip_extension(char *file) takes away the extension part of a string containing a file name.

DISPLAY.C

The purpose of this program is to draw graphs to display simulation results. The program runs in an endless loop, from which it may escape via en exit statement. A number of different properties may be displayed, and the pictures may be stored as files, to be entered into other programs.

The main loop has the following content:
First the property to be displayed is chosen.
```
 Choose_type();
```

The result is stored in the global variable Type. The following choices are available:
pH vs. position
pE vs. position
Frequency of pH - pE combinations
Concentrations of elements vs. position
Quit

The program checks if the quit option has been chosen.
```
 if( Type == 6 ) exit(0);
```

If the user wants to see a diagram, the user is asked whether a hard copy of the display is desired and then the program reads the geometry file (POSITION.DAT) to check that a display is meaningful.
```
 Read_position();
```
If there is nothing of interest to show, the program quits automatically.
```
 if( XStepNo < 4 ) exit(0);
```

Most of the properties are displayed in two dimensions. The frequency of pH - pE combinations is an exception. The program checks the number of dimensions:
```
 if( Type != 3 )
```

If the variable Type is not 3 the program defines a two dimensional display, draws axis, sets marks and labels along the axis and writes some text.
```
 Setup_2d_display();
```

The program then reads the global result file, extracts the desired information and draws the curves.
```
 Read_result();
```

Should the user request a three dimensional diagram, the first step is to get information on how to organise the pH- and pE-axis. This information is copied from the corresponding information on two dimensional diagrams showing pH vs. position and pE vs. position:
```
 Define_pH_pE_axis();
```

Then a three dimensional coordinate system is initiated, the transformations from space coordinates to screen coordinates are determined and the coordinate system with marks and labels is displayed.

```
Setup_3d_display( "pH", "pE", "Freq" );
```

The result is read from disk, and pH - pE information is sampled in classes depending upon which pH - pE -rectangle the value belongs to.

```
Read_result();
```

After all values have been read, the samples are displayed as vertical lines.

```
Draw_pH_pE_sample();
```

The picture is displayed until the user presses a key.

```
Wait_key();
```

If there has been a request for a hard copy a disk file containing bitmap information is produced.

```
if( File_name[0] != (char)0 ) Dump_screen( File_name );
```

Finally, the screen is closed and the main loop started over again.

HACKER .C

The purpose of the subprogram HACKER is to generate the actual rock, to propagate the water, perform mixing, produce indata files for PHREEQE and save the results in the three files NEWSLICE.DAT, OLDSLICE.DAT and RESULT.DAT. The main function does not contain any loops and it is able to give three kinds of exit messages: NEED HELP, CRACK END and SUCCESS.

The first step taken by the program is to read element data from the PHREEQE data base (LIBRARY.DAT). The information needed is element names and the corresponding PHREEQE numbers.

```
Read_element_data();
```

The last result file from PHREEQE is evaluated. If PHREEQE has failed, a message is sent to CRACKER and HACKER exits. Otherwise relevant parts of the result are gathered and stored in data structures.

```
 if(Read_PHREEQE_result() == FAILURE) exit( NEED_HELP );
```

The Read_PHREEQE_result function essentially looks for key sequences in the result file of PHREEQE. An example of a key sequence is "PHASE BOUNDARIES". To find such sequences the function Search_text is used:

```
 if( ( result=Search_text("PHASE BOUNDARIES") ) == SUCCESS ) ...
```

When a key sequence has been found, the program knows where to find the desired information.

The geometry file (POSITION.DAT) is then read. In this file, there is information on present position, fracture size, water speed, seed for random number and similar information that is needed to organise the simulation in a proper way.

```
 Read_position_data();
```

Data extracted from the PHREEQE result is stored together with geometrical information in the result files.

```
 Save_desired_result();
```

To generate the rock surface, the program needs information on mineral abundancies. This information is present in the ROCK.DAT file created by CRACKER.

```
 Read_rock_data();
```

A new diffusion cell is created, water from previous cells are made into SOLUTION 1 and SOLUTION 2 to be mixed and mineral or minerals are chosen. The information is used to create an indata file for PHREEQE. In each step, the program propagates perpendicular to the water flow direction. When the edge is reached, the OLDSLICE file is appended to the global result file, NEWSLICE becomes OLDSLICE, a new NEWSLICE file is created, step number in y direction is reset and the x step is increased by one.

```
 Make_next_cell();
```

The updated geometrical information is stored in the file POSITION.DAT.

```
 Save_position();
```

Finally the program checks if the end of the simulation has been reached.
```
 if( Spot.XStepNo > Spot.XMaxStep )
```

If so, the message CRACK END is sent to CRACKER:
```
 exit(CRACK_END);
```

Otherwise, the message for CRACKER simply states that everything is under control:
```
 exit( SUCCESS );
```

HELPIT.C

In one to twenty per cent of the cases PHREEQE fails in some way. The failure may result in erroneous results or in a failure to converge. The present version of HACKER does not check that the result is reasonable. Thus only failure to converge is detected. If such an error is found, HACKER calls for help. When CRACKER receives such a message, the HELPIT program is called. This program inserts a KNOBS statement in the indata file for PHREEQE. Then PHREEQE is called by CRACKER again. Usually it will now converge, although it will be comparatively slow.

PHREEQE

PHREEQE is the well-known program written by D.L. Parkhurst, D. C. Thorstenson and L. N. Plummer. The version used in CRACKER is the 1984 NEA version of NEWPHREEQE. It has been slightly modified to permanently make use of the following file names: LIBRARY.DAT for the library file, INFIL.DAT for the indata file and UTDATA for the result file of PHREEQE. If a more recent version of PHREEQE is used in CRACKER, the corresponding changes have to be made in this version.

SETUP.C

The purpose of the SETUP sub program is to create the data files needed to run a simulation. The program displays a number of main screens in sequence. When necessary, it branches into sub screens for detailed modifications before it continues along the main route.

The first task is to decide a project name. The user may choose among earlier projects, which may be continued or an entirely new project.

```
 Choose_project();
```

If the "new project" option is chosen, the user is requested to choose an old project as prototype. In this way, most properties may be inherited rather than set up from scratch. When the project has been defined, a number of project description files are copied to the WORK directory. Then the user is given the opportunity to modify the project geometry.

```
 Setup_geometry();
```

Next the present choice of packages describing water, rock, PHREEQE header, sorption properties and display properties are shown. The user is able to get other packages or create new ones, in which case the prototypes are chosen from available data files.

```
 Setup_descriptions();
```

The contents of the data packages may be modified if desired. An element may be added to the water, a mineral removed from the rock etc.

```
 Modify_descriptions();
```

When the details of a project have been defined, a file is created, which contains information on names of the files describing the project. This file is stored in the directory PROJECTS with the name extension .PRJ.

```
 Store_project();
```

Finally files that might still be missing (like pseudo result files and indata file for PHREEQE) are created, and all files needed for the simulation are copied to the work directory.

```
 Prepare_start();
```

During the set-up process, the program determines whether a new simulation is started or an old simulation is continued. (If an old simulation is started over again it is treated as a new simulation). The information is communicated to the calling program, which is normally CRACKER.

```
 exit( Project.status ); /* OLD or NEW */
```

SORPTION.C

The present version of the SORPTION sub program has a limitation that is not present in the other sub programs of CRACKER. The present limits are set by the two define statements

```
 #define MAXLENGTH 110
 #define MAXWIDTH 21
```

Thus a sorption simulation cannot exceed a length of 110 steps (in the water flow direction). Actually the product of length and width is not allowed to be greater than 110 * 21 = 2310. If a greater length is desired and a narrower fracture is acceptable, or vice versa, changes in the statements may be made after which the program has to be re-compiled. The lines may look like

```
 #define MAXLENGTH 256
 #define MAXWIDTH 9
```

Should it be necessary to use larger arrays, it may be necessary to make the program use an array of pointers to structures of type Fracture, rather than one pointer as the present version does. The Fracture structure is very simple as is seen below

```
struct Fracture { short Mineral_no;
                  char  Mineral[10];
                  float Sorbed_qty,
                        Conc,
                        pH,
                        pE;};
```

The program starts by allocating memory for an array of Fracture variables, each of which is supposed to describe one diffusion cell. The pointer to that memory area is supplied to all functions making use of the Fracture structure

```
struct Fracture far *Fract;
if(
   (Fract = ( struct Fracture *)farcalloc(MAXLENGTH * MAXWIDTH,
                sizeof(struct Fracture) )
   ) == NULL )
 {
  printf("Heap full. Decrease MAXLENGTH or MAXWIDTH in source\n");
  printf("code or re-compile with a greater memory model.\n");
  Wait_key();
  exit(0); }
```

The program then asks the user if a hard copy of the display is desired, and in that case under what name. If the program runs under Windows, screen images may be transferred to the Clipboard to get a hard copy. This probably is a better alternative, and the option may be deleted in future versions of the program.

```
Choose_actions();
```

Data and result files from the groundwater simulation are read.

```
Read_initial_cond();
Read_position();
Read_fracture_data( Fract );
```

From the simulation geometry, the sorption diagram is set up.

```
Make_diagram();
```

Finally the sorption simulation is performed as described above-

```
Move_water( Fract );
```

AUXILIARY PROGRAMS

A number of programs that presently are not controlled by the CRACKER program will be found upon the distribution disk. Some of them will be commented here.

The **ENDMIX.EXE** program makes a mixture of all the different waters leaving the fracture. It thus simulates the sampling procedure, in which water is pumped from a fracture into some vessel for analytic purposes. Further it may be used to simulate the mixing that occurs where different fractures meet.

HACKER.SRP is an old version of the sorption handling, in which sorption is performed parallel with the water chemistry simulation. If more than trace quantities of a sorbed element has to be used, this program may have to be modified and used. It will however be very slow.

The **INSTALL.BAT** is used to install CRACKER on the hard disk. CRACKER may be uninstalled by the program REMOVE.EXE. If this program is placed in the DOS directory, it may be used for other purposes as well. REMOVE is recursive program that removes a specified directory with all its content of files and sub directories. It is used in the following way:

```
>REMOVE dirname
```

PRINTSCREEN is a program that transfers a stored screen image to an Epson Fx80 compatible dot matrix printer. The program asks for the name of the file into which the bitmap has been stored and the desired picture quality (single, double or quadruple density). If CRACKER is run in the DOS environment under WINDOWS, the Clipboard device is probably a better alternative. In future versions of CRACKER, the PRINTSCREEN program may be taken away.

PRINTER.H is an include file containing information necessary to compile the PRINTSCREEN.C subprogram.

STDSCRN.BAT restores the character mode of the screen. It may be needed if the CRACKER program is cancelled in a non standard way. Then the screen may be left in graphic mode, which might cause trouble with other programs.

UNISORP.C may be compiled and used to simulate sorption upon a hypothetical fracture surface in a rock where all mineral grains are microscopic. As a result all minerals from which the rock is composed should be expected to be present in each diffusion cell. Such a simulation is meaningful only if the minerals are forming a set of compatible minerals.

REFERENCES

1. D L Parkhurst, D C Thorstenson, L N Plummer, "PHREEQE - a computer program for geochemical calculations", U. S. Geological Survey, Water Resources Investigations, report USGS/WRI-80-96 (1980, 1985 revision)

2. B Allard, "Sorption of actinides in granitic rock", SKBF/KBS (Swedish Nuclear Fuel and Waste Management Co) TR 82-21 (1982)

3. I. Puigdomènech, A. T. Emrén, "Performance of EQ3/6, PHREEQE and Solgaswater on Geochemical Path Calculations Involving Redox and Solid Phase Changes", SKI (Swedish Nuclear Power Inspectorate) TR 90:16 (1990)

Appendix A: CRACKER.C

```
/*
 *    CRACKER version 1992 - 05
 *
 *       by   Allan T Emrén
 *            Dept Nuclear Chemistry
 *            Chalmers University of Technology
 *            S-41296 Göteborg
 *            SWEDEN
 */

#include "ALLAN.H"
#include "SCREEN.H"
#include <PROCESS.H>

#define SUCCESS 1
#define NEED_HELP 2
#define FAILED 0
#define CRACK_END 3
#define STOP 2

struct Proj_description {short status;    /* OLD, NEW or STOP */
                         char  name[20],
                               header[20],
                               water[20],
                               sorpdescr[20],
                               rock[20],
                               diagram[20],
                               sorpdiagram[20];
                        }
                        Project;

void Current_project_name(),
     Decide_action(),
     Delete_project(),
     Read_project(),
     Water_chemistry(),
     Save_old_result(),
     Start_up();



void main()
{
 Project.status = NEW;

 while( Project.status == NEW )
 {
  Start_up();
  if( ( Project.status = spawnl( P_WAIT, "SETUP.EXE", "SETUP.EXE", NULL ) )
        == FAILED )
  {
   printf("Setup program failed\n");
   exit(0);
  }
  Current_project_name();
  Read_project();
  Decide_action();
 }
}
```

Appendix A: CRACKER.C

```
void Start_up()
{
 short choice;


 /* Want to save old result? */
 Inscreen.x = 65;                   /* X-position of input fields      */
 Inscreen.y = 19;                   /* Y-position of first input field */
 Inscreen.items = 1;                /* Number of input fields          */

 sprintf(Inscreen.text[0],"n"); /* No, probably saved already */

 Screen_input(
              "..\\screens\\intro"  /* Name of screen picture */
             );

 if( Inscreen.text[0][0] == 'y' ) Save_old_result();
}



void Read_project()
{
 char garbage[15],
      file_name[70];
 FILE *in;

 sprintf(file_name,"projects\\%s.prj", Project.name );
 if( (in = fopen( file_name, "r")) == NULL )
 {
  printf("Could not open file %s\n",file_name);
  exit(0);
 }
 Filetext(Project.header,9,in);
 Filetext(Project.water,9,in);
 Filetext(Project.sorpdescr,9,in);
 Filetext(Project.rock,9,in);
 Filetext(Project.diagram,9,in);
 Filetext(Project.sorpdiagram,9,in);
 fclose(in);
}



void Current_project_name()
{
 FILE *in;

 if( (in = fopen( "current.prj", "r")) == NULL )
 {
  printf("Could not open file current.prj\n");
  exit(0);
 }
 Filetext(Project.name,9,in);
}



void Save_old_result()
{
 Current_project_name();
 printf("Name: %s\n",Project.name);
 printf("Saving geometry\n");
 sprintf(DOS_command,"copy position.dat projects\\%s.pos",
          Project.name );
 system(DOS_command);

 printf("Saving PHREEQE indata file\n");
 sprintf(DOS_command,"copy infil.dat waters\\%s.phr",
          Project.name );
 system(DOS_command);

 printf("Saving global results\n");
 sprintf(DOS_command,"copy result.dat results\\%s.res",
          Project.name );
 system(DOS_command);

 printf("Saving local results\n");
 sprintf(DOS_command,"copy oldslice.dat results\\%s.old",
          Project.name );
 system(DOS_command);
 sprintf(DOS_command,"copy newslice.dat results\\%s.new",
          Project.name );
 system(DOS_command);
}


void Decide_action()
{
 short choice;

 /* Make choice of direction files */
 do
 {
  Inscreen.x = 10;                    /* X-position of input fields      */
  Inscreen.y = 12;                    /* Y-position of first input field */
  Inscreen.items = 7;                 /* Number of input fields          */

  sprintf(Inscreen.text[0],"Water chemistry simulation");
  sprintf(Inscreen.text[1],"Display the simulation result");
  sprintf(Inscreen.text[2],"Sorption simulation");
  sprintf(Inscreen.text[3],"Save the project result");
  sprintf(Inscreen.text[4],"Delete the project");
  sprintf(Inscreen.text[5],"Modify properties or start a new project");
  sprintf(Inscreen.text[6],"Quit");

  switch( choice = Screen_choice( "..\\screens\\actions" ) )
  {
   case 0:
    Water_chemistry();
   break;

   case 1:
    spawnl( P_WAIT, "DISPLAY.EXE", "DISPLAY.EXE", NULL );
   break;

   case 2:
    spawnl( P_WAIT, "SORPTION.EXE", "SORPTION.EXE", NULL );
   break;

   case 3:
    Save_old_result();
   break;

   case 4:
    Delete_project();
   break;
  }
 }
 while( (choice < 5) || (choice == 7) );

 Project.status = NEW;
 if( choice == 4 ) Delete_project();
 if( choice == 6 ) Project.status = OLD; /* Quit or OK has been chosen */
}
```

Appendix A: CRACKER.C

```
void Water_chemistry()
{
 short result;

 system("del utdata");
 printf("PHREEQE\n");
 spawnl( P_WAIT, "PHREEQE.EXE", "PHREEQE.EXE", NULL );

 printf("HACKER\n");
 while( ( result = spawnl( P_WAIT, "HACKER.EXE", "HACKER.EXE", NULL ) )
          != CRACK_END )
 {
  switch( result )
  {
   case NEED_HELP:
    GiveHelp:
    printf("Helping PHREEQE\n");
    spawnl( P_WAIT, "HELPIT.EXE", "HELPIT.EXE", NULL );
    system("del INFIL.DAT");
    system("ren INFIL.HLP *.DAT");

    system("del utdata");
    printf("PHREEQE\n");
    spawnl( P_WAIT, "PHREEQE.EXE", "PHREEQE.EXE", NULL );

    printf("HACKER\n");
    if( spawnl( P_WAIT, "HACKER.EXE", "HACKER.EXE", NULL ) == NEED_HELP )
    {
     printf("FAILED TO HELP PHREEQE! PLEASE TRY MANUALLY.\n");
     exit(0);
    }

   case SUCCESS:
    system("del utdata");
    printf("PHREEQE\n");
    spawnl( P_WAIT, "PHREEQE.EXE", "PHREEQE.EXE", NULL );
  }
 }
 printf("End of simulation\nPress ENTER to continue.");
 Wait_key();
}


void Delete_project()
{
 char DOS_command[70];

 system( "copy projects\\default.cur current.prj" );
 sprintf(DOS_command,"del results\\%s.*", Project.name );
 system(DOS_command);
 sprintf(DOS_command,"del projects\\%s.*", Project.name );
 system(DOS_command);

 /* Restore default data */
 system("copy projects\\DEFAULT.pos position.dat");
 system("copy waters\\DEFAULT.phr infil.dat");
 system("copy results\\DEFAULT.res result.dat");
 system("copy results\\DEFAULT.old oldslice.dat");
 system("copy results\\DEFAULT.new newslice.dat");
}
```

Appendix B: SETUP.C

```
/* Version 92-05 */
#include "ALLAN.H"
#include "SCREEN.H"

#define FAILURE 0
#define SUCCESS 1
#define MAX_NUMBER_OF_MINERALS 50
#define MAXELEMENT 31

FILE *In,
     *Out;

struct SolutionData { char  Comment[130],
                            ElementName[31][5];
                      float Concentration[31],
                            pH,
                            pE,
                            Temperature;
                      short Components,      /* Number of components */
                                             /* in the solution      */
                            ElementNo[31];
                    }
                    Solution;

char Element[31][6]; /* Max 4 characters in element names */

struct Proj_description {short status;    /* OLD or NEW */
                          char  name[20],
                                header[20],  /* PHREEQE header */
                                water[20],
                                sorpdescr[20],
                                rock[20],
                                diagram[20],
                                sorpdiagram[20];
                        }
                        Project;

struct Proj_geometry { char   text[10][45],   /* Descriptions */
                              mineral[12];
                       short  length,         /* In steps */
                              width,
                              x_step,
                              y_step;
                       long   last_random;
                       double temperature,    /*   C        */
                              temp_gradient,  /*   K/m      */
                              speed,          /*   m/year   */
                              step_length;    /*   m        */
                     }
                     Geom;

struct MineralData
       {
        char  Full_Name[82],
              PHREEQE_Name[12];
        float Nucleation_Prob,
              Frequency;
       };

struct RockData
       {
        char   Comment[120];
        short  Number_Of_Minerals;
        struct MineralData Mineral[MAX_NUMBER_OF_MINERALS];
        short  Mixing_Matrix[MAX_NUMBER_OF_MINERALS]
                            [MAX_NUMBER_OF_MINERALS];
       }
       Rock;
```

Appendix B: SETUP.C

```
struct SorpData
       {
        char element[5];
        float startconc;
       }
       Sorption;

struct Display {
                 short Curves,        /* Max 20     */
                       Maxtime;       /* Time steps */
                 char  Diagram_descr[30],
                       Y_unit_descr[30],
                       Element[MAXELEMENT][5];
                 float XLeft,
                       XRight,
                       XMark,  /* Distance between marks  */
                       XLabel, /* Distance between labels */
                       YDown,
                       YUp,
                       YMark,
                       YLabel,
                       ZDown,
                       ZUp,
                       ZMark,
                       ZLabel,
                       Conc[MAXELEMENT];
               } Screen[6];

short Diagram_type,
      X_unitno;

void Choose_project(),
     Find_element_numbers(),
     Give_mineral(),
     Make_head(),
     Make_PHREEQE_infile(),
     Make_rock_file(),
     Make_solution(),
     Make_sorp_diagram_file(),
     Make_sorption_file(),
     Make_water_file(),
     Make_water_diagram_file(),
     Modify_descriptions(),
     Modify_rock(),
     Modify_sorp_diagram(),
     Modify_sorption(),
     Modify_water(),
     Modify_water_diagram(),
     New_rock(),
     New_sorp_diagram(),
     New_sorption(),
     New_water_diagram(),
     Prepare_start(),
     Read_element_data(),
     Read_geometry(),
     Read_project(),
     Read_rock(),
     Read_sorp_diagram(),
     Read_sorption(),
     Read_water(),
     Read_water_diagram(),
     Setup_descriptions(),
     Setup_geometry(),
     Store_project();
```

Appendix B: SETUP.C

```
void main()
{
 Choose_project();
 Setup_geometry();
 Setup_descriptions();
 Modify_descriptions();
 Store_project();
 Prepare_start();
 exit( Project.status ); /* OLD or NEW */
}



void Choose_project()
{
 short choice;

 Inscreen.items = 1;
 Inscreen.x = 30;
 Inscreen.y = 1;
 Inscreen.status = OLD;

 sprintf(Inscreen.text[0],"%s","Make choice of old or new project.");

 sprintf(Project.name,"%s",
         Screen_file("projects\\*.prj") );

 if( (Project.status = Inscreen.status) == NEW )
 {
  sprintf(DOS_command,"copy projects\\%s.* projects\\%s.*",
          Inscreen.prototype,Project.name );
  system(DOS_command);
 }
 Read_project();
}



void Setup_geometry()
{
 short choice = 0;
 char file_name[70];
 FILE *out;

 Read_geometry();

 Inscreen.x = 50;                    /* X-position of input fields      */
 Inscreen.y = 10;                    /* Y-position of first input field */
 Inscreen.items = 10;                /* Number of input fields          */

 sprintf(Inscreen.text[choice++],"%d",Geom.length);
 sprintf(Inscreen.text[choice++],"%d",Geom.width);
 sprintf(Inscreen.text[choice++],"%d",Geom.x_step);
 sprintf(Inscreen.text[choice++],"%d",Geom.y_step);
 sprintf(Inscreen.text[choice++],"%d",Geom.last_random);
 sprintf(Inscreen.text[choice++],"%s",Geom.mineral);
 sprintf(Inscreen.text[choice++],"%f",Geom.temperature);
 sprintf(Inscreen.text[choice++],"%f",Geom.temp_gradient);
 sprintf(Inscreen.text[choice++],"%f",Geom.speed);
 sprintf(Inscreen.text[choice++],"%f",Geom.step_length);

 Screen_input(
             "..\\screens\\geometry"  /* Name of screen picture */
            );

 sprintf(file_name,"projects\\%s.pos", Project.name );
```

```
  if( (out = fopen( file_name, "w")) == NULL )
  {
   printf("Could not open file %s\n",file_name);
   printf("Press ENTER\n");
   Wait_key();
   exit(0);
  }
  choice = 0;
  fprintf(out,"%s\n",Geom.text[choice]);
  fprintf(out,"%d\n",Inscreen.number[choice++]);
  fprintf(out,"%s\n",Geom.text[choice]);
  fprintf(out,"%d\n",Inscreen.number[choice++]);
  fprintf(out,"%s\n",Geom.text[choice]);
  fprintf(out,"%d\n",Inscreen.number[choice++]);
  fprintf(out,"%s\n",Geom.text[choice]);
  fprintf(out,"%d\n",Inscreen.number[choice++]);
  fprintf(out,"%s\n",Geom.text[choice]);
  fprintf(out,"%ld\n",Inscreen.number[choice++]);
  fprintf(out,"%s\n",Geom.text[choice]);
  fprintf(out,"%s\n",Inscreen.text[choice++]);
  fprintf(out,"%s\n",Geom.text[choice]);
  fprintf(out,"%f\n",Inscreen.real[choice++]);
  fprintf(out,"%s\n",Geom.text[choice]);
  fprintf(out,"%f\n",Inscreen.real[choice++]);
  fprintf(out,"%s\n",Geom.text[choice]);
  fprintf(out,"%f\n",Inscreen.real[choice++]);
  fprintf(out,"%s\n",Geom.text[choice]);
  fprintf(out,"%f\n",Inscreen.real[choice]);
  fclose(out);

  Read_geometry();
}



void Read_geometry()
{
 char file_name[70];
 short i = 0;
 FILE *in;

 sprintf(file_name,"projects\\%s.pos", Project.name );
 if( (in = fopen( file_name, "r")) == NULL )
 {
  printf("Could not open file %s\n",file_name);
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 }
 Filetext(Geom.text[i++],42,in);
 Geom.length = (short)Fileinput(in);
 Filetext(Geom.text[i++],42,in);
 Geom.width = (short)Fileinput(in);
 Filetext(Geom.text[i++],42,in);
 Geom.x_step = (short)Fileinput(in);
 Filetext(Geom.text[i++],42,in);
 Geom.y_step = (short)Fileinput(in);
 Filetext(Geom.text[i++],42,in);
 Geom.last_random = (long)Fileinput(in);
 Filetext(Geom.text[i++],42,in);
 Filetext(Geom.mineral,10,in);
 Filetext(Geom.text[i++],42,in);
 Geom.temperature = Fileinput(in);
 Filetext(Geom.text[i++],42,in);
 Geom.temp_gradient = Fileinput(in);
 Filetext(Geom.text[i++],42,in);
 Geom.speed = Fileinput(in);
 Filetext(Geom.text[i++],42,in);
 Geom.step_length = Fileinput(in);
 fclose(in);
}


void Read_project()
{
 char garbage[15],
      file_name[70];
 FILE *in;

 sprintf(file_name,"projects\\%s.prj", Project.name );
 if( (in = fopen( file_name, "r")) == NULL )
 {
  printf("Could not open file %s\n",file_name);
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 }
 Filetext(Project.header,12,in);
 Filetext(Project.water,12,in);
 Filetext(Project.sorpdescr,12,in);
 Filetext(Project.rock,12,in);
 Filetext(Project.diagram,12,in);
 Filetext(Project.sorpdiagram,12,in);
 fclose(in);
}


void Setup_descriptions()
{
 short choice;

 Read_element_data();   /* from the data base */
 Read_water();          /* water data file */

 /* Make choice of direction files */
 Direction:
 Inscreen.x = 47;                    /* X-position of input fields      */
 Inscreen.y = 9;                     /* Y-position of first input field */
 Inscreen.items = 7;                 /* Number of input fields          */
 Inscreen.status = OLD;

 sprintf(Inscreen.text[0],"%s",Project.name);
 sprintf(Inscreen.text[1],"%s",Project.water);
 sprintf(Inscreen.text[2],"%s",Project.rock);
 sprintf(Inscreen.text[3],"%s",Project.header);
 sprintf(Inscreen.text[4],"%s",Project.sorpdescr);
 sprintf(Inscreen.text[5],"%s",Project.diagram);
 sprintf(Inscreen.text[6],"%s",Project.sorpdiagram);

 while( (choice = Screen_choice( "..\\screens\\main" ) ) < 7 )
 {
  Close_screen();
  Inscreen.x = 30;                    /* X-position of text fields */
  Inscreen.y = 1;                     /* Y-position of text fields */
  Inscreen.items = 1;                 /* Number of text fields     */
```

```
switch( choice )
{
 case 0:
  Open_screen();
  Clear_screen();
  printf("The project name cannot be changed at this stage!\n%s\n",
         "Press ENTER to continue.");
  Wait_key();
  goto Direction;

 case 1:  /* He wants another kind of water for the project */

  sprintf(Inscreen.text[0],"Make choice of old or new water.");
  sprintf(Project.water,"%s", Screen_file("waters\\*.wat") );

  if( Inscreen.status == NEW )  /* A new name chosen. Copy prototype */
  {                             /* files to the new name.            */
   sprintf(DOS_command,"copy waters\\%s.wat waters\\%s.wat",
           Inscreen.prototype, Project.water );
   system(DOS_command);

   Read_water();        /* And make sure that correct mineral is used */
   Make_water_file();
  }
  goto Direction;  /* Change anything else */

 case 2:  /* He wants another kind of rock for the project */

  sprintf(Inscreen.text[0],"Make choice of old or new rock.");
  sprintf(Project.rock,"%s", Screen_file("rocks\\*.rck") );

  if( Inscreen.status == NEW )  /* A new name chosen. Copy prototype */
  {                             /* files to the new name.            */
   sprintf(DOS_command,"copy rocks\\%s.* rocks\\%s.*",
           Inscreen.prototype, Project.rock );
   system(DOS_command);
  }
  goto Direction;

 case 3:  /* He wants another kind of header for the project */

  sprintf(Inscreen.text[0],"Make choice of old or new header.");
  sprintf(Project.header,"%s", Screen_file("headers\\*.hed") );

  if( Inscreen.status == NEW )  /* A new name chosen. Copy prototype */
  {                             /* files to the new name.            */
   sprintf(DOS_command,"copy headers\\%s.hed headers\\%s.hed",
           Inscreen.prototype, Project.header );
   system(DOS_command);
  }
  goto Direction;

 case 4:  /* He wants another kind of sorption description */

  sprintf(Inscreen.text[0],"Make choice of old or new sorption descr.");
  sprintf(Project.sorpdescr,"%s", Screen_file("sorpdata\\*.srd") );

  if( Inscreen.status == NEW )  /* A new name chosen. Copy prototype */
  {                             /* files to the new name.            */
   sprintf(DOS_command,"copy sorpdata\\%s.srd sorpdata\\%s.srd",
           Inscreen.prototype, Project.sorpdescr );
   system(DOS_command);
  }
  goto Direction;

 case 5: /* He wants another kind of diagram for the project */

  sprintf(Inscreen.text[0],"Make choice of old or new diagram
     description.");
```

```
    sprintf(Project.diagram,"%s", Screen_file("diagrams\\*.sol") );

    if( Inscreen.status == NEW )   /* A new name chosen. Copy prototype */
    {                              /* files to the new name.            */
     sprintf(DOS_command,"copy diagrams\\%s.sol diagrams\\%s.sol",
             Inscreen.prototype, Project.diagram );
     system(DOS_command);
    }
    goto Direction;

   case 6:  /* He wants another kind of sorption diagram */

    sprintf(Inscreen.text[0],"Make choice of old or new sorption
     diagram.");
    sprintf(Project.sorpdiagram,"%s", Screen_file("diagrams\\*.srp") );

    if( Inscreen.status == NEW )   /* A new name chosen. Copy prototype */
    {                              /* files to the new name.            */
     sprintf(DOS_command,"copy diagrams\\%s.srp diagrams\\%s.srp",
             Inscreen.prototype, Project.sorpdiagram );
     system(DOS_command);
    }
    goto Direction;
  }
 }
 Close_screen();
}



void Modify_descriptions()
{
 short choice;

 Direction:       /* Make choice of subject */
 Inscreen.x = 20;                    /* X-position of input fields      */
 Inscreen.y = 12;                     /* Y-position of first input field */
 Inscreen.items = 5;                 /* Number of input fields          */

 sprintf(Inscreen.text[0],"Water properties");
 sprintf(Inscreen.text[1],"Rock properties");
 /* sprintf(Inscreen.text[2],"%s",Project.header); Future extension */
 sprintf(Inscreen.text[2],"Sorbed element");
 sprintf(Inscreen.text[3],"Water diagram");
 sprintf(Inscreen.text[4],"Sorption diagram");

 while( (choice = Screen_choice( "..\\screens\\modify" ) ) < 5 )
 {
  Inscreen.x = 30;                    /* X-position of text fields */
  Inscreen.y = 1;                     /* Y-position of text fields */
  Inscreen.items = 1;                 /* Number of text fields     */

  switch( choice )
  {
   case 0:
    Modify_water();
    goto Direction;

   case 1:
    Modify_rock();
    goto Direction;

   case 2:
    Modify_sorption();
    goto Direction;
    case 3:
     Modify_water_diagram();
     goto Direction;

    case 4:
     Modify_sorp_diagram();
     goto Direction;
  }
 }
}



void Read_element_data()
{
 short elementno,
       maxlength=128;

 char string[130],
      part[20],
      *found;

 if( (In = fopen("LIBRARY.DAT", "r")) == NULL)
 {
  printf("Could not find library file\n");
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 }
 if( Search_text("ELEMENTS") == SUCCESS )
 {
  do                            /* Read element numbers and names */
  {
   fgets(string,maxlength,In);
   found = string + 10;         /* Position of element number */

   stccpy( part,
           found,
           3 );

   elementno = atoi(part);
   stccpy( Element[ elementno ], /* Read element name */
           string,
           4 );
  }
  while( (string[11] != 32) && (strlen(string) > 15) );
 }
 fclose(In);
}



void Read_water()
{
 char string[120],
      file_name[40];

 short component,
       maxcomp,
       maxlength;

 maxlength=118;
 sprintf(file_name,"waters\\%s.wat", Project.water );
 if( (In = fopen( file_name, "r")) == NULL )
 {
  printf("Could not open file *%s*\n",file_name);
  printf("Press ENTER\n");
  Wait_key();
  exit(0); }
 fgets(Solution.Comment,118,In); /* Header line */
 Trim(Solution.Comment);
 fgets(string,maxlength,In); /* Mineral. Use Geom.mineral instead */
 fgets(string,maxlength,In);
 Solution.pH = atof(string);
 fgets(string,maxlength,In);
 Solution.pE = atof(string);
 fgets(string,maxlength,In);
 Solution.Temperature = atof(string);
 fgets(string,maxlength,In); /* Number of elements */
 maxcomp = atoi(string);
 Solution.Components = maxcomp;

 for( component=1; component <= maxcomp; component++ )
 {
  fgets(string,maxlength,In); /* Ex:  Na 1.51e-2 */
  stccpy( Solution.ElementName[component], string, 2 );
  Solution.Concentration[ component ] = atof( string+2 );
 }
 fclose(In);
 Find_element_numbers();
}



void Modify_water()
{
 char name[9];
 short key,
       component;

 /* First a screen for input of general water properties */

 Clear_screen();
 Inscreen.x = 19;                /* X-position of input fields      */
 Inscreen.y = 6;                 /* Y-position of first input field */
 Inscreen.items = 4;             /* Number of input fields          */

 sprintf(Inscreen.text[0],"%s",Solution.Comment);
 sprintf(Inscreen.text[1],"%f",Solution.pH);
 sprintf(Inscreen.text[2],"%f",Solution.pE);
 sprintf(Inscreen.text[3],"%f",Solution.Temperature);

 Screen_input(
              "..\\screens\\water1"  /* Name of screen picture */
             );

 sprintf(Solution.Comment,"%s",Inscreen.text[0]);
 Solution.pH = Inscreen.real[1];
 Solution.pE = Inscreen.real[2];
 Solution.Temperature = Inscreen.real[3];

 /* Now make the element screen #1 */
 Open_screen();

 ListElements:
 Clear_screen();
 printf("\nElements present in the solution.\n");
 printf("OBS! Carbon has to be present!\n");
 printf("If you do not wish any C, give a small conc, like 1e-20\n\n\n");
 for( component=1; component <= Solution.Components; component++ )
 {
  printf(" %s",Solution.ElementName[component]);
 }
 printf("\n\n\nRemove an element (r) or add an element (a).\n");
 printf("If present set of elements is OK, press ENTER");
 if( (key = Wait_key()) != RETURNKEY)
 {
  switch(key = (char)key)
  {
   case 'a':
    Solution.Components++;
    printf("\nWhich element do you want to add? ");
    gets(Solution.ElementName[Solution.Components]);
    if( Solution.ElementName[Solution.Components][1] < ' ' )
     Solution.ElementName[Solution.Components][1] = ' ';

    if( ( Solution.ElementNo[Solution.Components] =
          Find_an_element( Solution.ElementName[Solution.Components] )
        ) == 0
      )
    {
     printf("\nThe element is not present in the data base!\n");
     printf("To use this element, you have to modify the data base.\n");
     printf("To continue press ENTER");
     Wait_key();
     Solution.Components--;
    }
    break;

   case 'r':
    printf("\nWhich element do you want to remove? ");
    gets(name);
    if( name[1] < ' ' ) name[1] = ' ';
    for( component=1; component <= Solution.Components; component++ )
    {
     if( strncmpi( Solution.ElementName[component], name, 2) == 0 )
      break;
    }
    if( component <= Solution.Components )
    {
     Solution.Components--;
     for( ; component <= Solution.Components; component++ )
     {
      strcpy( Solution.ElementName[component],
              Solution.ElementName[component+1] );
      Solution.Concentration[component] =
       Solution.Concentration[component+1];
     }
    }
  }
  goto ListElements;
 }

 /* Element set OK. Get concentrations */
 Close_screen();
 Inscreen.description_x = 1;         /* X-position of element names     */
 Inscreen.x = 4;                     /* X-position of input fields      */
 Inscreen.y = 1;                     /* Y-position of first input field */
 Inscreen.items = Solution.Components;
 for( component=1; component <= Solution.Components; component++ )
 {
  sprintf( Inscreen.description[component-1],"%s",
           Solution.ElementName[component] );
  sprintf( Inscreen.text[component-1],"%e",
           Solution.Concentration[component] );
 }
 Clear_screen();
 Screen_list( "..\\screens\\water3" );

 for( component=1; component <= Solution.Components; component++ )
  Solution.Concentration[component] = atof( Inscreen.text[component-1] );

 Make_water_file();
}
```

Appendix B: SETUP.C

```
void Find_element_numbers()
{
 short j;

 for( j = 1; j <= Solution.Components; j++ )
  Solution.ElementNo[j] = Find_an_element( Solution.ElementName[j] );
}



Find_an_element( name )
           char *name;
{
 short i;

 for( i = 1; i < 31; i++ )
  if( strncmpi( Element[i], name, 2) == 0 ) return(i);
 return(0);
}



void Make_PHREEQE_infile()
{
 char file_name[40],
      command[80];

 sprintf(file_name,"waters\\%s.phr", Project.name );
 if( (Out = fopen( file_name, "w")) == NULL )
 {
  printf("Could not open file *%s*\n",file_name);
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 }
 Make_head();
 Make_solution( 1 );
 Make_solution( 2 );
 Give_mineral();
 fprintf(Out,"\n");
 fprintf(Out,"END     \n");
 fclose(Out);
}



void Make_head()
{
 char infile[80],     /* Name of header file */
      line[130];      /* Next line from header file */
 short maxlength=128; /* Max line length            */

 char *buf; /* for diagnostic purpose */

 sprintf(infile,"headers\\%s.hed", Project.header );
 if( (In = fopen( infile, "r")) == NULL)
 {
  printf("Could not open file *%s*\n",infile);
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 }
 buf=line;
 while(1) /* Loop is ended elsewhere */
 {
  buf=fgets(line, maxlength, In);
  if( buf == NULL) break;        /* End of file */

  Trim( buf );
  fprintf(Out,"%s\n",line);
 }
 fclose(In);
}


void Make_solution( PHREEQE_no )
             short PHREEQE_no;
{
 short component,
       on_line=0;

 fprintf(Out,"SOLUTION %1d\n",PHREEQE_no);
 fprintf(Out,"%s\n",Solution.Comment);
 fprintf(Out,"%2d  0 0   %9.3f %9.3f %9.3f  1.000\n",
         /* number        pH    pE     T     dens   */
             Solution.Components,
             Solution.pH,
             Solution.pE,
             Solution.Temperature);

 for( component=1; component <= Solution.Components; component++ )
 {
              /*  i4,11.3D */
  fprintf(Out,"  %2d%11.4E", Solution.ElementNo[component],
                             Solution.Concentration[component]
         );

  on_line++;
  if( on_line == 5 ) /* max 5 concentrations pr line */
  {
   if( component <= Solution.Components )
   {
    fprintf(Out,"\n");
    on_line = 0;
   }
  }
 }
 fprintf(Out,"\n");
}


void Give_mineral()
{
 char line[130],      /* Next line from MINERAL file */
      data_file[45],
      directory[] = "MINERALS\\",
      mineral_ext[] = ".MIN",
      *buf;

 short maxlength=128; /* Max line length            */

 strcpy( data_file, directory );
 strcat( data_file,Geom.mineral );
 strcat( data_file, mineral_ext );
 if( (In = fopen( data_file, "r")) == NULL)
 {
  printf(" Cannot open *%s*\n",data_file);
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 }
 fprintf(Out,"MINERALS\n");

 buf=line;
 while(1)
 {
  buf=fgets(line, maxlength, In);
  if( buf == NULL) break;

  if( (buf = strchr(line, 10) ) != NULL)
  {
   *buf = 0;       /* get rid of LINE FEED character */
  }

  if( (buf = strchr(line, 13) ) != NULL)
  {
   *buf = 0;       /* get rid of RETURN character */
  }

  fprintf(Out,"%s\n",line);
 }
 fclose(In);
}


void Make_water_file()
{
 char file_name[40];
 short component;

 sprintf(file_name,"waters\\%s.wat", Project.water );
 if( (Out = fopen( file_name, "w")) == NULL )
 {
  printf("Could not open file *%s*\n",file_name);
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 } fprintf(Out,"%s\n",Solution.Comment);
 fprintf(Out,"%s\n",Geom.mineral);
 fprintf(Out,"%f\n",Solution.pH);
 fprintf(Out,"%f\n",Solution.pE);
 fprintf(Out,"%f\n",Solution.Temperature);
 fprintf(Out,"%d\n",Solution.Components);

 for( component=1; component <= Solution.Components; component++ )
 {
  fprintf(Out,"%s %e\n",Solution.ElementName[ component ],
                        Solution.Concentration[ component ]);
 }
 fclose(Out);
}


void Modify_rock()
{
 Read_rock();
 New_rock();
 Make_rock_file();
}
```

```
void Read_rock()
{
 char string[120],
      file_name[40];

 short i,
       maxlength=118;

 sprintf(file_name,"rocks\\%s.rck", Project.rock );

 if( (In = fopen( file_name, "r")) == NULL )
 {
  printf("Cannot open %s",file_name);
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 }
 fgets( Rock.Comment,119,In);
 Trim( Rock.Comment );
 fgets(string,maxlength,In); /* Skip over descriptor */
 fgets(string,maxlength,In);
 Rock.Number_Of_Minerals = atoi(string);
 for( i = 1; i <= Rock.Number_Of_Minerals; i++ )
 {
  fgets(string,maxlength,In); /* Empty line */
  fgets(string,maxlength,In); /* Skip over descriptor */
  fgets( Rock.Mineral[i].Full_Name,80,In);
  Trim( Rock.Mineral[i].Full_Name );
  fgets(string,maxlength,In); /* Skip over descriptor */
  fgets(string,maxlength,In);
  string[9] = 0;  /* Make sure that it is not too long */
  Trim( string );
  strcpy( Rock.Mineral[i].PHREEQE_Name, string);

  fgets(string,maxlength,In); /* Skip over descriptor */
  fgets(string,maxlength,In);
  Rock.Mineral[i].Nucleation_Prob = atof(string);

  fgets(string,maxlength,In); /* Skip over descriptor */
  fgets(string,maxlength,In);
  Rock.Mineral[i].Frequency = atof(string);
 }
 fclose(In);
}



void New_rock()
{
 char name[9];
 short key,
       component;

 /* First a screen for input of general rock description */

 Clear_screen();
 Inscreen.x = 1;                    /* X-position of input fields      */
 Inscreen.y = 8;                    /* Y-position of first input field */
 Inscreen.items = 1;                /* Number of input fields          */

 sprintf(Inscreen.text[0],"%s", Rock.Comment);

 Screen_input(
             "..\\screens\\rock1"  /* Name of screen picture */
            );

 sprintf( Rock.Comment,"%s",Inscreen.text[0]);
```

```
/* Now make the mineral screen #2 */
Close_screen();
Open_screen();

ListMinerals:
Clear_screen();
printf("Minerals present in the rock:\n\n");

for( component=1; component <= Rock.Number_Of_Minerals;
     component++ )
{
 printf(" %s\n",Rock.Mineral[component].PHREEQE_Name );
}
printf("\nRemove a mineral (r) or add a mineral (a).\n");
printf("If present set of minerals is OK, press ENTER");
if( (key = Wait_key()) != RETURNKEY)
{
 switch(key = (char)key)
 {
  case 'a':
   Rock.Number_Of_Minerals++;
   printf("\nWhich mineral do you want to add? ");
   gets( Rock.Mineral[Rock.Number_Of_Minerals].PHREEQE_Name );
   break;

  case 'r':
   printf("\nWhich mineral do you want to remove? ");
   gets( name );
   Trim_space( name );
   for( component=1;        component <= Rock.Number_Of_Minerals;
        component++ )
   {
    if( strncmpi( Rock.Mineral[component].PHREEQE_Name,
                  name,
                  strlen(name)
                ) == 0
      ) break;
   }
   if( component <= Rock.Number_Of_Minerals )
   {
    Rock.Number_Of_Minerals--;
    for( ; component <= Rock.Number_Of_Minerals; component++ )
    {
     strcpy( Rock.Mineral[component].PHREEQE_Name,
             Rock.Mineral[component+1].PHREEQE_Name );
     strcpy( Rock.Mineral[component].Full_Name,
             Rock.Mineral[component+1].Full_Name );
     Rock.Mineral[component].Frequency
      = Rock.Mineral[component+1].Frequency;
     Rock.Mineral[component].Nucleation_Prob
      = Rock.Mineral[component+1].Nucleation_Prob;
    }
   }
 }
 goto ListMinerals;
}

/* Mineral set OK. Get frequences */
Close_screen();
Inscreen.description_x = 1;        /* X-position of element names     */
Inscreen.x = 15;                   /* X-position of input fields      */
Inscreen.y = 1;                    /* Y-position of first input field */

Inscreen.items = Rock.Number_Of_Minerals;
```

```
 for( component = 1; component <= Rock.Number_Of_Minerals; component++ )
 {
  sprintf( Inscreen.description[component-1],"%s (%)",
           Rock.Mineral[component].PHREEQE_Name );
  sprintf( Inscreen.text[component-1],"%f",
           Rock.Mineral[component].Frequency );
 }
 Clear_screen();
 Screen_list( "..\\screens\\rock3" );

 for( component = 1; component <= Rock.Number_Of_Minerals; component++ )
  Rock.Mineral[component].Frequency = atof( Inscreen.text[component-1] );

 Close_screen();

 for( component = 1; component <= Rock.Number_Of_Minerals; component++ )
 {
  sprintf( Inscreen.description[component-1],"%s",
           Rock.Mineral[component].PHREEQE_Name );
  sprintf( Inscreen.text[component-1],"%s",
           Rock.Mineral[component].Full_Name );
 }

 Clear_screen();
 Screen_list( "..\\screens\\rock4" );

 for( component = 1; component <= Rock.Number_Of_Minerals; component++ )
  sprintf( Rock.Mineral[component].Full_Name,"%s",
           Inscreen.text[component-1]);
}


void Make_rock_file()
{
 char string[120],
      file_name[40];

 short i,
       maxlength=118;

 sprintf(file_name,"rocks\\%s.rck", Project.rock );

 if( ( Out = fopen( file_name, "w")) == NULL )
 {
  printf("Cannot open %s\n",file_name);
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 }
 fprintf( Out,"%s\n", Rock.Comment );
 fprintf( Out,"Number of minerals\n" );
 fprintf( Out,"%d\n", Rock.Number_Of_Minerals );

 for( i = 1; i <= Rock.Number_Of_Minerals; i++ )
 {
  fprintf( Out,"\n" ); /* Empty line */
  fprintf( Out,"Comment\n" );
  fprintf( Out,"%s\n", Rock.Mineral[i].Full_Name );
  fprintf( Out,"PHREEQE name\n" );
  fprintf( Out,"%s\n", Rock.Mineral[i].PHREEQE_Name );
  fprintf( Out,"Nucleation prob\n" );
  fprintf( Out,"%f\n", Rock.Mineral[i].Nucleation_Prob );
  fprintf( Out,"Frequency\n",string,maxlength); /* Skip over descriptor */
  fprintf( Out,"%f\n", Rock.Mineral[i].Frequency );
 }
 fclose( Out );
}
```

```
void Modify_sorption()
{
 Read_sorption();
 New_sorption();
 Make_sorption_file();
}



void Read_sorption()
{
 char string[120],
      file_name[40];

 short i,
       maxlength=118;

 sprintf(file_name,"sorpdata\\%s.srd", Project.sorpdescr );
 if( (In = fopen( file_name, "r")) == NULL )
 {
  printf("Cannot open %s\n",file_name);
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 }
 fgets(string,maxlength,In); /* Skip over descriptor */
 fgets(Sorption.element,5,In);
 Trim(Sorption.element);
 fgets(string,maxlength,In);
 Sorption.startconc = Fileinput( In );
 fclose(In);
}



void New_sorption()
{
 char name[9];
 short key,
       component;

 /* First a screen for input of general rock description */

 Clear_screen();
 Inscreen.x = 35;                    /* X-position of input fields      */
 Inscreen.y = 9;                     /* Y-position of first input field */
 Inscreen.items = 2;                 /* Number of input fields          */

 sprintf(Inscreen.text[0],"%s", Sorption.element);
 sprintf(Inscreen.text[1],"%e", Sorption.startconc);

 Screen_input(
             "..\\screens\\sorption"  /* Name of screen picture */
            );

 sprintf( Sorption.element,"%s",Inscreen.text[0]);
 Sorption.startconc = Inscreen.real[1];
}



void Make_sorption_file()
{
 char string[120],
      file_name[40];

 sprintf(file_name,"sorpdata\\%s.srd", Project.sorpdescr );
 if( ( Out = fopen( file_name, "w")) == NULL )
 {
  printf("Cannot open %s\n",file_name);
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 }
 fprintf( Out,"PHREEQE name of studied nuclide\n" );
 fprintf( Out,"%s\n", Sorption.element );
 fprintf( Out,"Concentration at start of fracture ( M )\n" );
 fprintf( Out,"%e\n", Sorption.startconc );

 fclose( Out );
}



Search_text( desired_text )
       char *desired_text;
{
 short maxlength=128,
       result;

 char string[130];

 Next_record:
  if( fgets(string,maxlength,In) == NULL)    /* End of file found */
  {
   result = FAILURE;
   goto Stop_searching;
  }

  if( strstr( string, "TERMINATED" ) != NULL)
  {
   result = FAILURE;
   goto Stop_searching;
  }
  if( strstr( string, desired_text ) == NULL)
   goto Next_record;

  result = SUCCESS;

 Stop_searching:
 return( result );
}



void Modify_water_diagram()
{
 Read_water_diagram();
 New_water_diagram();
 Make_water_diagram_file();
}



void Read_water_diagram() /* Read the file NAME.SOL to find out what kind
*/
{                         /* of diagram to use, which intervals, marks and
*/
 short i,                 /* labels are desired etc
*/
       element,
       maxlength=118;

 FILE *diagram;
 char string[120],
      file_name[40];

 sprintf(file_name,"diagrams\\%s.sol", Project.diagram );
 if( (diagram = fopen( file_name, "r")) == NULL )
 {
  printf("Cannot open %s\n",file_name);
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 }
 Filetext(string,maxlength,diagram); /* Type descriptions */
 Diagram_type = (short)Fileinput(diagram); /* 1=pH 2=pE 3=pH-pE */
                                  /* 4=Precip-dissol 5=Concentrations */
                                  /* 6=No picture */

 Filetext(string,maxlength,diagram);             /* X-unit descriptions   */
 X_unitno = (short)Fileinput(diagram);           /* 1=step 2=meter 3=year */

 Filetext(string,maxlength,diagram);                      /* X-mark descr */
 Screen[0].XMark =  Fileinput(diagram);

 Filetext(string,maxlength,diagram);                       /* X-lbl descr */
 Screen[0].XLabel =  Fileinput(diagram);

 for( i = 1; i < 6; i++ )
 {
  Filetext(string,maxlength,diagram);                      /* Empty line  */
  Filetext(Screen[i].Diagram_descr,maxlength,diagram);     /* Diagram
                                                              headline */
  if( i == 5 )    /* Concentration diagram */
  {
   Filetext(string,maxlength,diagram);           /* Headline no of curves */
   Screen[i].Curves = (short)Fileinput(diagram);     /* Number of curves */
   for( element = 0; element < Screen[i].Curves; element++ )
    Filetext(Screen[i].Element[element],maxlength,diagram);
                     /* Element name */
   Filetext(string,maxlength,diagram);  /* Empty line */
  }
  Filetext(string, maxlength, diagram); /* Headline bottom y */
  Screen[i].YDown =  Fileinput(diagram);

  Filetext(string,maxlength,diagram); /* Headline Top y */
  Screen[i].YUp =  Fileinput(diagram);
  Filetext(string,maxlength,diagram); /*Headline Mark distance */
  Screen[i].YMark =  Fileinput(diagram);
  Filetext(string,maxlength,diagram); /* Headline lbl dist */
  Screen[i].YLabel =  Fileinput(diagram);
 }
 fclose(diagram);
}


void New_water_diagram()
{
 char name[9],
      key;
 short component,
       choice;

 /* First a screen for input of general diagram description */

 Clear_screen();
 Inscreen.x = 50;                  /* X-position of input fields      */
 Inscreen.y = 11;                  /* Y-position of first input field */
 Inscreen.items = 4;               /* Number of input fields          */
```

Appendix B: SETUP.C

```
/*
 * Next line not in use. Decided at display time in this version.
 * sprintf(Inscreen.text[0],"%d", Diagram_type);
 */
sprintf(Inscreen.text[0],"%d", X_unitno);
sprintf(Inscreen.text[1],"%e", Screen[0].XMark);
sprintf(Inscreen.text[2],"%e", Screen[0].XLabel);

Screen_input( "..\\screens\\watdia0" );

/*
 * Not in use. (Program fossil)
 * Diagram_type = Inscreen.number[0];
 */
X_unitno = Inscreen.number[0];
Screen[0].XMark = Inscreen.real[1];
Screen[0].XLabel = Inscreen.real[2];

Direction:
Inscreen.x = 20;                     /* X-position of input fields       */
Inscreen.y = 5;                      /* Y-position of first input field */
Inscreen.items = 4;                  /* Number of input fields           */

sprintf(Inscreen.text[0],"pH vs X");
sprintf(Inscreen.text[1],"pE vs X");
sprintf(Inscreen.text[2],"Frequency vs pH and pE (3 dim)");
sprintf(Inscreen.text[3],"Concentrations vs X");

if( (choice = Screen_choice( "..\\screens\\watdia1" ) ) < 4 )
{
 Close_screen();
 Inscreen.x = 48;                    /* X-position of text fields */
 Inscreen.y = 7;                     /* Y-position of text fields */
 Inscreen.items = 4;                 /* Number of text fields     */

 switch( choice )
 {
  case 0:  /* He wants another kind of pH - X diagram */

   sprintf(Inscreen.text[0],"%f",Screen[1].YUp);
   sprintf(Inscreen.text[1],"%f",Screen[1].YMark);
   sprintf(Inscreen.text[2],"%f",Screen[1].YLabel);
   sprintf(Inscreen.text[3],"%f",Screen[1].YDown);

   Screen_input( "..\\screens\\watdia11" );

   Screen[1].YUp = Inscreen.real[0];
   Screen[1].YMark = Inscreen.real[1];
   Screen[1].YLabel = Inscreen.real[2];
   Screen[1].YDown = Inscreen.real[3];

   Screen[3].XLeft = Inscreen.real[0];     /* Modify X-axis of      */
   Screen[3].XMark = Inscreen.real[1];     /* freq vs pH and pE too */
   Screen[3].XLabel = Inscreen.real[2];
   Screen[3].XRight = Inscreen.real[3];
   goto Direction;  /* Change anything else */

  case 1:  /* He wants another kind of pE - X diagram */

   sprintf(Inscreen.text[0],"%f",Screen[2].YUp);
   sprintf(Inscreen.text[1],"%f",Screen[2].YMark);
   sprintf(Inscreen.text[2],"%f",Screen[2].YLabel);
   sprintf(Inscreen.text[3],"%f",Screen[2].YDown);
  Screen_input( "..\\screens\\watdial2" );

  Screen[2].YUp = Inscreen.real[0];
  Screen[2].YMark = Inscreen.real[1];
  Screen[2].YLabel = Inscreen.real[2];
  Screen[2].YDown = Inscreen.real[3];
  Screen[3].YUp = Inscreen.real[0];      /* Modify X-axis of      */
  Screen[3].YMark = Inscreen.real[1];    /* freq vs pH and pE too */
  Screen[3].YLabel = Inscreen.real[2];
  Screen[3].YDown = Inscreen.real[3];
  goto Direction;  /* Change anything else */

 case 2:  /* He wants another kind of freq vs pH and pE diagram */

  Inscreen.items = 3;
  Inscreen.x = 55;
  Inscreen.y = 16;

  sprintf(Inscreen.text[0],"%f",Screen[3].YUp);
  sprintf(Inscreen.text[1],"%f",Screen[3].YMark);
  sprintf(Inscreen.text[2],"%f",Screen[3].YLabel);

  Screen_input( "..\\screens\\watdial3" );

  Screen[3].YUp = Inscreen.real[0];
  Screen[3].YMark = Inscreen.real[1];
  Screen[3].YLabel = Inscreen.real[2];
  Screen[3].YDown = 0;

  Inscreen.x = 48;                       /* Restore input screen */
  Inscreen.y = 7;
  Inscreen.items = 4;
  goto Direction;

 case 3:  /* He wants another kind of Conc vs X diagram (type 5) */

  /* Now make an element screen */
  Open_screen();

  ListElements:
   Clear_screen();
   printf("\nElements to be displayed (max 5).\n\n\n");
   for( component = 0; component < Screen[5].Curves; component++ )
    printf( " %s", Screen[ 5 ].Element[ component ] );

   printf("\n\n\nRemove an element (r) or add an element (a).\n");
   printf("If present set of elements is OK, press ENTER");

   if( (key = (char)Wait_key()) != (char)RETURNKEY)
   {
    switch( key )
    {
     case 'a':
      printf("\nWhich element do you want to add? ");
      gets( Screen[ 5 ].Element[ Screen[5].Curves ] );

      /* Check one- or two-character name */
      if( Screen[ 5 ].Element[ Screen[5].Curves ][ 1 ] < ' ' )
       Screen[ 5 ].Element[ Screen[5].Curves ][ 1 ] = ' ';

      /* Convert characters to upper case */
      strupr( Screen[ 5 ].Element[ Screen[5].Curves ] );
      Screen[5].Curves++;
      break;
```

```
       case 'r':
        printf("\nWhich element do you want to remove? ");
        gets(name);
        if( name[1] < ' ' ) name[1] = ' ';

        for( component = 0; component < Screen[5].Curves; component++ )
         if( strncmpi( Screen[5].Element[component], name, 2) == 0 )
          break;

        if( component < Screen[5].Curves )
        {
         Screen[5].Curves--;
         for( ; component < Screen[5].Curves; component++ )
          strcpy( Screen[5].Element[component],
                  Screen[5].Element[component+1] );
        }
      }
      goto ListElements;
     }
    sprintf(Inscreen.text[0],"%f",Screen[5].YUp);
    sprintf(Inscreen.text[1],"%f",Screen[5].YMark);
    sprintf(Inscreen.text[2],"%f",Screen[5].YLabel);
    sprintf(Inscreen.text[3],"%f",Screen[5].YDown);

    Screen_input( "..\\screens\\watdial5" );

    Screen[5].YUp = Inscreen.real[0];
    Screen[5].YMark = Inscreen.real[1];
    Screen[5].YLabel = Inscreen.real[2];
    Screen[5].YDown = Inscreen.real[3];
    goto Direction;
  }
 }
 Close_screen();
}


void Make_water_diagram_file()
{
 short i,
       element;

 FILE *diagram;

 char string[120],
      file_name[40];

 sprintf(file_name,"diagrams\\%s.sol", Project.diagram );
 if( (diagram = fopen( file_name, "w")) == NULL )
 {
  printf("Cannot open %s\n",file_name);
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 }
 fprintf( diagram,"1=pH 2=pE 3=pH vs pE 4 not used 5=conc 6 no picture\n");
 fprintf( diagram,"%d\n",Diagram_type);

 fprintf( diagram,"1=step 2=meter 3=year\n");
 fprintf( diagram,"%d\n",X_unitno);

 fprintf( diagram,"Distance between X-marks\n");
 fprintf( diagram,"%f\n", Screen[0].XMark);

 fprintf( diagram,"Distance between X-labels\n");
 fprintf( diagram,"%f\n", Screen[0].XLabel);
 for( i = 1; i < 6; i++ )
 {
  fprintf( diagram,"\n");                           /* Empty line        */
  fprintf( diagram,"%s\n",Screen[i].Diagram_descr); /* Diagram headline */

  if( i == 5 )    /* Concentration diagram */
  {
   fprintf( diagram,"Number of curves\n");
   fprintf( diagram,"%d\n",Screen[i].Curves);
   for( element = 0; element < Screen[i].Curves; element++ )
    fprintf( diagram,"%s\n",Screen[i].Element[element]);

   fprintf( diagram,"\n");  /* Empty line */
  }
  fprintf( diagram,"Bottom y or z\n");
  fprintf( diagram,"%f\n",Screen[i].YDown);

  fprintf( diagram,"Max y or z\n");
  fprintf( diagram,"%f\n", Screen[i].YUp);

  fprintf( diagram,"Y or Z mark dist\n");
  fprintf( diagram,"%f\n",Screen[i].YMark);

  fprintf( diagram,"Y or Z label dist\n");
  fprintf( diagram,"%f\n", Screen[i].YLabel);
 }
 fclose(diagram);
}



void Modify_sorp_diagram()
{
 Read_sorp_diagram();
 New_sorp_diagram();
 Make_sorp_diagram_file();
}



void Read_sorp_diagram() /* Read the file NAME.SRP to find out what kind
*/
{                         /* of diagram to use, which intervals, marks and
*/
 short i,                 /* labels are desired etc
*/
       element,
       maxlength=118;

 FILE *diagram;

 char string[120],
      file_name[40];

 sprintf(file_name,"diagrams\\%s.srp", Project.sorpdiagram );
 if( (diagram = fopen( file_name, "r")) == NULL )
 {
  printf("Cannot open %s\n",file_name);
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 }
 /*Type (1=Package 2=Snapshots 3=Break-through 4, 5 not use 6=No picture)*/
 Filetext(string,maxlength,diagram);
 Diagram_type = (short)Fileinput(diagram);

 /*X-unit (1=step 2=meter 3=year)*/
 Filetext(string,maxlength,diagram);
 X_unitno = (short)Fileinput(diagram);
 /*Mark dist at X-axis*/
 Filetext(string,maxlength,diagram);
 Screen[0].XMark = Fileinput(diagram);

 /*Label dist at X-axis*/
 Filetext(string,maxlength,diagram);
 Screen[0].XLabel =  Fileinput(diagram);

 for( i = 1; i < 6; i++ )
 {
  Filetext(string,maxlength,diagram);                    /* Empty line   */
  Filetext(Screen[i].Diagram_descr,maxlength,diagram);
  Filetext(string,maxlength,diagram);  /* Headline Bottom y */
  Screen[i].YDown =  Fileinput(diagram);

  Filetext(string,maxlength,diagram); /* Headline Top y */
  Screen[i].YUp =  Fileinput(diagram);
  Filetext(string,maxlength,diagram); /*Headline Mark distance */
  Screen[i].YMark =  Fileinput(diagram);
  Filetext(string,maxlength,diagram); /* Headline lbl dist */
  Screen[i].YLabel =  Fileinput(diagram);

  if( i == 2 )
  {
   Filetext(string,maxlength,diagram); /* Headline Curve interval */
   Screen[i].Curves =  Fileinput(diagram);
   Filetext(string,maxlength,diagram); /* Headline Max time */
   Screen[i].Maxtime =  Fileinput(diagram);  }
 }
 fclose(diagram);
}



void New_sorp_diagram()
{
 char name[9],
      key;

 short choice;

 /* First a screen for input of general diagram description */

 Clear_screen();
 Inscreen.x = 50;                    /* X-position of input fields      */
 Inscreen.y = 11;                    /* Y-position of first input field */
 Inscreen.items = 4;                 /* Number of input fields          */

 sprintf(Inscreen.text[0],"%d", Diagram_type);
 sprintf(Inscreen.text[1],"%d", X_unitno);
 sprintf(Inscreen.text[2],"%e", Screen[0].XMark);
 sprintf(Inscreen.text[3],"%e", Screen[0].XLabel);

 Screen_input( "..\\screens\\srpdia0" );

 Diagram_type = Inscreen.number[0];
 X_unitno = Inscreen.number[1];
 Screen[0].XMark = Inscreen.real[2];
 Screen[0].XLabel = Inscreen.real[3];

 Direction:
 Inscreen.x = 20;                    /* X-position of input fields      */
 Inscreen.y = 5;                     /* Y-position of first input field */
 Inscreen.items = 2;                 /* Number of input fields          */
```

Appendix B: SETUP.C

```
/*Type (1=Package 2=Snapshots 3=Break-through 4, 5 not use 6=No picture)*/
sprintf(Inscreen.text[0],"Follow a water package");
sprintf(Inscreen.text[1],"Snapshots");
/* sprintf(Inscreen.text[2],"Break through curves"); Not yet in use!
 * sprintf(Inscreen.text[3],"Concentrations vs X and time");
 */
if( (choice = Screen_choice( "..\\screens\\srpdial" ) ) < 2 )
{
 Close_screen();
 Inscreen.x = 48;                    /* X-position of text fields */
 Inscreen.y = 7;                     /* Y-position of text fields */
 Inscreen.items = 4;                 /* Number of text fields     */

 switch( choice )
 {
  case 0:  /* He wants another kind of water package diagram */

   sprintf(Inscreen.text[0],"%f",Screen[1].YUp);
   sprintf(Inscreen.text[1],"%f",Screen[1].YMark);
   sprintf(Inscreen.text[2],"%f",Screen[1].YLabel);
   sprintf(Inscreen.text[3],"%f",Screen[1].YDown);

   Screen_input( "..\\screens\\srpdial1" );

   Screen[1].YUp = Inscreen.real[0];
   Screen[1].YMark = Inscreen.real[1];
   Screen[1].YLabel = Inscreen.real[2];
   Screen[1].YDown = Inscreen.real[3];

   goto Direction;  /* Change anything else */

  case 1:  /* He wants another kind of snapshot diagram */

   Inscreen.items = 6;                 /* Number of text fields      */
   sprintf(Inscreen.text[0],"%f",Screen[2].YUp);
   sprintf(Inscreen.text[1],"%f",Screen[2].YMark);
   sprintf(Inscreen.text[2],"%f",Screen[2].YLabel);
   sprintf(Inscreen.text[3],"%f",Screen[2].YDown);
   sprintf(Inscreen.text[4],"%f",(float)Screen[2].Curves
                                 * Geom.step_length
                                 * Geom.speed        );
   sprintf(Inscreen.text[5],"%f",(float)Screen[2].Maxtime
                                 * Geom.step_length
                                 * Geom.speed        );

   Screen_input( "..\\screens\\srpdial2" );

   Screen[2].YUp = Inscreen.real[0];
   Screen[2].YMark = Inscreen.real[1];
   Screen[2].YLabel = Inscreen.real[2];
   Screen[2].YDown = Inscreen.real[3];
   Screen[2].Curves = Inscreen.real[4] / Geom.step_length / Geom.speed;
   Screen[2].Maxtime = Inscreen.real[5] / Geom.step_length / Geom.speed;

   goto Direction;  /* Change anything else */

  case 2:  /* He wants another kind of break through diagram */
  /* Never happens! */
   sprintf(Inscreen.text[0],"%f",Screen[3].YUp);
   sprintf(Inscreen.text[1],"%f",Screen[3].YMark);
   sprintf(Inscreen.text[2],"%f",Screen[3].YLabel);
   sprintf(Inscreen.text[3],"%f",Screen[3].YDown);

   Screen_input( "..\\screens\\srpdial3" );
```

```
    Screen[3].YUp = Inscreen.real[0];
    Screen[3].YMark = Inscreen.real[1];
    Screen[3].YLabel = Inscreen.real[2];
    Screen[3].YDown = Inscreen.real[3];
    goto Direction;

   case 3:  /* He wants another kind of diagram */
   /* Never happens! */

    goto Direction;
  }
 }
}

void Make_sorp_diagram_file()
{
 short i,
       element;

 FILE *diagram;

 char string[120],
      file_name[40];

 sprintf(file_name,"diagrams\\%s.srp", Project.sorpdiagram );
 if( (diagram = fopen( file_name, "w")) == NULL )
 {
  printf("Cannot open %s\n",file_name);
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 }
 fprintf( diagram,"%s\n",
 "Type (1=Package 2=Snapshots 3=Break-through 4, 5 not use 6=No picture)");
 fprintf( diagram,"%d\n",Diagram_type);

 fprintf( diagram,"1=step 2=meter 3=year\n");
 fprintf( diagram,"%d\n",X_unitno);

 fprintf( diagram,"Distance between X-marks\n");
 fprintf( diagram,"%f\n", Screen[0].XMark);

 fprintf( diagram,"Distance between X-labels\n");
 fprintf( diagram,"%f\n", Screen[0].XLabel);

 for( i = 1; i < 6; i++ )
 {
  fprintf( diagram,"\n");                          /* Empty line       */
  fprintf( diagram,"%s\n",Screen[i].Diagram_descr);  /* Diagram headline */

  fprintf( diagram,"Bottom y\n");
  fprintf( diagram,"%f\n",Screen[i].YDown);

  fprintf( diagram,"Top y\n");
  fprintf( diagram,"%f\n", Screen[i].YUp);

  fprintf( diagram,"Y mark dist\n");
  fprintf( diagram,"%f\n",Screen[i].YMark);

  fprintf( diagram,"Y label dist\n");
  fprintf( diagram,"%f\n", Screen[i].YLabel);
  if( i == 2 )
  {
   fprintf( diagram,"Curve interval\n");
   fprintf( diagram,"%d\n", Screen[i].Curves);
   fprintf( diagram,"Max time (in time steps)\n");
   fprintf( diagram,"%d\n", Screen[i].Maxtime);
  }
 }
 fclose(diagram);
}

void Store_project()
{
 char garbage[15],
      file_name[70];

 FILE *out;

 if( (out = fopen( "CURRENT.PRJ", "w")) == NULL )
 {
  printf("Could not open file CURRENT.PRJ\n");
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 }
 fprintf( out,"%s\n", Project.name );  /* Save name of current project */
 fclose( out );

 sprintf(file_name,"projects\\%s.prj", Project.name );
 if( (out = fopen( file_name, "w")) == NULL )
 {
  printf("Could not open file %s\n",file_name);
  printf("Press ENTER\n");
  Wait_key();
  exit(0);
 }
 fprintf( out,"%s\n",Project.header);
 fprintf( out,"%s\n",Project.water);
 fprintf( out,"%s\n",Project.sorpdescr);
 fprintf( out,"%s\n",Project.rock);
 fprintf( out,"%s\n",Project.diagram);
 fprintf( out,"%s\n",Project.sorpdiagram);
 fclose(out);
}



void Prepare_start()
{
 short step,
       component;

 char file_name[70];

 FILE *out;

 printf("Setting up data files needed for the simulation\n\n");
 if( (Geom.x_step == 1) && (Geom.y_step == 1) ) /* Make initial results */
 {
  Project.status = NEW;
  printf("Creating local pseudo result files..\n");
  sprintf(DOS_command,"copy results\\default.res results\\%s.res",
                                                   Project.name );
  system(DOS_command);            /* Make default result file embryo */
  sprintf(DOS_command,"copy results\\default.new results\\%s.new",
                                                   Project.name );
  system(DOS_command);            /* Make default newslice file embryo */
  printf("Create PHREEQE file for the new project\n");
  Read_water();
  Make_PHREEQE_infile();

  sprintf(file_name,"results\\%s.old", Project.name ); /* Make oldslice */
  if( (out = fopen( file_name, "w")) == NULL )
  {
   printf("Could not open file %s\n",file_name);
   printf("Press ENTER\n");
   Wait_key();
   exit(0);
  }
  fprintf( out,"Length step no 0\n");

  for( step = 1; step <= Geom.width; step++ )
  {
   fprintf( out,"Width step no %d\n", step);
   fprintf( out,"%s\n",Geom.mineral);
   fprintf( out,"pH=%e\n",Solution.pH);
   fprintf( out,"pE=%e\n",Solution.pE);
   fprintf( out,"Number of elements: %d\n",Solution.Components);
   for( component=1; component <= Solution.Components; component++ )
    fprintf( out,"%s    %e\n",Solution.ElementName[ component ],
                           Solution.Concentration[ component ]);
  }
  fclose( out );
 }
 else
 {
  Project.status = OLD;
 }
 printf("Header file\n");
 sprintf(DOS_command,"copy headers\\%s.hed header.dat", Project.header );
 system(DOS_command);
 printf("Convergence parameters\n");
 system( "copy headers\\knobs.dat *.*" );
 printf("PHREEQE indata file\n");
 sprintf(DOS_command,"copy waters\\%s.phr infil.dat", Project.name );
 system(DOS_command);
 printf("Geometry file\n");
 sprintf(DOS_command,"copy projects\\%s.pos position.dat", Project.name );
 system(DOS_command);
 printf("Global and local result files\n");
 sprintf(DOS_command,"copy results\\%s.res result.dat", Project.name );
 system(DOS_command);
 sprintf(DOS_command,"copy results\\%s.old oldslice.dat", Project.name );
 system(DOS_command);
 sprintf(DOS_command,"copy results\\%s.new newslice.dat", Project.name );
 system(DOS_command);
 printf("Sorption file\n");
 sprintf(DOS_command,"copy sorpdata\\%s.srd sorpdscr.dat",
        Project.sorpdescr );
 system(DOS_command);
 printf("Diagram descriptions\n");
 sprintf(DOS_command,"copy diagrams\\%s.srp sorpdia.dat",
        Project.sorpdiagram );
 system(DOS_command);
 sprintf(DOS_command,"copy diagrams\\%s.sol diagram.dat", Project.diagram
       );
 system(DOS_command);
 printf("Rock data\n");
 sprintf(DOS_command,"copy rocks\\%s.rck rock.dat", Project.rock );
 system(DOS_command);
}
```

Appendix C: HACKER.C

```
/*
 *    HACKER  2d version 1992 - 05
 *       by   Allan T Emren
 *            Dept Nuclear Chemistry
 *            Chalmers University of Technology
 *            S-41296 Goteborg
 *            SWEDEN
 */

#include "allan.h"

#define FAILURE 0
#define SUCCESS 1
#define NEED_HELP 2
#define CRACK_END 3

#define MAX_NUMBER_OF_MINERALS 50
#define MAXWIDTH 50

struct SolutionData { char  Name[130],
                            ElementName[31][5];
                      float Concentration[31],
                            pH,
                            pE,
                            Temperature;
                      short Components,      /* Number of components */
                                             /* in the solution      */
                            ElementNo[31],
                            Iterations;
                    }
                    Solution[MAXWIDTH];

struct MineralData
       {
        char  Full_Name[82],
              PHREEQE_Name[12];
        float Nucleation_Prob,
              Frequency;
       };

struct RockData
       {
        char   Name[120];
        short  Number_Of_Minerals;
        struct MineralData Mineral[MAX_NUMBER_OF_MINERALS];
        short Mixing_Matrix[MAX_NUMBER_OF_MINERALS]
                           [MAX_NUMBER_OF_MINERALS];
       }
       Rock;

char Element[31][5]; /* Max 4 characters in element names */

struct PositionData { short XMaxStep
                            XStepNo,
                            YMaxStep,
                            YStepNo;
                      long  LastRandom;
                      char  MineralName[30];
                      float Temperature,
                            TemperatureStep,
                            WaterSpeed,
                            StepLength;
                  }
                  Spot;

FILE *In,
     *Out,
     *Resultfile;
```

Appendix C: HACKER.C

```
void Give_mineral(),
     Make_solution(),
     Make_head(),
     Make_next_cell(),
     Make_next_slice(),
     Find_element_numbers(),
     Save_desired_result(),
     Read_old_slice(),
     Read_rock_data(),
     Read_position_data(),
     Read_element_data(),
     Save_position(),
     Trim(),
     Append_result();

double atoDf();



void main()
{
 Read_element_data();    /* from the data base */

 if( Read_PHREEQE_result() == FAILURE ) exit( NEED_HELP );

 Read_position_data();
 Save_desired_result();

 Read_rock_data();
 Make_next_cell();
 Save_position();

 if( Spot.XStepNo > Spot.XMaxStep )
  exit(CRACK_END);
 else
  exit( SUCCESS );
}



void Read_element_data()
{
 short elementno,
       maxlength=128;

 char string[130],
      part[20],
      *found;

 if( (In = fopen("LIBRARY.DAT", "r")) == NULL) exit(CRACK_END);

 if( Search_text("ELEMENTS") == SUCCESS )
 {
  do                        /* Read element numbers and names */
  {
   fgets(string,maxlength,In);
   found = string + 10;     /* Position of element number */

   stccpy( part, found, 3 );

   elementno = atoi(part);
   stccpy( Element[ elementno ], /* Read element name */
         string,
         2 );
  }
  while( (string[16] != 32) && (strlen(string) > 15) );
 }
 fclose(In);
}
```

Appendix C: HACKER.C

```
Read_PHREEQE_result()
{
 short i=1,
       result,
       maxlength=128;

 char string[130],
      part[20],
      *found;

 if( (In = fopen("UTDATA", "r")) == NULL) exit(CRACK_END);

 if( ( result=Search_text("PHASE BOUNDARIES") ) == SUCCESS )
 {
     /* Do nothing about it yet */

  Search_text("TOTAL MOLALITIES");
     /* Go to first concentration */

  fgets(string,maxlength,In); /* underline */
  fgets(string,maxlength,In); /* empty     */
  fgets(string,maxlength,In); /* headline  */
  fgets(string,maxlength,In); /* empty     */

  do                          /* Read molalities */
  {
   fgets(string,maxlength,In);
   found = string + 16;       /* Position of element name */

   stccpy( Solution[0].ElementName[i],
          found,
          3 );
   found = string + 29;       /* Position of element concentration */

   stccpy( part,
          found,
          16 );

   Solution[0].Concentration[i] = atoDf(part);
   i++;
  }
  while( (string[16] != 32) && (strlen(string) > 15) );

  Solution[0].Components=i-2;
  Search_text("DESCRIPTION OF SOLUTION");

     /* Go to pH-line */

  fgets(string,maxlength,In); /* empty */
  fgets(string,maxlength,In);
  found = string + 44;       /* Position of pH value */

  stccpy( part,
         found,
         9 );

  Solution[0].pH = atoDf(part);
  fgets(string,maxlength,In);
  found = string + 44;       /* Position of pE value */

  stccpy( part,
         found,
         9 );

  Solution[0].pE = atoDf(part);
     /* Go to temperature */
  fgets(string,maxlength,In); /* eH               */
  fgets(string,maxlength,In); /* activity         */
  fgets(string,maxlength,In); /* ionic strength   */
  fgets(string,maxlength,In); /* temperature      */

     /* Go to # of iterations */

  fgets(string,maxlength,In); /* electrical balance */
  fgets(string,maxlength,In); /* thor               */
  fgets(string,maxlength,In); /* total alcalinity   */

  fgets(string,maxlength,In);
  found = string + 44;       /* Position of iteration value */

  stccpy( part,
         found,
         9 );

  Solution[0].Iterations = atoi(part);
 }
 fclose(In);
 if( result == SUCCESS )
  Find_element_numbers( 0 ); /* 0 = solution number for previous cell */
 return( result );
}


void Read_position_data()
{
 char string[130];

 short maxlength=128;

 if( (In = fopen("POSITION.DAT", "r")) == NULL)
 {
  printf("Cannot open POSITION.DAT");
  exit(CRACK_END);
 }
 fgets(string,maxlength,In);                /* Max X stepno */
 fgets(string,maxlength,In);
 Spot.XMaxStep = atoi(string);

 fgets(string,maxlength,In);                /* Max Y stepno */
 fgets(string,maxlength,In);
 Spot.YMaxStep = atoi(string);

 fgets(string,maxlength,In);                /* Current X stepno */
 fgets(string,maxlength,In);
 Spot.XStepNo = atoi(string);

 fgets(string,maxlength,In);                /* Current Y stepno */
 fgets(string,maxlength,In);
 Spot.YStepNo = atoi(string);

 fgets(string,maxlength,In);                /* Last random */
 fgets(string,maxlength,In);
 Spot.LastRandom = atol(string);

 fgets(string,maxlength,In);                /* Next mineral */
 fgets(string,maxlength,In);
 Trim(string);
 strcpy(Spot.MineralName,string);

 fgets(string,maxlength,In);                /* Temperature */
 fgets(string,maxlength,In);
 Spot.Temperature = atof(string);
 fgets(string,maxlength,In);              /* Temperature gradient (K/m) */
 fgets(string,maxlength,In);
 Spot.TemperatureStep = atof(string);

 fgets(string,maxlength,In);               /* Water speed (m/year) */
 fgets(string,maxlength,In);
 Spot.WaterSpeed = atof(string);

 fgets(string,maxlength,In);               /* Step length (m) */
 fgets(string,maxlength,In);
 Spot.StepLength = atof(string);
 fclose(In);                              /* Water speed and step length */
}                                         /* ignored in this version     */


void Read_rock_data()
{
 char string[130];

 short i,
       maxlength=128;

 if( (In = fopen("ROCK.DAT", "r")) == NULL)
 {
  printf("Cannot open ROCK.DAT");
  exit(CRACK_END);
 }
 fgets( Rock.Name,119,In);
 Trim( Rock.Name );
 fgets(string,maxlength,In); /* Skip over descriptor */
 fgets(string,maxlength,In);
 Rock.Number_Of_Minerals = atoi(string);
 for( i=0; i < Rock.Number_Of_Minerals; i++ )
 {
  fgets(string,maxlength,In); /* Empty line */
  fgets(string,maxlength,In); /* Skip over descriptor */
  fgets( Rock.Mineral[i].Full_Name,80,In);
  Trim( Rock.Mineral[i].Full_Name );
  fgets(string,maxlength,In); /* Skip over descriptor */
  fgets(string,maxlength,In);
  string[9] = 0;  /* Make sure that it is not too long */
  Trim( string );
  strcpy( Rock.Mineral[i].PHREEQE_Name, string);

  fgets(string,maxlength,In); /* Skip over descriptor */
  fgets(string,maxlength,In);
  Rock.Mineral[i].Nucleation_Prob = atof(string);

  fgets(string,maxlength,In); /* Skip over descriptor */
  fgets(string,maxlength,In);
  Rock.Mineral[i].Frequency = atof(string) / 100.0;/*Percent in the file */
 }
 fclose(In);
}


void Save_position()
{
 short i;

 if( (Out = fopen("POSITION.DAT", "w")) == NULL)
 {
  printf("Cannot open POSITION.DAT for output");
  exit(CRACK_END);
 }
 fprintf(Out,"Fracture length (steps)\n");
 fprintf(Out,"%u\n", Spot.XMaxStep);
 fprintf(Out,"Fracture width (steps)\n");
 fprintf(Out,"%u\n", Spot.YMaxStep);
 fprintf(Out,"Length StepNo\n");
 fprintf(Out,"%u\n", Spot.XStepNo);
 fprintf(Out,"Width StepNo\n");
 fprintf(Out,"%u\n", Spot.YStepNo);
 fprintf(Out,"LastRandom\n");
 fprintf(Out,"%u\n", Spot.LastRandom);
 fprintf(Out,"Next mineral\n");
 fprintf(Out,"%s\n", Spot.MineralName);
 fprintf(Out,"Temperature (C)\n%f\n",Spot.Temperature);
 fprintf(Out,"Temperature gradient (K/m)\n%f\n",Spot.TemperatureStep);
 fprintf(Out,"Water speed (m/year)\n%f\n",Spot.WaterSpeed);
 fprintf(Out,"Step length (m)\n%f\n",Spot.StepLength);
 fclose(Out);
}


Search_text( desired_text )
       char *desired_text;
{
 short maxlength=128,
       result;

 char string[130];

 Next_record:
  if( fgets(string,maxlength,In) == NULL)    /* End of file found */
  {
   result = FAILURE;
   goto Stop_searching;
  }

  if( strstr( string, "TERMINATED" ) != NULL)
  {
   result = FAILURE;
   goto Stop_searching;
  }
  if( strstr( string, desired_text ) == NULL)
   goto Next_record;

  result = SUCCESS;

 Stop_searching:
 return( result );
}


void Find_element_numbers( s )
                  short s;
{
 short i,
       j;

 for( j = 1; j <= Solution[s].Components; j++ )
 {
  for( i = 1; i < 31; i++ )
  {
   if(  strncmpi( Element[i], Solution[s].ElementName[j], 2 ) == 0 )
    Solution[s].ElementNo[j] = i;
  }
 }
}
```

```
void Save_desired_result()
{
 short component,
       i = 1;

 Append_result("NEWSLICE.DAT"); /* Open result file for append */

 fprintf(Resultfile,"Width step no %d\n",Spot.YStepNo);
 fprintf(Resultfile,"%s\n", Spot.MineralName);
 fprintf(Resultfile,"pH=%f \npE=%f\n",Solution[0].pH,
                                  Solution[0].pE
       );
 printf("\nX-step %d   ",Spot.XStepNo);
 printf("Y-step %d   ",Spot.YStepNo);
 printf("%s\n", Spot.MineralName);
 printf("pH=%f   pE=%f\n",Solution[0].pH,Solution[0].pE );

 fprintf(Resultfile,"Number of elements: %u\n", Solution[0].Components );
 for( component=1;
      component <= Solution[0].Components;
      component++ )
 {
  fprintf(Resultfile, "%s %11.3E\n",  Solution[0].ElementName[component],
                                      Solution[0].Concentration[component]
          );
  printf("%s %9.3E   ", Solution[0].ElementName[component],
                        Solution[0].Concentration[component]
        );
  if( (i++ % 4) == 0 ) printf("\n");
 }
 fclose(Resultfile);
 printf("\n");
}



void Make_next_cell()
{
 short leftcell,
       rightcell,
       slice_parity;


 Spot.YStepNo++;
 if( Spot.YStepNo > Spot.YMaxStep )
  Make_next_slice();

 Read_old_slice();
                                         /*            / \ / \ / \       */
 slice_parity = Spot.XStepNo % 2;        /*    /|\    | 1 | 2 | 3 |      */
                                         /*     |      \ / \ / \ / \     */
                                         /*     |       | 1 | 2 | 3 |    */
                                         /*     |      / \ / \ / \ /     */
                                         /*     |     | 1 | 2 | 3 |      */
                                         /*            \ / \ / \ /       */
 leftcell = Spot.YStepNo - slice_parity;
 leftcell = ( leftcell >= 1 )? leftcell
                             : Spot.YMaxStep;

 rightcell = Spot.YStepNo - slice_parity + 1;
 rightcell = ( rightcell > Spot.YMaxStep )? 1
                                          : rightcell;
 Find_element_numbers( leftcell );
 Find_element_numbers( rightcell );

 if( (Out = fopen("INFIL.DAT", "w")) == NULL) exit(CRACK_END);
 Make_head();
 Make_solution( 1 , leftcell );
 Make_solution( 2 , rightcell );
 Give_mineral();
 fprintf(Out,"\n");
 fprintf(Out,"END      \n");
 fclose(Out);
}



void Make_next_slice()
{
 char *buf,             /* for diagnostic purpose */
       line[130];       /* Next line from header file */
 short maxlength=128;   /* Max line length           */

 system("del OLDSLICE.DAT");  /* This slice is already saved */
 system("ren NEWSLICE.DAT OLDSLICE.DAT");


 Append_result("RESULT.DAT"); /* Open large result file for append */


 if( (In = fopen("OLDSLICE.DAT", "r")) == NULL)
 {
  printf("Could not open OLDSLICE.DAT\n");
  exit(CRACK_END);
 }

 buf=line;
 while(1)
 {
  buf=fgets(line, maxlength, In);
  if( buf == NULL) break;           /* End of file */

  Trim( line );

  fprintf( Resultfile, "%s\n", line );
 }
 fclose(In);
 fclose(Resultfile);

 Spot.XStepNo++;

 if( (Out = fopen("NEWSLICE.DAT", "w")) == NULL)
 {
  printf("Could not open a new NEWSLICE.DAT file\n");
  exit(CRACK_END);
 }
 fprintf(Out,"Length step no %d\n",Spot.XStepNo);
 fclose(Out);

 Spot.Temperature += Spot.TemperatureStep;
 Spot.YStepNo = 1;
}



void Read_old_slice()
{
 static char string[120];

 static short i,
             component,
             maxlength = 118;
```

```
if( (In = fopen("OLDSLICE.DAT", "r")) == NULL) exit(CRACK_END);
fgets(string,maxlength,In); /* X-Step No */
for( i = 1; i <= Spot.YMaxStep; i++ )
{
 fgets(string,maxlength,In); /* Y-Step No */
 fgets(string,maxlength,In); /* Mineral */
 fgets(string,maxlength,In); /* pH */
 Solution[i].pH = atof(string+3);

 fgets(string,maxlength,In); /* pE */
 Solution[i].pE = atof(string+3);
 fgets(string,maxlength,In);
 Solution[i].Components = atoi(string+20);

 for( component = 1;
      component <= Solution[i].Components;
      component++ )
  {   fgets(string,maxlength,In); /* Ex:  Na 1.51e-2 */
   stccpy( Solution[i].ElementName[component],
           string,
           3 );

   Solution[i].Concentration[ component ] = atof( string+5 );
  }
 }
 fclose(In);}



void Make_head()
{
 char line[130];      /* Next line from header file */
 short maxlength=128; /* Max line length            */

 char *buf; /* for diagnostic purpose */

 if( (In = fopen("HEADER.DAT", "r")) == NULL)
 {
  printf("Could not open HEADER.DAT\n");
  exit(CRACK_END);
 }

 buf=line;
 while(1)
 {
  buf=fgets(line, maxlength, In);
  if( buf == NULL) break;        /* End of file */

  Trim( line );

  fprintf(Out,"%s\n",line);
 }
 fclose(In);
}



void Make_solution( PHREEQE_no, pos )
            short PHREEQE_no, pos;
{
 short component,
       on_line=0;

 Solution[pos].Temperature = Spot.Temperature;
 fprintf(Out,"SOLUTION %1d\n",PHREEQE_no);
 fprintf(Out,"Groundwater from cell %d.\n",pos);
 fprintf(Out,"%2d  0 0   %9.3f %9.3f %9.3f  1.000\n",
        /* antal    pH        pE       T      dens    */
           Solution[pos].Components,
           Solution[pos].pH,
           Solution[pos].pE,
           Solution[pos].Temperature);

 for( component=1;
      component <= Solution[pos].Components;
      component++ )
 {
            /*  i4,11.3D */
  fprintf(Out," %2d %11.4E", Solution[pos].ElementNo[component],
                             Solution[pos].Concentration[component]
          );

  on_line++;
  if( on_line == 5 ) /* max 5 concentrations pr line */
  {
   if( component <= Solution[pos].Components )
   {
    fprintf(Out,"\n");
    on_line = 0;
   }
  }
 }
 fprintf(Out,"\n");
}



void Give_mineral()
{
 char line[130],      /* Next line from header file */
      data_file[25],
      directory[] = "MINERALS\\",
      mineral_ext[] = ".MIN",
      *buf;

 short maxlength=128, /* Max line length           */
       next_mineral=0;

 float random_number;

 Start_random_gen( Spot.LastRandom );
 random_number = Random_float();
 while( random_number > 0.0 )
 {
  random_number -= Rock.Mineral[ next_mineral++ ].Frequency;
 }
 next_mineral--;

 Spot.LastRandom = Last_random;
 Trim_space(Rock.Mineral[ next_mineral ].PHREEQE_Name);
 strcpy( Spot.MineralName, Rock.Mineral[ next_mineral ].PHREEQE_Name );

 strcpy( data_file, directory );
 strcat( data_file, Rock.Mineral[ next_mineral ].PHREEQE_Name );
 strcat( data_file, mineral_ext );
 if( (In = fopen( data_file, "r")) == NULL)
 {
  printf("Number of minerals: %d\n",Rock.Number_Of_Minerals);
  printf("Number of next mineral: %d\n", next_mineral );
  printf(" Cannot open **%s**\n",data_file);
  exit(CRACK_END);
 }
 fprintf(Out,"MINERALS\n");
 buf=line;
 while(1)
 {
  buf=fgets(line, maxlength, In);
  if( buf == NULL) break;

  if( (buf = strchr(line, 10) ) != NULL)
  {
   *buf = 0;      /* get rid of LINE FEED character */
  }

  if( (buf = strchr(line, 13) ) != NULL)
  {
   *buf = 0;      /* get rid of RETURN character */
  }

  fprintf(Out,"%s\n",line);
 }
 fclose(In);
}


double atoDf(string) /* Change FORTRAN D-form into e-form */
char string[];
{
 char c='D',
      *position;

 position=strchr(string,c);
 *position='e';
 return(atof(string));
}


void Append_result( name) /* System specific procedure due to bug in */
            char *name; /* Borland Turbo C                        */
{
 long pos;

 if( ( Resultfile = fopen( name, "r+")) == NULL)
 {
  printf("Cannot open the result file!");
  exit(CRACK_END);
 }
 fseek( Resultfile, -1, SEEK_END ); /* Position of last character */

 if( fgetc(Resultfile) == 10 )    /* move to next position */
 {
  fseek( Resultfile, 0, SEEK_END ); /* Position of EOF character */
 }
 else                /* has bypassed EOF */
 {
  fseek( Resultfile, -1, SEEK_END ); /* Position of EOF character */
 }
}
```

Appendix D: HELPIT.C

```
/*
 *     HELPIT version 1990 - 06
 *       by   Allan T Emren
 *            Dept Nuclear Chemistry
 *            Chalmers University of Technology
 *            S-41296 Goteborg
 *            SWEDEN
 */

#include "allan.h"

FILE *in,
      *knobs,
      *out;

void Copy_header(),
     Copy_knobs(),
     Copy_rest();

void main()
{
 if( (in = fopen("INFIL.DAT", "r")) == NULL) exit(0);
 if( (knobs = fopen("KNOBS.DAT", "r")) == NULL) exit(0);
 if( (out = fopen("INFIL.HLP", "w")) == NULL) exit(0);
 Copy_header();
 Copy_knobs();
 Copy_rest();

 fclose(in);
 fclose(knobs);
 fclose(out);
}



void Copy_header()
{
 short maxlength=128,
       i;

 char string[130],
      *buf;

 for( i=0; i<2; i++ )
 {
  fgets(string,maxlength,in);
  if( (buf = strchr(string, 10) ) != NULL)
  {
   *buf = 0;      /* get rid of LINE FEED character */
  }
  if( (buf = strchr(string, 13) ) != NULL)
  {
   *buf = 0;      /* get rid of RETURN character */
  }

  fprintf(out,"%s\n",string);
 }
}
```

Appendix D: HELPIT.C

```
void Copy_knobs()
{
 short maxlength=128,
       i;
 char string[130],
      *buf;

 for( i=0; i<2; i++ )
 {
  fgets(string,maxlength,knobs);
  if( (buf = strchr(string, 10) ) != NULL)
  {
   *buf = 0;      /* get rid of LINE FEED character */
  }
  if( (buf = strchr(string, 13) ) != NULL)
  {
   *buf = 0;      /* get rid of RETURN character */
  }
  fprintf(out,"%s\n",string);
 }
}



void Copy_rest()
{
 char line[130],      /* Next line from header file */
      *buf;
 short maxlength=128; /* Max line length            */

 buf=line;

 while(1)
 {
  buf=fgets(line, maxlength, in);
  if( buf == NULL) break;
  if( (buf = strchr(line, 10) ) != NULL)
  {
   *buf = 0;      /* get rid of LINE FEED character */
  }
  if( (buf = strchr(line, 13) ) != NULL)
  {
   *buf = 0;      /* get rid of RETURN character */
  }

  fprintf(out,"%s\n",line);
 }
}
```

Appendix E: DISPLAY.C

```c
/*
 *    DISPLAY version 1992 - 05
 *       by   Allan T Emren
 *            Dept Nuclear Chemistry
 *            Chalmers University of Technology
 *            S-41296 Goteborg
 *            SWEDEN
 */

#include "allan.h"
#include "screen.h"

#define MAXELEMENT 20
#define MAXCLASSES 20
#define MAXWIDTH 50

FILE *in,
     *pos,
     *diagram,
     *out;

float X,
      pH[ MAXWIDTH ],
      pE[ MAXWIDTH ],
      Conc[MAXELEMENT][MAXWIDTH],
      Temperature,
      TemperatureStep,
      WaterSpeed,
      StepLength;

float XSize_3d,  /* Will be moved to ALLAN.H */
      YSize_3d,
      ZSize_3d,
      XLeft_3d,
      XRight_3d,
      YDown_3d,
      YUp_3d,
      ZDown_3d,
      ZUp_3d;

char Key,
     File_name[15],
     Diagram_descr[65][130],
     NextMineral[30],
     Diagram_head[6][15] = {
                              " ",
                              "pH",
                              "pE",
                              "pH-pE",
                              "Prec-dissol",
                              "Log conc"
                            },

     X_unit[4][15] = {
                        " ",
                        "steps",
                        "meters",
                        "years",
                      };
```

Appendix E: DISPLAY.C

```
short Length,
      Width,
      XStepNo,
      YStepNo,
      pH_pE_Sample[40][40],
      File_size, /* Number of lines in file describing displays */

      X_unitno,
      Type;   /* Screen type (pH, pE etc)
               * pH             = 1
               * pE             = 2
               * pH-pE (3dim)   = 3
               * Precipitation  = 4
               * Concentrations = 5
               * No picture     = 6  */

struct Display {
               short Curves;         /* Max 20 */
               char  Diagram_descr[30],
                     Y_unit_descr[30],
                     Element[MAXELEMENT][5];
               float XLeft,
                     XRight,
                     XMark,  /* Distance between marks  */
                     XLabel, /* Distance between labels */
                     YDown,
                     YUp,
                     YMark,
                     YLabel,
                     ZDown,
                     ZUp,
                     ZMark,
                     ZLabel,
                     Conc[MAXELEMENT];
              } Screen[6];

void Read_position(),
     Define_pH_pE_axis(),
     Setup_2d_display(),
     Setup_3d_display(),
     Read_result(),
     Draw_pH_pE_sample(),
     Next_point(),
     Sample_pH_pE(),
     Choose_type(),
     Read_diagram_descr(),
     Set_marks(),
     Set_3d_marks(),
     Set_labels(),
     Set_3d_labels(),
     Coordinate_system_3d(),
     Coordinates_3d(),
     Move_3d_to(),
     Line_3d_to(),
     Trim();
```

Appendix E: DISPLAY.C

```
void main()
{
 while( TRUE )
 {
  Choose_type();
  if( Type == 6 ) exit(0);

  Read_position();
  if( XStepNo < 4 ) exit(0); /* There is nothing interesting to show */

  if( Type != 3 )
  {
   Setup_2d_display();
   Read_result();           /* and draw curves */
  }
  else
  {
   Define_pH_pE_axis();
   Setup_3d_display( "pH", "pE", "Freq" );
   Read_result();
   Draw_pH_pE_sample();
  }
  Wait_key();

  if( File_name[0] != (char)0 )
   Dump_screen( File_name );

  Close_screen();
 }
}



void Read_position()
{
 char string[130];

 short i,
       maxlength=128;

 if( (pos = fopen("POSITION.DAT", "r")) == NULL)
 {
  printf("Cannot open POSITION.DAT");
  exit(0);
 }
 fgets(string,maxlength,pos);
 fgets(string,maxlength,pos);
 Length = atoi(string);

 fgets(string,maxlength,pos);
 fgets(string,maxlength,pos);
 Width = atoi(string);

 fgets(string,maxlength,pos);
 fgets(string,maxlength,pos);
 XStepNo = atoi(string);

 fgets(string,maxlength,pos);
 fgets(string,maxlength,pos);
 YStepNo = atoi(string);

 fgets(string,maxlength,pos);
 fgets(string,maxlength,pos);
/*
 *LastSpot.LastRandom = atol(string);
 */
 fgets(string,maxlength,pos);      /* Next mineral */
 fgets(NextMineral,maxlength,pos);
 fgets(string,maxlength,pos); /* Temperature */
 fgets(string,maxlength,pos);
 Temperature = atof(string);

 fgets(string,maxlength,pos); /* Temperature step */
 fgets(string,maxlength,pos);
 TemperatureStep = atof(string);

 fgets(string,maxlength,pos); /* Water speed (m/year) */
 fgets(string,maxlength,pos);
 WaterSpeed = atof(string);

 fgets(string,maxlength,pos); /* Step length (m) */
 fgets(string,maxlength,pos);
 StepLength = atof(string);

/* fgets(string,maxlength,pos); Thickness
 * fgets(string,maxlength,pos);
 */
 fclose(pos);
}



void Setup_2d_display()
{
 /* Declared in allan.h: XLeft, YDown, XRight, YUp, XSize, YSize */
 XLeft = 0;

 switch( X_unitno )                     /* Unit along X-axis: */
 {
  case 1:                                           /* Step */
    XRight = XStepNo + 5;
   break;

  case 2:                                           /* Meter */
    XRight = ( XStepNo + 5 ) * StepLength;
   break;

  case 3:                                           /* Year */
    XRight = ( XStepNo + 5 ) * StepLength / WaterSpeed;
   break;
 }

 Open_screen();
 Coordinate_system( XLeft, Screen[Type].YDown, XRight, Screen[Type].YUp );

 Set_marks( XLeft,   Screen[Type].XMark,  YDown,   Screen[Type].YMark );
        /* X-start      X-dist           Y-start       Y-dist */

 Set_labels(XLeft,
            Screen[Type].XMark,
            Screen[Type].XLabel,
            YDown,
            Screen[Type].YMark,
            Screen[Type].YLabel,
            4 /* string length */);

 printf("\n %s vs pos", Diagram_head[Type] );

 /* The normal X-unit is step. Temporarily, it has been changed to meter
  * or year, for drawing the axis correctly. Now, make a logical change
  * in thhe length of the X-axis, to return to the step as unit.
  */
 switch( X_unitno )                         /* Unit along X-axis: */
 {
  case 1:                                              /* Step */
   break;

  case 2:                                              /* Meter */
    XRight /= StepLength;
   break;

  case 3:                                              /* Year */
    XRight /= StepLength / WaterSpeed;
   break;
 }
 Coordinates( XLeft, Screen[Type].YDown, XRight, Screen[Type].YUp );
}

void Setup_3d_display( x_descr, y_descr, z_descr )
                   char x_descr[MAXCLASSES],
                        y_descr[MAXCLASSES],
                        z_descr[MAXCLASSES];
{
 Open_screen();
 Coordinate_system_3d( Screen[Type].XLeft,
                       Screen[Type].YDown,
                       Screen[Type].ZDown,
                       Screen[Type].XRight,
                       Screen[Type].YUp,
                       Screen[Type].ZUp
                     );

 Set_3d_marks( Screen[Type].XLeft, Screen[Type].XMark,
               Screen[Type].YDown, Screen[Type].YMark,
               Screen[Type].ZDown, Screen[Type].ZMark
             );

 Move_3d_to( Screen[Type].XLeft + 0.45 * XSize_3d,
             Screen[Type].YUp + 0.14 * YSize_3d,
             Screen[Type].ZDown );
 outtext( x_descr );

 Move_3d_to( Screen[Type].XRight + 0.10 * XSize_3d,
             Screen[Type].YDown + 0.5 * YSize_3d,
             Screen[Type].ZDown );
 outtext( y_descr );

 Move_3d_to( Screen[Type].XLeft + 0.03 * XSize_3d,
             Screen[Type].YDown,
             Screen[Type].ZUp + 0.05 * ZSize_3d );
 outtext( z_descr );

 Set_3d_labels( XLeft_3d, Screen[Type].XMark, Screen[Type].XLabel,
                YDown_3d, Screen[Type].YMark, Screen[Type].YLabel,
                ZDown_3d, Screen[Type].ZMark, Screen[Type].ZLabel );
}


void Define_pH_pE_axis()
{
 Screen[3].ZDown = Screen[3].YDown; /* Z-limits from data file         */
                                    /* (Y-value position)              */
 Screen[3].XLeft = Screen[1].YDown; /* pH-axis from pH vs Step diagram */
 Screen[3].YDown = Screen[2].YDown; /* pE-axis from pE vs Step diagram */

 Screen[3].ZUp    = Screen[3].YUp;
 Screen[3].XRight = Screen[1].YUp;
 Screen[3].YUp    = Screen[2].YUp;
 Screen[3].ZMark = Screen[3].YMark;
 Screen[3].XMark = Screen[1].YMark;
 Screen[3].YMark = Screen[2].YMark;

 Screen[3].ZLabel = Screen[3].YLabel;
 Screen[3].XLabel = Screen[1].YLabel;
 Screen[3].YLabel = Screen[2].YLabel;
}



void Read_diagram_descr() /* Read the file DIAGRAM.DAT to find */
{                         /* intervals, marks, labels etc.     */
 short i,
       j = 0,
       element,
       maxlength=128;

 if( (diagram = fopen("DIAGRAM.DAT", "r")) == NULL)
 {
  printf("Cannot open DIAGRAM.DAT");
  exit(0);
 }
 fgets(Diagram_descr[j++],maxlength,diagram); /* Type descriptions */
 fgets(Diagram_descr[j++],maxlength,diagram); /* 1=pH 2=pE 3=pH-pE */
 /* Ignore! */                                /* 4=Precip-dissol   */
                                              /* 5=Concentrations  */
                                              /* 6=No picture      */

 fgets(Diagram_descr[j++],maxlength,diagram); /* X-unit descriptions */
 fgets(Diagram_descr[j++],maxlength,diagram);
 X_unitno = atoi(Diagram_descr[j-1]);         /* 1=step 2=meter 3=year */

 fgets(Diagram_descr[j++],maxlength,diagram); /* X-mark descr */
 fgets(Diagram_descr[j++],maxlength,diagram);
 Screen[Type].XMark = atof(Diagram_descr[j-1]);

 fgets(Diagram_descr[j++],maxlength,diagram); /* X-lbl descr */
 fgets(Diagram_descr[j++],maxlength,diagram);
 Screen[Type].XLabel = atof(Diagram_descr[j-1]);

 for( i = 1; i < 6; i++ )
 {
  fgets(Diagram_descr[j++],maxlength,diagram);  /* Empty line */
  fgets(Diagram_descr[j++],maxlength,diagram);  /* Diagram headline */
  strncpy(Screen[i].Diagram_descr,Diagram_descr[j-1],29);
  Trim(Screen[i].Diagram_descr);

  if( i == 5 )    /* Concentration diagram */
  {
   fgets(Diagram_descr[j++],maxlength,diagram); /* Headline no of curves */
   fgets(Diagram_descr[j++],maxlength,diagram);
   Screen[i].Curves = atoi(Diagram_descr[j-1]); /* Number of curves */

   for( element = 0; element < Screen[i].Curves; element++ )
   {
    fgets(Diagram_descr[j++],maxlength,diagram); /* Element name */
    strncpy(Screen[i].Element[element],Diagram_descr[j-1],29);
    Trim( Screen[i].Element[element] );
   }
   fgets(Diagram_descr[j++],maxlength,diagram);  /* Empty line */
  }
  fgets(Diagram_descr[j++],maxlength,diagram); /* Headline bottom y */
  fgets(Diagram_descr[j++],maxlength,diagram);
  Screen[i].YDown = atof(Diagram_descr[j-1]);

  fgets(Diagram_descr[j++],maxlength,diagram); /* Headline Top y */
  fgets(Diagram_descr[j++],maxlength,diagram);
  Screen[i].YUp = atof(Diagram_descr[j-1]);
  fgets(Diagram_descr[j++],maxlength,diagram); /*Headline Mark distance */
  fgets(Diagram_descr[j++],maxlength,diagram);
  Screen[i].YMark = atof(Diagram_descr[j-1]);

  fgets(Diagram_descr[j++],maxlength,diagram); /* Headline lbl dist */
  fgets(Diagram_descr[j++],maxlength,diagram);
  Screen[i].YLabel = atof(Diagram_descr[j-1]);
 }
 fclose(diagram);
}



void Read_result()
{
 short i,
       j,
       profile; /* Y value for display of properties */

 float old_conc[MAXELEMENT][MAXWIDTH];

 char string[120];

 if( (in = fopen("RESULT.DAT", "r")) == NULL) exit(0);

 fgets(string,118,in);           /* Header line */
 profile = (1 + Width) / 2;      /* Centre line */

 /* Move to the first point */
 Next_point();
 switch( Type )
 {
  case 1:  /* pH curve */
   Move_to( 0.0, pH[profile] );
   break;

  case 2:  /* pE curve */
   Move_to( 0.0, pE[ profile ] );
   break;

  case 3:
   Sample_pH_pE(); /* 3 dim pH - pE diagram */
   break;

  case 4:  /* Precipitation and dissolution */
   /* Not in service */
   break;

  case 5:  /* Element concentrations */
   for( j=0; j <= Screen[5].Curves; j++ )
   {
              /* Move to bottom of the diagram if C < Cmin */
    Conc[j][profile] = (Conc[j][profile] == 0)? Screen[5].YDown
                                              : Conc[j][profile];
    old_conc[j][profile] = Conc[j][profile];
   }
   break;
 }

 for( i = 1; i <= XStepNo-2; i++ )
 {
  Next_point();
  switch( Type )
  {
   case 1:  /* pH curve */
    Line_to( (float)(i), pH[profile] );
    break;
    case 2:  /* pE curve */
     Line_to( (float)(i), pE[profile] );
     break;

    case 3:  /* 3 dim pH - pE diagram */
     Sample_pH_pE();
     break;

    case 4:  /* Precipitation and dissolution */
     /* Not in service */
     break;

    case 5:  /* Element concentrations */
     for( j=0; j < Screen[5].Curves; j++ )
     {
      Move_to( (float)i, old_conc[j][profile] );
      Line_to( (float)(i+1), Conc[j][profile] );
                     /* Move to bottom of the diagram if C < Cmin */
      old_conc[j][profile]
           = ( Conc[j][profile] < Screen[5].YDown )? Screen[5].YDown
                                                   : Conc[j][profile];
     }
     break;
  }
 }
 fclose(in);
}


void Next_point()
{
 static char string[120];

 static short xstepno,
              ystepno,
              component,
              element,
              maxcomp,
              maxlength;

 maxlength=118;
 fgets(string,maxlength,in); /* X Step No */
 X = atof(string+14);
 xstepno = (short)X;
/*
 * fgets(string,maxlength,in); Number of minerals for future use
 * for ( i=0; i < LastSpot.NumberOfMinerals; i++ )
 * {
 *  fprintf(out,"%s\n", LastSpot.Mineral[i]);
 * }
 */
 for( ystepno = 1; ystepno <= Width; ystepno++ )
 {
  fgets(string,maxlength,in); /* Y Step No */
  fgets(string,maxlength,in); /* Mineral */
  fgets(string,maxlength,in); /* pH */
  pH[ystepno] = atof(string+3);

  fgets(string,maxlength,in); /* pE */
  pE[ystepno] = atof(string+3);

  /* "Number of elements: %u", Current_Solution.Components */
  fgets(string,maxlength,in);
  maxcomp = atoi(string+20);
  for( component=1; component <= maxcomp; component++ )
  {
   fgets(string,maxlength,in); /* Ex:  Na 1.51e-2 */
   /* Search for specified elements */

   for( element = 0;
        element < Screen[5].Curves;
        element++ )
   {
    if( strncmpi( Screen[5].Element[element],string,2) == 0 )
    {
     /* Element name found. Get conc */
     Conc[ element ][ ystepno ] = log10( atof( string+5 ) );
    }
   }
  }
 }
}


void Sample_pH_pE()
{
 float ph_size,
       pe_size;

 short ph_class,
       pe_class,
       ystepno;

 for( ystepno = 1; ystepno <= Width; ystepno++ )
 {
  ph_class = (pH[ystepno] -  Screen[3].XLeft) / Screen[3].XMark * 4;
  pe_class = (pE[ystepno] -  Screen[3].YDown) / Screen[3].YMark * 4;

  if(     (ph_class >= 0)
      &&  (ph_class < 40)
      &&  (pe_class >= 0)
      &&  (pe_class < 40)
    )
    pH_pE_Sample[ph_class][pe_class]++;
 }
}


void Draw_pH_pE_sample()
{
 float ph_step,
       pe_step,
       ph_pos,
       pe_pos;

 short ph_class,
       pe_class;

 ph_step = Screen[3].XMark / 4;
 pe_step = Screen[3].YMark / 4;
 ph_pos = Screen[3].XLeft + Screen[3].XMark / 8;
 for( ph_class = 0; ph_class < 40; ph_class++ )
 {
  pe_pos = Screen[3].YDown + Screen[3].YMark / 8;

  for( pe_class = 0; pe_class < 40; pe_class++ )
  {
   if( pH_pE_Sample[ph_class][pe_class] > 0 )
   {
    Move_3d_to( ph_pos, pe_pos, 0.0 );
    Line_3d_to( ph_pos, pe_pos, (float)pH_pE_Sample[ph_class][pe_class] );
   }
   pe_pos += pe_step;
  }
  ph_pos += ph_step;
 }
}



void Choose_type()
{
 short line;

 Clear_screen();
 Inscreen.x = 15;                    /* X-position of input fields      */
 Inscreen.y = 8;                     /* Y-position of first input field */
 Inscreen.items = 5;                 /* Number of input fields          */

 sprintf(Inscreen.text[0],"pH vs position\n\n");
 sprintf(Inscreen.text[1],"pE vs position\n\n");
 sprintf(Inscreen.text[2],"pH-pE ( 3-dimensional )\n\n");
 /* sprintf(Inscreen.text[3],"Precipitation-dissolution vs position"); */
 sprintf(Inscreen.text[3],"Log concentrations vs position\n\n");
 sprintf(Inscreen.text[4],"Quit\n\n");

 Type = Screen_choice( "..\\screens\\display1" ) + 1;

 if( Type > 3 ) Type++;  /* Type 4 inactive */

 Read_diagram_descr();
 if( Type < 6 )
 {
  Clear_screen();
  printf("Do you want a hardcopy (y/n)? ");
  if( (char)Wait_key() == 'y' )
  {
   printf("\n\nFile name? ");
   gets(File_name);
  }
 }
}



void Set_marks(x_start, x_division, y_start, y_division)
        float x_start, x_division, y_start, y_division;
{
 float x,
       y,
       mark_length;

 mark_length = YSize / 50;
 for( x = x_start + x_division;  /* Set marks along upper */
      x < XRight;                /* and lower x axis      */
      x += x_division )
 {
  Move_to( x, YDown );
  Line_to( x, YDown + mark_length );
  Move_to( x, YUp );
  Line_to( x, YUp - mark_length );
 }

 mark_length = XSize / 100;
 for( y = y_start + y_division; /* Set marks along left */
      y < YUp;                  /* and right y axis     */
      y += y_division )
 {
  Move_to( XLeft, y );
  Line_to( XLeft + mark_length, y );
  Move_to( XRight, y );
  Line_to( XRight - mark_length, y );
 }
}



void Set_3d_marks(x_start, x_division,
                  y_start, y_division,
                  z_start, z_division)
 float x_start,
       x_division,
       y_start,
       y_division,
       z_start,
       z_division;
{
 float x,
       y,
       z,
       mark_length;

 for( x = x_start + x_division;  /* Set marks along */
      x < XRight_3d;             /* x axis          */
      x += x_division )
 {
  Move_3d_to( x, YDown_3d, ZDown_3d );
  Line_3d_to( x, YUp_3d, ZDown_3d );
 }

 for( y = y_start + y_division;  /* Set marks along */
      y < YUp_3d;                /* y axis          */
      y += y_division )
 {
  Move_3d_to( XLeft_3d,  y, ZDown_3d );
  Line_3d_to( XRight_3d, y, ZDown_3d );
 }

 mark_length = XSize_3d / 100;
 for( z = z_start + z_division;    /* Set marks along */
      z < ZUp_3d;                  /* Z axis          */
      z += z_division )
 {
  Move_3d_to( XLeft_3d, YDown_3d, z );
  Line_3d_to( XLeft_3d + mark_length, YDown_3d, z );
 }
}
```

Appendix E: DISPLAY.C

```
void Set_labels( xleft, xmark, xlabel, ydown, ymark, ylabel,figures )
           float xleft, xmark, xlabel, ydown, ymark, ylabel;
                 short figures;
{
 float x,
       absx_l,
       y,
       absy_l;

 char textbuffer[20];

 /* Declared in allan.h: XLeft, YDown, XRight, YUp, XSize, YSize */

 for( y = ydown + ymark; y < YUp; y += ymark ) /* Along left Y-axis */
 {
  absy_l = fabs( y / ylabel );
  if( ylabel * ( absy_l - floor( absy_l ) ) < ymark / 2 )
  {
   Move_to( 0.02 * XSize, y + YSize / 100 );
   gcvt( y, 10, textbuffer);
   textbuffer[ figures ] = 0;
   outtext(textbuffer);
  }
 }

 for( x = xleft + xmark; x < XRight; x += xmark ) /* Along lower X-axis */
 {
  absx_l = fabs( x / xlabel );
  if( xlabel * ( absx_l - floor( absx_l ) ) < xmark / 2 )
  {
   Move_to( x - XSize * 0.017, YDown + 0.06 * YSize );
   gcvt( x, 10, textbuffer);
   textbuffer[ figures + 1 ] = 0;
   outtext(textbuffer);
  }
 }
}

void Set_3d_labels( xleft, xmark, xlabel,
                    ydown, ymark, ylabel,
                    zdown, zmark, zlabel )

              float xleft, xmark, xlabel,
                    ydown, ymark, ylabel,
                    zdown, zmark, zlabel;
{
 float x,
       absx_l,
       y,
       absy_l,
       z,
       absz_l;

 char textbuffer[20];

 for( x = xleft + xmark; x < XRight_3d; x += xmark ) /*Along upper X-axis*/
 {
  absx_l = fabs( x / xlabel );

  if( xlabel * ( absx_l - floor( absx_l ) ) < xmark / 2 )
  {
   Move_3d_to( x - XSize_3d * 0.017, YUp_3d + 0.06 * YSize_3d, ZDown_3d );
   gcvt( x, 10, textbuffer);
   outtext(textbuffer);
  }
 }
 for( y = ydown + ymark; y < YUp_3d; y += ymark ) /* Along right Y-axis */
 {
  absy_l = fabs( y / ylabel );
  if( ylabel * ( absy_l - floor( absy_l ) ) < ymark / 2 )
  {
   Move_3d_to( XRight_3d + 0.02 * XSize_3d, y + YSize_3d / 100, ZDown_3d );
   gcvt( y, 10, textbuffer);
   outtext(textbuffer);
  }
 }

 for( z = zdown + zmark; z < ZUp_3d; z += zmark ) /* Along left Z-axis */
 {
  absz_l = fabs( z / zlabel );
  if( zlabel * ( absz_l - floor( absz_l ) ) < zmark / 2 )
  {
   Move_3d_to( XLeft_3d + XSize_3d * 0.017, YDown_3d, z + ZSize_3d * 0.01);
   gcvt( z, 10, textbuffer);
   outtext(textbuffer);
  }
 }
}


void Coordinate_system_3d( xleft, ydown, zdown, xright, yup, zup )
                     float xleft, ydown, zdown, xright, yup, zup;
{
 Coordinates_3d( xleft, ydown, zdown, xright, yup, zup );

 Move_3d_to( xleft, ydown, zdown );
 Line_3d_to( xright, ydown, zdown );   /* Lower X-axis */

 Move_3d_to( xleft, yup, zdown );
 Line_3d_to( xright, yup, zdown );     /* Upper X-axis */

 Move_3d_to( xleft, ydown, zdown );
 Line_3d_to( xleft, yup, zdown );      /* Left Y-axis  */

 Move_3d_to( xright, ydown, zdown );
 Line_3d_to( xright, yup, zdown );     /* Right Y-axis */

 Move_3d_to( xleft, ydown, zdown );
 Line_3d_to( xleft, ydown, zup );      /* Z-axis       */
}


void Coordinates_3d( xleft, ydown, zdown, xright, yup, zup )
               float xleft, ydown, zdown, xright, yup, zup;
{
 XSize_3d  = xright - xleft;
 YSize_3d  = yup - ydown;
 ZSize_3d  = zup - zdown;
 XLeft_3d  = xleft;
 XRight_3d = xright;
 YDown_3d  = ydown;
 YUp_3d    = yup;
 ZDown_3d  = zdown;
 ZUp_3d    = zup;

 Coordinates( xleft / XSize_3d       + ydown / YSize_3d / 2.0,
              zdown / ZSize_3d / 2.0 + ydown / YSize_3d / 2.0,
              xright / XSize_3d      + yup / YSize_3d / 2.0,
              zup / ZSize_3d / 2.0   + yup / YSize_3d  / 2.0
            );
}
```

```
void Move_3d_to( x, y, z )
      float x, y, z;
{
 Move_to( x / XSize_3d       + y / YSize_3d/ 2.0,
          z / ZSize_3d / 2.0 + y / YSize_3d/ 2.0 );
}


void Line_3d_to( x, y, z )
      float x, y, z;
{
 Line_to( x / XSize_3d       + y / YSize_3d/ 2.0,
          z / ZSize_3d / 2.0 + y / YSize_3d/ 2.0 );
}
```

Appendix F: SORPTION.C

```
/*
 *    2 dimensional SORPTION version 1992 - 05
 *      by   Allan T Emren
 *           Dept Nuclear Chemistry
 *           Chalmers University of Technology
 *           S-41296 Goteborg
 *           SWEDEN
 *
 *   Compile with compact memory model (small code, large data),
 *   which is implemented in the file MKC.BAT
 */

#include "allan.h"
#include <alloc.h>

#define MAX_NUMBER_OF_MINERALS 50
#define MAXLENGTH 110
#define MAXWIDTH 21
#define MAXCLASSES 20

FILE *in,
     *pos,
     *diagram,
     *sorpfile;

float Startconc,
      Conc,
      Area,       /* of cell */
      V_solution;

float XSize_3d,  /* Will be moved to 3D.H */
      YSize_3d,
      ZSize_3d,
      XLeft_3d,
      XRight_3d,
      YDown_3d,
      YUp_3d,
      ZDown_3d,
      ZUp_3d;

char Key,
     File_name[15],
     Element_name[6],         /* Sorbed element */
     NextMineral[30],
     Diagram_head[6][28] = {
                            " ",
                            "Log C in a water package",
                            "Log C at times",
                            "3-dim",
                            "Breakthrough-curve",
                            " "
                           },
     X_unit[4][15] = {
                       " ",
                       "steps",
                       "meters",
                       "years",
                     };
```

Appendix F: SORPTION.C

```
short Step,
      Number_of_minerals,
      pH_pE_Sample[40][40],
      File_size, /* Number of lines in file describing displays */

      X_unitno, /*  Unit no:
                 *  steps       = 1
                 *  meters      = 2
                 *  years       = 3  */

      Type;    /* Screen type:
                * Snapshot      = 1
                * Breakthrough  = 2
                * 3-dim         = 3
                *               = 4
                *               = 5
                *               = 6  */

struct Display {
                short Curves,             /* Max 20        */
                      Maxtime;            /* In time steps */
                char  Diagram_descr[30],
                      Y_unit_descr[30],
                      Element[5];
                float XLeft,
                      XRight,
                      XMark,  /* Distance between marks  */
                      XLabel, /* Distance between labels */
                      YDown,
                      YUp,
                      YMark,
                      YLabel,
                      ZDown,
                      ZUp,
                      ZMark,
                      ZLabel;
               } Screen[6];

struct Sorp { char  Name[30]; /* Name of mineral */
              short Mineral_no;
              float Thickness, /* Penetration depth */
                    KV,       /* Volume based sorption koeff */
                    pE_coeff,
                    pH_coeff;
            }
            Sorpdata[MAX_NUMBER_OF_MINERALS];

struct Fracture {
                 short Mineral_no;

                 char Mineral[10];

                 float Sorbed_qty,
                       Conc,
                       pH,
                       pE;
                };
```

Appendix F: SORPTION.C

```
struct PositionData {
                     short XMaxStep,
                           XStepNo,
                           YMaxStep,
                           YStepNo;
                     long  LastRandom;
                     char  MineralName[30];
                     float Temperature,
                           TemperatureStep,
                           WaterSpeed,
                           StepLength,
                           Thickness;
                    }
                    Spot;

void Read_position(),
     Define_X_Y_axis(),
     Setup_2d_display(),
     Setup_3d_display(),
     Draw_3d(),
     Sample_pH_pE(),
     Choose_actions(),
     Read_diagram_descr(),
     Read_initial_cond(),
     Read_fracture_data(),
     Read_sorption_data(),
     Move_water(),
     Sorb(),
     Make_diagram(),
     Draw_curve(),
     Set_marks(),
     Set_3d_marks(),
     Set_labels(),
     Set_3d_labels(),
     Coordinate_system_3d(),
     Coordinates_3d(),
     Move_3d_to(),
     Line_3d_to();

void main()
{
 struct Fracture far *Fract;

 if(
    (Fract = ( struct Fracture *)farcalloc(MAXLENGTH * MAXWIDTH,
                                           sizeof(struct Fracture) )
    ) == NULL )
  {
   printf("Heap full. Decrease MAXLENGTH or MAXWIDTH in the source code\n");
   printf("or re-compile with a greater memory model.\n");
   Wait_key();
   exit(0);
  }
 Choose_actions();

 Read_initial_cond();
 Read_position();
 Read_fracture_data( Fract );

 Make_diagram();

 Move_water( Fract );

 Wait_key();
 if( File_name[0] != (char)0 ) Dump_screen( File_name );

 farfree( Fract );        /* Release memory to heap */
}
```

Appendix F: SORPTION.C

```
void Read_position()
{
 char string[130];

 short i,
       maxlength=128;

 if( (pos = fopen("POSITION.DAT", "r")) == NULL)
 {
  printf("Cannot open POSITION.DAT");
  Wait_key();
  exit(0);
 }
 fgets(string,maxlength,pos);
 fgets(string,maxlength,pos);
 /* Spot.XMaxStep = atoi(string);*/

 fgets(string,maxlength,pos);
 fgets(string,maxlength,pos);
 Spot.YMaxStep = atoi(string);
 Spot.YStepNo = atoi(string);

 if( Spot.YMaxStep > MAXWIDTH - 1 )
 {
  printf("WARNING!! Fracture width truncated to %d steps\n",MAXWIDTH-1);
  printf("Press ENTER to continue\n");
  Wait_key();
 }

 fgets(string,maxlength,pos);
 fgets(string,maxlength,pos);
 Spot.XStepNo = atoi(string) - 1; /* Skip present slice */

 if( Spot.XStepNo > MAXLENGTH - 1 )
 {
  Spot.XStepNo = MAXLENGTH - 1;
  printf("********************************************\n");
  printf("* WARNING!! Fracture truncated to %d steps *\n",MAXLENGTH-1);
  printf("********************************************\n");
  printf("Press ENTER to continue\n");
  Wait_key();
 }

 fgets(string,maxlength,pos);
 fgets(string,maxlength,pos);
 /*Spot.YStepNo = atoi(string);*/

 fgets(string,maxlength,pos);
 fgets(string,maxlength,pos);
/*
 *Spot.LastRandom = atol(string);
 */
 fgets(string,maxlength,pos);      /* Next mineral */
 fgets(Spot.MineralName,maxlength,pos);

 fgets(string,maxlength,pos); /* Temperature */
 fgets(string,maxlength,pos);
 Spot.Temperature = atof(string);

 fgets(string,maxlength,pos); /* Temperature step */
 fgets(string,maxlength,pos);
 Spot.TemperatureStep = atof(string);

 fgets(string,maxlength,pos); /* Water speed (m/year) */
 fgets(string,maxlength,pos);
 Spot.WaterSpeed = atof(string);

 fgets(string,maxlength,pos); /* Step length (m) */
 fgets(string,maxlength,pos);
 Spot.StepLength = atof(string);
 fgets(string,maxlength,pos);
 fgets(string,maxlength,pos);
 Spot.Thickness = atof(string);
 fclose(pos);
}


void Make_diagram()
{
 if( Type == 6 )
 {
 }

 if( Type != 3 )
 {
  Setup_2d_display();
 }
 else
 {
  Define_X_Y_axis();
  Setup_3d_display( "Pos", "Time", "Log conc" );
  /*  Draw_3d(); */
 }
}


void Setup_2d_display()
{
 XLeft = 0;

 switch( X_unitno )                              /* Unit along X-axis: */
 {
  case 1:                                              /* Step */
    XRight = Spot.XStepNo + 5;
   break;

  case 2:                                              /* Meter */
    XRight = ( Spot.XStepNo + 5 ) * Spot.StepLength;
   break;

  case 3:                                              /* Year */
    XRight = ( Spot.XStepNo + 5 ) * Spot.StepLength
                                  / Spot.WaterSpeed;
   break;
 }

 Open_screen();

 Coordinate_system( XLeft, Screen[Type].YDown, XRight, Screen[Type].YUp );

 Set_marks( XLeft,   Screen[Type].XMark,  YDown,   Screen[Type].YMark );
        /* X-start      X-dist            Y-start       Y-dist */

 Set_labels( XLeft, Screen[Type].XMark, Screen[Type].XLabel,
             YDown, Screen[Type].YMark, Screen[Type].YLabel,
             4 /* string length */);

 Locate(2,2);
 printf("%s vs pos (%s)\n", Diagram_head[Type],X_unit[X_unitno]);

 /* The normal X-unit is step. Temporarily, it has been changed to meter
  * or year, for drawing the axis correctly. Now, make a logical change
  * in thhe length of the X-axis, to return to the step as unit.
  */
 switch( X_unitno )                    /* Unit along X-axis: */
 {
  case 1:                                             /* Step */
   break;

  case 2:                                             /* Meter */
    XRight /= Spot.StepLength;
   break;

  case 3:                                             /* Year */
    XRight /= Spot.StepLength / Spot.WaterSpeed;
   break;
 }
 Coordinates( XLeft, Screen[Type].YDown, XRight, Screen[Type].YUp );
}



void Setup_3d_display( x_descr, y_descr, z_descr )
                  char x_descr[MAXCLASSES],
                       y_descr[MAXCLASSES],
                       z_descr[MAXCLASSES];
{
 Open_screen();

 Coordinate_system_3d( Screen[Type].XLeft,
                       Screen[Type].YDown,
                       Screen[Type].ZDown,
                       Screen[Type].XRight,
                       Screen[Type].YUp,
                       Screen[Type].ZUp
                     );

 Set_3d_marks( Screen[Type].XLeft, Screen[Type].XMark,
               Screen[Type].YDown, Screen[Type].YMark,
               Screen[Type].ZDown, Screen[Type].ZMark
             );

 Move_3d_to( Screen[Type].XLeft + 0.45 * XSize_3d,
             Screen[Type].YUp + 0.14 * YSize_3d,
             Screen[Type].ZDown );
 outtext( x_descr );

 Move_3d_to( Screen[Type].XRight + 0.10 * XSize_3d,
             Screen[Type].YDown + 0.5 * YSize_3d,
             Screen[Type].ZDown );
 outtext( y_descr );

 Move_3d_to( Screen[Type].XLeft + 0.03 * XSize_3d,
             Screen[Type].YDown,
             Screen[Type].ZUp + 0.05 * ZSize_3d );
 outtext( z_descr );

 Set_3d_labels( XLeft_3d, Screen[Type].XMark, Screen[Type].XLabel,
                YDown_3d, Screen[Type].YMark, Screen[Type].YLabel,
                ZDown_3d, Screen[Type].ZMark, Screen[Type].ZLabel );
}



void Define_X_Y_axis()   /* Not valid yet */
{
 Screen[3].ZDown = Screen[3].YDown; /* Z-limits from data file         */
                                    /* (Y-value position)              */
 Screen[3].XLeft = Screen[1].YDown; /* pH-axis from pH vs Step diagram */
 Screen[3].YDown = Screen[2].YDown; /* pE-axis from pE vs Step diagram */

 Screen[3].ZUp    = Screen[3].YUp;
 Screen[3].XRight = Screen[1].YUp;
 Screen[3].YUp    = Screen[2].YUp;
 Screen[3].ZMark = Screen[3].YMark;
 Screen[3].XMark = Screen[1].YMark;
 Screen[3].YMark = Screen[2].YMark;

 Screen[3].ZLabel = Screen[3].YLabel;
 Screen[3].XLabel = Screen[1].YLabel;
 Screen[3].YLabel = Screen[2].YLabel;
}


void Read_initial_cond()
{
 char string[120];

 short maxlength=118;

 if( (in = fopen("SORPDSCR.DAT", "r")) == NULL)
 {
  printf("Could not find file SORPDSCR.DAT\n");
  Wait_key();
  exit(0);
 }
 fgets(string,maxlength,in);  /* PHREEQE name of element */
 fgets(string,maxlength,in);  /* Ex: NP */
 strncpy( Element_name, string, 3);

 fgets(string,maxlength,in); /* Initial concentration (M) */
 fgets(string,maxlength,in);  /* Ex: 1.0e-6 */
 Startconc = atof(string);
 fclose( in );
}


void Read_diagram_descr()
{                       /* Read the file DIAGRAM.SRP to find out what kind */
                        /* of diagram to use, which intervals, marks and   */
 short i,               /* labels are desired etc                          */
       element,
       maxlength=128;

 char string[130];

 if( (diagram = fopen("SORPDIA.DAT", "r")) == NULL)
 {
  printf("Cannot open SORPDIA.DAT");
  Wait_key();
  exit(0);
 }
 fgets(string,maxlength,diagram); /* Type descriptions */
 fgets(string,maxlength,diagram); /* 1=Snapshot
                                   * 2=Break through curve
                                   * 3=C-X-t */
 Type = atoi(string);             /* 4, 5 not used */
                                  /* 6 = No picture */

 fgets(string,maxlength,diagram); /* X-unit descriptions */
 fgets(string,maxlength,diagram);
 X_unitno = atoi(string);         /* 1=step 2=meter 3=year */

 fgets(string,maxlength,diagram); /* X-mark descr */
 fgets(string,maxlength,diagram);
 Screen[Type].XMark = atof(string);

 fgets(string,maxlength,diagram); /* X-lbl descr */
 fgets(string,maxlength,diagram);
 Screen[Type].XLabel = atof(string);
 for( i = 1; i < 6; i++ )
 {
  fgets(string,maxlength,diagram);  /* Empty line */
  fgets(string,maxlength,diagram);  /* Diagram headline */
  strncpy( Screen[i].Diagram_descr, string, 29);
  Trim( Screen[i].Diagram_descr );

  fgets(string,maxlength,diagram); /* Headline bottom y */
  fgets(string,maxlength,diagram);
  Screen[i].YDown = atof(string);

  fgets(string,maxlength,diagram); /* Headline Top y */
  fgets(string,maxlength,diagram);
  Screen[i].YUp = atof(string);

  fgets(string,maxlength,diagram); /*Headline Mark distance */
  fgets(string,maxlength,diagram);
  Screen[i].YMark = atof(string);

  fgets(string,maxlength,diagram); /* Headline lbl dist */
  fgets(string,maxlength,diagram);
  Screen[i].YLabel = atof(string);

  if( i == 2 )
  {
   Filetext(string,maxlength,diagram); /* Headline Curve interval */
   Screen[i].Curves =  Fileinput(diagram);
   Filetext(string,maxlength,diagram); /* Headline Max time */
   Screen[i].Maxtime =  Fileinput(diagram);
  }
 }
 fclose(diagram);
}



void Read_fracture_data( Fract )
    struct Fracture far *Fract;
{
 char string[120];

 short i,
       j,
       component,
       element,
       maxcomp,
       maxlength=118;

 if( (in = fopen("RESULT.DAT", "r")) == NULL) exit(0);

 fgets(string,118,in); /* Header line */

 for( i = 0; i < Spot.XStepNo; i++ )
 {
  fgets(string,maxlength,in); /* Step No */
  if( i != ( atoi(string+15) - 1 ) )
  {
   printf("Found: %s\n",string);
   printf("Expected: Length step no %d\n",i+1);
   printf("Error in RESULT.DAT\n");
   Wait_key();
   exit(0);
  }
 for( j = 1; j <= Spot.YMaxStep; j++ )
 {
  fgets(string,maxlength,in); /* Step No */
  if( j != atoi(string+14) )
  {
   printf("Found: %s\n",string);
   printf("Expected: Width step no %d\n",j);
   printf("Error in RESULT.DAT\n");
   Wait_key();
   exit(0);
  }
  if( j < MAXWIDTH )
  {
   fgets( (char *) ( Fract + MAXWIDTH * i + j ) ->Mineral,18,in);
   Trim( ( Fract + MAXWIDTH * i + j ) ->Mineral);
   ( Fract + MAXWIDTH * i + j ) ->Mineral_no =
      Search_mineral_no(i,j,Fract);

   fgets(string,maxlength,in); /* pH */
   ( Fract + MAXWIDTH * i + j ) ->pH = atof(string+3);

   fgets(string,maxlength,in); /* pE */
   ( Fract + MAXWIDTH * i + j ) ->pE = atof(string+3);

   fgets(string,maxlength,in);
   maxcomp = atoi(string+20);

   for( component=1; component <= maxcomp; component++ )
   {
    fgets(string,maxlength,in); /* Ex:  Na 1.51e-2 */
   }
  }
  else
  {
   fgets(string,18,in); /* Mineral */
   fgets(string,maxlength,in); /* pH */
   fgets(string,maxlength,in); /* pE */
   fgets(string,maxlength,in);
   maxcomp = atoi(string+20);

   for( component=1; component <= maxcomp; component++ )
   {
    fgets(string,maxlength,in); /* Ex:  Na 1.51e-2 */
   }
  }
 }
}
fclose(in);}


Search_mineral_no(i,j,Fract)
            short i,j;
            struct Fracture far *Fract;
{
 short n = 0;

 while( n < Number_of_minerals )
 {
  if( strncmp( (char *) ( Fract + MAXWIDTH * i + j ) -> Mineral,
                                                    Sorpdata[n].Name,
           8 ) == 0 ) break;
  n++;
 }
 if( n == Number_of_minerals )
 {
  strncpy( Sorpdata[n].Name,
           (char *) ( Fract + MAXWIDTH * i + j ) ->Mineral, 8);
  Read_sorption_data(n);
  Number_of_minerals++;
 }
 return( n );
}



void Read_sorption_data(mineral)
                  short mineral;  /* Mineral number */
{
 char line[130],      /* Next line from SRP file */
      data_file[25],
      directory[] = "SORPDATA\\",
      sorption_ext[] = ".SRP",
      *buf;

 short maxlength=128, /* Max line length           */
       elements,
       element,
       n;

 strcpy( data_file, directory );            /* Open sorption data file */
 strcat( data_file, Sorpdata[mineral].Name );
 strcat( data_file, sorption_ext );
 if( (sorpfile = fopen( data_file, "r")) == NULL)
 {
  Clear_screen();
  Close_screen();
  printf(" No sorption data known for %s\n",Sorpdata[mineral].Name);
  Wait_key();
  exit(0);
 }
 else
 {
  fgets(line,maxlength,sorpfile);                /* Comment line   */

  fgets(line,maxlength,sorpfile);                /* No of elements */
  fgets(line,maxlength,sorpfile);
  elements = atoi(line);
  for(element=0; element<elements; element++ )
  {
   fgets(line,maxlength,sorpfile);                    /* element no */
   fgets(line,7,sorpfile);                            /* Element name */
   if( strncmp( line, Element_name, 2) == 0 )
   {
    fgets(line,maxlength,sorpfile);
    fgets(line,maxlength,sorpfile);
    Sorpdata[mineral].Thickness = atof(line);
    fgets(line,maxlength,sorpfile);
    fgets(line,maxlength,sorpfile);
    Sorpdata[mineral].KV = atof(line);

    fgets(line,maxlength,sorpfile);
    fgets(line,maxlength,sorpfile);
    Sorpdata[mineral].pE_coeff = atof(line);

    fgets(line,maxlength,sorpfile);
    fgets(line,maxlength,sorpfile);
    Sorpdata[mineral].pH_coeff = atof(line);
   }
   else     /* Skip this element */
   {
    fgets(line,maxlength,sorpfile);  /* Thickness */
    fgets(line,maxlength,sorpfile);
    fgets(line,maxlength,sorpfile);  /* KV */
    fgets(line,maxlength,sorpfile);

    fgets(line,maxlength,sorpfile);  /* pE--koeff */
    fgets(line,maxlength,sorpfile);

    fgets(line,maxlength,sorpfile);  /* pH-koeff  */
    fgets(line,maxlength,sorpfile);
   }
  }
  fclose(sorpfile);
 }
}



void Move_water( Fract )
       struct Fracture far *Fract;
{
 short x,
       y,
       slice_parity,
       leftcell,
       rightcell,
       timestep;

 Area = Spot.StepLength * Spot.StepLength;
 V_solution = Spot.Thickness * Area;

 switch( Type )
 {
  case 1:                                         /* Package */
   for( x = 0, y = 1; y <= Spot.YMaxStep; y++ )
    ( Fract + MAXWIDTH * x + y ) ->Conc = Startconc;

   for( x = 1; x < Spot.XStepNo; x++ )
   {
                                            /*          / \ / \ / \      */
    slice_parity = x % 2;                   /*    /|\   | 1 | 2 | 3 |    */
                                            /*     |     \ / \ / \ / \   */
                                            /*     |      | 1 | 2 | 3 |  */
                                            /*     |     / \ / \ / \ /   */
                                            /*     |    | 1 | 2 | 3 |    */
                                            /*           \ / \ / \ /     */
    for( y = 1; y <= Spot.YMaxStep; y++ )
    {
     leftcell = y - slice_parity;
     leftcell = ( leftcell > 0 )? leftcell
                                : Spot.YMaxStep;

     rightcell = y - slice_parity + 1;
     rightcell = ( rightcell > Spot.YMaxStep )? 1
                                              : rightcell;

     Conc = ( ( Fract + MAXWIDTH * (x-1) + leftcell ) ->Conc
           + ( Fract + MAXWIDTH * (x-1) + rightcell ) ->Conc ) / 2;

     Sorb( x, y, Fract );
    }
   }
   Draw_curve( Fract );
  break;
  case 2:            /* Snapshots */
   for( timestep = 1; timestep <= Screen[2].Maxtime; timestep++ )
   {                              /* 480 steps = 3 years */
    for( x = Spot.XStepNo - 1; x > 0; x-- )
    {
                                                /*        / \ / \ / \     */
     slice_parity = x % 2;                      /*  /|\  | 1 | 2 | 3 |    */
                                                /*   |    \ / \ / \ / \   */
                                                /*   |     | 1 | 2 | 3 |  */
                                                /*   |    / \ / \ / \ /   */
                                                /*   |   | 1 | 2 | 3 |    */
                                                /*        \ / \ / \ /     */
     for( y = 1; y <= Spot.YMaxStep; y++ )
     {
      leftcell = y - slice_parity;
      leftcell = ( leftcell > 0 )? leftcell
                                 : Spot.YMaxStep;

      rightcell = y - slice_parity + 1;
      rightcell = ( rightcell > Spot.YMaxStep )? 1
                                               : rightcell;
      Conc = ( ( Fract + MAXWIDTH * (x-1) + leftcell ) ->Conc
            + ( Fract + MAXWIDTH * (x-1) + rightcell ) ->Conc ) / 2;
      Sorb( x, y, Fract );

      /*printf("t=%d x=%d y=%d %e\n",timestep, x, y, Conc);*/
     }
    }

    x = 0;
    for( y = 1; y <= Spot.YMaxStep; y++ )
    {
     /*     if( timestep <= 5 )*/
      Conc = Startconc;
/*     else
 *     Conc = 1.0e-20;
 */
     Sorb( x, y, Fract );
    }
    if( timestep % Screen[2].Curves == 0 )
     Draw_curve( Fract);
   }
  break;
 }
}


void Sorb( x, y, Fract )
     short x, y;
       struct Fracture far *Fract;
{
 short mineral;
 float kv,
       pE,
       V_sorbed_layer;

 struct Fracture far *position;

 position = Fract + MAXWIDTH * x + y;

 mineral = position->Mineral_no;

 pE = ( position->pE > -2.0 )? position->pE
                             : -2.0;
 if( (kv = Sorpdata[ mineral ].KV) > 0 )
 {
  kv /= pow10( ( pE + 2.0 ) * Sorpdata[ mineral ].pE_coeff );
  kv *= pow10( ( position->pH - 9.2 ) * Sorpdata[ mineral ].pH_coeff );

  V_sorbed_layer = Sorpdata[ mineral ].Thickness * Area * 2; /* Two walls*/

  Conc += position->Sorbed_qty / V_solution;

  /* If the concentration is great enough let sorption take place */

  if( Conc > 1.0e-20 )
  {
   position->Conc = Conc * V_solution / ( V_solution + V_sorbed_layer * kv
                                        );
   position->Sorbed_qty = V_solution * ( Conc - position->Conc );

   if( position->Conc < 1.0e-20 ) position->Conc = 1.0e-20;
  }
  else
  {
   position->Conc = 1e-20;
  }
 }
 else
 {
  position->Conc = ( Conc > 1.0e-20 )? Conc
                                     : 1.0e-20;
 }
}


void Draw_curve( Fract )
       struct Fracture far *Fract;
{
 short i,
       j;

 /* Move to the first point */
 switch( Type )
 {
  case 1:  /* Water package */
   Move_to( 0.0,
            log10(( Fract + MAXWIDTH * 0 + Spot.YMaxStep / 2 ) ->Conc) );
   break;

  case 2:  /* Snapshots */
   /* printf("step %d   C=%e\n",
         0,( Fract + MAXWIDTH * 0 + Spot.YMaxStep / 2 ) ->Conc);*/
   Move_to( 0.0,
            log10(( Fract + MAXWIDTH * 0 + Spot.YMaxStep / 2 ) ->Conc) );
   break;

  case 3:
  /* 3 dim C vs X-t */
   break;

  case 4:  /* Sorbed qty */
   Move_to( 0.0,
            log10(
                  (Fract + MAXWIDTH * 0 + Spot.YMaxStep / 2) ->Sorbed_qty
                 )
          );
   break;

  case 5:  /* */
   break;
 }
 for( i = 1; i < Spot.XStepNo; i++ )
 {
  switch( Type )
  {
   case 1:  /* Water package */
    Line_to( (float)i,
               log10(( Fract + MAXWIDTH * i + Spot.YMaxStep / 2 ) ->Conc) );
    break;

   case 2:  /* Snapshots */
    /*printf("step %d   C=%e\n",
     *         i,( Fract + MAXWIDTH * 0 + Spot.YMaxStep / 2 ) ->Conc);*/
    Line_to( (float)i,
             log10(( Fract + MAXWIDTH * i + Spot.YMaxStep / 2 ) ->Conc) );
    break;

   case 3:  /* 3 dim diagram */
    break;

   case 4:  /* Precipitation and dissolution */
    /* Not in service */
    break;

   case 5:  /* */
    break;
  }
 }
}


void Draw_3d()  /* To be changed before use in this prg */
{
 float ph_step,
       pe_step,
       ph_pos,
       pe_pos;

 short ph_class,
       pe_class;

 ph_step = Screen[3].XMark / 4;
 pe_step = Screen[3].YMark / 4;
 ph_pos = Screen[3].XLeft + Screen[3].XMark / 8;

 for( ph_class = 0; ph_class < 40; ph_class++ )
 {
  pe_pos = Screen[3].YDown + Screen[3].YMark / 8;

  for( pe_class = 0; pe_class < 40; pe_class++ )
  {
   if( pH_pE_Sample[ph_class][pe_class] > 0 )
   {
    Move_3d_to( ph_pos, pe_pos, 0.0 );
    Line_3d_to( ph_pos, pe_pos, (float)pH_pE_Sample[ph_class][pe_class] );
   }
   pe_pos += pe_step;
  }
  ph_pos += ph_step;
 }
}
```

```
void Choose_actions()
{
 Read_diagram_descr();
 Clear_screen();
 printf("Do you want a hardcopy (y/n)? ");
 if( Wait_key() == 'y' )
 {
  printf("\n\nFile name? ");
  gets(File_name);
 }
}



void Set_marks(x_start, x_division, y_start, y_division)
         float x_start, x_division, y_start, y_division;
{
 float x,
       y,
       mark_length;

 mark_length = YSize / 50;

 for( x = x_start + x_division; /* Set marks along upper */
      x < XRight;               /* and lower x axis      */
      x += x_division )
 {
  Move_to( x, YDown );
  Line_to( x, YDown + mark_length );
  Move_to( x, YUp );
  Line_to( x, YUp - mark_length );
 }

 mark_length = XSize / 100;
 for( y = y_start + y_division; /* Set marks along left */
      y < YUp;                  /* and right y axis     */
      y += y_division )
 {
  Move_to( XLeft, y );
  Line_to( XLeft + mark_length, y );
  Move_to( XRight, y );
  Line_to( XRight - mark_length, y );
 }
}



void Set_3d_marks(x_start, x_division,
                  y_start, y_division,
                  z_start, z_division)
 float x_start,
       x_division,
       y_start,
       y_division,
       z_start,
       z_division;
{
 float x,
       y,
       z,
       mark_length;

 for( x = x_start + x_division;  /* Set marks along */
      x < XRight_3d;             /* x axis          */
      x += x_division )
 {
  Move_3d_to( x, YDown_3d, ZDown_3d );
  Line_3d_to( x, YUp_3d, ZDown_3d );
 }
 for( y = y_start + y_division;  /* Set marks along */
      y < YUp_3d;                /* y axis          */
      y += y_division )
 {
  Move_3d_to( XLeft_3d,  y, ZDown_3d );
  Line_3d_to( XRight_3d, y, ZDown_3d );
 }

 mark_length = XSize_3d / 100;
 for( z = z_start + z_division; /* Set marks along */
      z < ZUp_3d;               /* Z axis         */
      z += z_division )
 {
  Move_3d_to( XLeft_3d, YDown_3d, z );
  Line_3d_to( XLeft_3d + mark_length, YDown_3d, z );
 }
}



void Set_labels( xleft, xmark, xlabel, ydown, ymark, ylabel,figures )
           float xleft, xmark, xlabel, ydown, ymark, ylabel;
                 short figures;
{
 float x,
       absx_1,
       y,
       absy_1;

 char textbuffer[20];

 /* Declared in allan.h: XLeft, YDown, XRight, YUp, XSize, YSize */

 for( y = ydown + ymark; y < YUp; y += ymark ) /* Along left Y-axis */
 {
  absy_1 = fabs( y / ylabel );
  if( ylabel * ( absy_1 - floor( absy_1 ) ) < ymark / 2 )
  {
   Move_to( 0.02 * XSize, y + YSize / 100 );
   gcvt( y, 10, textbuffer);
   textbuffer[ figures ] = 0;
   outtext(textbuffer);
  }
 }

 for( x = xleft + xmark; x < XRight; x += xmark ) /* Along lower X-axis */
 {
  absx_1 = fabs( x / xlabel );
  if( xlabel * ( absx_1 - floor( absx_1 ) ) < xmark / 2 )
  {
   Move_to( x - XSize * 0.017, YDown + 0.06 * YSize );
   gcvt( x, 10, textbuffer);
   textbuffer[ figures + 1 ] = 0;
   outtext(textbuffer);
  }
 }
}
```

```
void Set_3d_labels( xleft, xmark, xlabel,
                    ydown, ymark, ylabel,
                    zdown, zmark, zlabel )

              float xleft, xmark, xlabel,
                    ydown, ymark, ylabel,
                    zdown, zmark, zlabel;
{
 float x,
       absx_l,
       y,
       absy_l,
       z,
       absz_l;

 char textbuffer[20];

 for( x = xleft + xmark; x < XRight_3d; x += xmark ) /*Along upper X-axis*/
 {
  absx_l = fabs( x / xlabel );
  if( xlabel * ( absx_l - floor( absx_l ) ) < xmark / 2 )
  {
   Move_3d_to( x - XSize_3d * 0.017, YUp_3d + 0.06 * YSize_3d, ZDown_3d );
   gcvt( x, 10, textbuffer);
   outtext(textbuffer);
  }
 }

 for( y = ydown + ymark; y < YUp_3d; y += ymark ) /* Along right Y-axis */
 {
  absy_l = fabs( y / ylabel );
  if( ylabel * ( absy_l - floor( absy_l ) ) < ymark / 2 )
  {
   Move_3d_to( XRight_3d + 0.02 * XSize_3d, y + YSize_3d / 100, ZDown_3d );
   gcvt( y, 10, textbuffer);
   outtext(textbuffer);
  }
 }

 for( z = zdown + zmark; z < ZUp_3d; z += zmark ) /* Along left Z-axis */
 {
  absz_l = fabs( z / zlabel );
  if( zlabel * ( absz_l - floor( absz_l ) ) < zmark / 2 )
  {
   Move_3d_to( XLeft_3d + XSize_3d * 0.017,
               YDown_3d,
               z + ZSize_3d * 0.01 );

   gcvt( z, 10, textbuffer);
   outtext(textbuffer);
  }
 }
}


void Coordinate_system_3d( xleft, ydown, zdown, xright, yup, zup )
                     float xleft, ydown, zdown, xright, yup, zup;
{
 Coordinates_3d( xleft, ydown, zdown, xright, yup, zup );

 Move_3d_to( xleft, ydown, zdown );
 Line_3d_to( xright, ydown, zdown );   /* Lower X-axis */

 Move_3d_to( xleft, yup, zdown );
 Line_3d_to( xright, yup, zdown );   /* Upper X-axis */

 Move_3d_to( xleft, ydown, zdown );
 Line_3d_to( xleft, yup, zdown );      /* Left Y-axis */
 Move_3d_to( xright, ydown, zdown );
 Line_3d_to( xright, yup, zdown );      /* Right Y-axis */

 Move_3d_to( xleft, ydown, zdown );
 Line_3d_to( xleft, ydown, zup );      /* Z-axis */
}


void Coordinates_3d( xleft, ydown, zdown, xright, yup, zup )
               float xleft, ydown, zdown, xright, yup, zup;
{
 XSize_3d  = xright - xleft;
 YSize_3d  = yup - ydown;
 ZSize_3d  = zup - zdown;
 XLeft_3d  = xleft;
 XRight_3d = xright;
 YDown_3d  = ydown;
 YUp_3d    = yup;
 ZDown_3d  = zdown;
 ZUp_3d    = zup;

 Coordinates( xleft / XSize_3d       + ydown / YSize_3d / 2.0,
              zdown / ZSize_3d / 2.0 + ydown / YSize_3d / 2.0,
              xright / XSize_3d      + yup / YSize_3d / 2.0,
              zup / ZSize_3d / 2.0   + yup / YSize_3d  / 2.0
            );
}


void Move_3d_to( x, y, z )
      float x, y, z;
{
 Move_to( x / XSize_3d       + y / YSize_3d/ 2.0,
          z / ZSize_3d / 2.0 + y / YSize_3d/ 2.0 );
}


void Line_3d_to( x, y, z )
      float x, y, z;
{
 Line_to( x / XSize_3d       + y / YSize_3d/ 2.0,
          z / ZSize_3d / 2.0 + y / YSize_3d/ 2.0 );
}
```

Appendix G: ALLAN:H

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <graphics.h>
#include <bios.h>

#define FAR far     /* Turbo C specific memory handler for 80XXX */

#define DEFAULT 2
#define TRUE 1
#define FALSE 0
#define UNDEFINED -1

#undef RAND_MAX
#define RAND_MAX 32767
#define stccpy strncpy
#define SCREENWIDTH 620
#define SCREENHEIGHT 350
#define BACKGROUND 0

float XLeft,
      YDown,
      XRight,
      YUp,
      XSize,
      YSize;

long Last_random;

short PixColor,
      Background = BACKGROUND,
      Boarder = UNDEFINED,
      BoarderColor,
      GraphicMode;

void *BitMap;
unsigned char Fill_pattern[8];

void Put_image(),
     Get_image(),
     Background_color(),
     Boarder_color(),
     Color(),
     Clear_screen(),
     Locate(),
     Close_screen(),
     Paint(),
     Setpixel(),
     Line_to(),
     Move_to(),
     Coordinate_system(),
     Coordinates(),
     Open_screen(),
     Printret(),
     Print(),
     Start_random_gen(),
     Trim(),
     Trim_space();
 char *Filetext();

 float Fileinput(FILE *source)
 {
  char string[30];
  fgets(string,30,source);
  return(atof(string));
 }
```

```
float input( question )
        char question[];
{
 float value;
 char string[30];

 printf("%s ",question);
 gets(string);
 value = atof(string);
 return( value );
}

void Open_screen()
{
 int g_driver = DETECT,
     g_mode;

 initgraph(&g_driver,&g_mode,"");
 if(g_driver<0) printf("\nERROR: driver=%d\n",g_driver);
 Color(WHITE);
 GraphicMode=TRUE;
}

void Coordinates(x_down,y_down,x_up,y_up)
float x_down,
      y_down,
      x_up,
      y_up;
{
 XLeft = x_down;
 YDown = y_down;
 XRight   = x_up;
 YUp   = y_up;
 XSize = XRight - XLeft;
 YSize = YUp - YDown;
}

void Coordinate_system(x_down,y_down,x_up,y_up)
float x_down,
      y_down,
      x_up,
      y_up;
{
 Coordinates(x_down,y_down,x_up,y_up);
 Move_to(x_down, y_down);
 Line_to(x_up, y_down);
 Line_to(x_up, y_up);
 Line_to(x_down, y_up);
 Line_to(x_down, y_down);
}

void Move_to(x,y)
float x,y;
{
 int x_screen,
     y_screen;

 y = ( y > YDown )? y
                  : YDown;
 x_screen = 5    + (int)((x - XLeft)/XSize*(SCREENWIDTH-5));
 y_screen = SCREENHEIGHT - 5 - (int)((y - YDown)/YSize*(SCREENHEIGHT-10));
 moveto(x_screen,y_screen);
}
```

Appendix G: ALLAN:H

```
void Line_to(x,y)
float x,y;
{
 int x_screen,
     y_screen;

 y = ( y > YDown )? y
                  : YDown;

 x_screen = 5   + (int)((x - XLeft)/XSize*(SCREENWIDTH-5));
 y_screen = SCREENHEIGHT - 5 - (int)((y - YDown)/YSize*(SCREENHEIGHT-10));
 lineto(x_screen,y_screen);
}



void Setpixel(x,y)
float x,y;
{
 int x_screen,
     y_screen;

 x_screen = 5   + (int)((x - XLeft)/XSize*(SCREENWIDTH-5));
 y_screen = SCREENHEIGHT - 5 - (int)((y - YDown)/YSize*(SCREENHEIGHT-10));
 putpixel(x_screen,y_screen,PixColor);
}



Read_pixel(x,y)
float x,y;
{
 int x_screen,
     y_screen;

 x_screen = 5   + (int)((x - XLeft)/XSize*(SCREENWIDTH-5));
 y_screen = SCREENHEIGHT - 5 - (int)((y - YDown)/YSize*(SCREENHEIGHT-10));
 return(getpixel(x_screen,y_screen));
}



void Paint(x,y)
     float x,y;
{
 int x_screen,
     y_screen;

 x_screen = 5   + (int)((x - XLeft)/XSize*(SCREENWIDTH-5));
 y_screen = SCREENHEIGHT - 5 - (int)((y - YDown)/YSize*(SCREENHEIGHT-10));
 floodfill(x_screen,y_screen,BoarderColor);
}



void Close_screen()
{
 if(BitMap != NULL) free(BitMap);
 closegraph();
 GraphicMode=FALSE;
}

float Random_float();
```

Appendix G: ALLAN:H

```
float Random_float()
{
 float number;

 Rnd();
 number = (float)Last_random / (float)RAND_MAX;
 return(number);
}



Rnd()
{
 Last_random = ( (long)Last_random * 31991 + 11 ) % 32768;
 return( Last_random );
}



void Start_random_gen( long seed )
{
 Last_random = seed;
}



void Locate(short line, short column)
{
 if(GraphicMode) moveto((8*column),(10*line));
 else gotoxy(column,line);
}



void Print(char *string)
{
 Trim(string);
 if(GraphicMode) outtext(string);
 else printf("%s",string);
}



void Printret(char *string) /* prints text + new line */
{
 short y;
 Trim(string);
 if(GraphicMode)
 {
  y=gety()/10;
  outtext(string);
 }
 else
 {
  y=wherey();
  printf("%s",string);
 }
 Locate(y+1,1);
}



void Clear_screen()
{
 if(GraphicMode) clearviewport();
 else clrscr();
}
```

Appendix G: ALLAN:H

```
void Color(int nr)
{
 static short i;
 setcolor(nr);
 PixColor=nr;
 setfillpattern(Fill_pattern,nr);
 if(Boarder == UNDEFINED)
 {
  for(i=0;i<8;Fill_pattern[i++]=255);
  Boarder = DEFAULT;
 }
 if(Boarder == DEFAULT) BoarderColor = nr;
}



void Boarder_color(short color)
{
 BoarderColor = color;
 Boarder = TRUE;
}



void Background_color( int color)
{
 Background=color;
 setbkcolor(color);
}



void Get_image(left,down,right,up)
         float left,down,right,up;
{
 long size;
 left = 5   + (int)((left - XLeft)/XSize*(SCREENWIDTH-5));
 down = SCREENHEIGHT - 5 - (int)((down - YDown)/YSize*(SCREENHEIGHT-10));
 right = 5   + (int)((right - XLeft)/XSize*(SCREENWIDTH-5));
 up = SCREENHEIGHT - 5 - (int)((up - YDown)/YSize*(SCREENHEIGHT-10));
 size = imagesize((int)left,(int)up,(int)right,(int)down);
 if(BitMap != NULL) free(BitMap);
 BitMap = malloc(size);
 getimage((int)left,(int)up,(int)right,(int)down,BitMap);
}



void Put_image(left,up,mode)
         float left,up;
         int mode;     /* COPY_PUT, XOR_PUT, OR_PUT, AND_PUT, NOT_PUT */
{
 left = 5   + (int)((left - XLeft)/XSize*(SCREENWIDTH-5));
 up = SCREENHEIGHT - 5 - (int)((up - YDown)/YSize*(SCREENHEIGHT-10));
 putimage((int)left,(int)up,BitMap,mode);
}
```

```
void Dump_screen( char *filename )
{
 FILE *out;
 short x, xend, y, yend, c, pix;
 char byte;

 xend = SCREENWIDTH/5;
 yend = SCREENHEIGHT/5;
 out=fopen(filename,"wb");
 fputc((char)xend,out);
 fputc((char)yend,out);
 for(y=0;y<=yend*5;y++)
 {
  for(x=0;x<=xend;x++)
  {
   byte=(char)2;
   for(pix=0;pix<5;pix++)
   {
    c=getpixel(5*x+pix,y);
    c &= 1;
    putpixel(5*x+pix,y,c);
    byte = (byte << 1)+(char)c;
   }
   fputc(byte,out);
  }
 }
 fclose(out);
 putchar((char)7);
}



void Read_screen( char *filename )
{
 FILE *in;
 short x, xend, y, yend, pix;
 char byte, bit;

 in=fopen(filename,"rb");
 xend=(short)fgetc(in);
 yend=(short)fgetc(in);
 for(y=0;y<=yend*5;y++)
 {
  for(x=0;x<=xend;x++)
  {
   byte=fgetc(in);
   for(pix=0;pix<5;pix++)
   {
    bit = byte & 1;
    if( bit > 0 )
     putpixel(5*x+4-pix,y,WHITE);
    byte = byte >> 1;
   }
  }
 }
 fclose(in);
 putchar((char)7);
}



Wait_key()
{
 int c;
 while( (c = Inkey()) == 0);
 return(c);
}
```

Appendix G: ALLAN:H

```
Inkey()
{
 int key=0;

 if( ( key = bioskey(1) ) != 0 ) /* Check if a key has been pressed */
  key = bioskey(0);
 return( key );
}



void Trim(buffer)
char buffer[];
{
 char ret=13,
      lf=10,
      *found;

 if( (found=strchr(buffer,ret)) != NULL ) *found=0;
 if( (found=strchr(buffer,lf)) != NULL ) *found=0;
}



void Trim_space(buffer)
char buffer[];
{
 char *found;
Obs! change according to Christian's version!

 while( (found=strrchr(buffer,' ')) != NULL ) *found=0;
}



char *Filetext(char *string,short length,FILE *source)
{
 char *result;
 result=fgets(string,length,source);
 Trim(string);
 return(result); /* NULL at EOF */
}
```

Appendix H: SCREEN.H

```
#include <dir.h>

#define RETURNKEY 7181
#define UPKEY 18432
#define DOWNKEY 20480
#define LEFTKEY 19200
#define RIGHTKEY 19712
#define F1KEY 15104
#define OLD 4
#define NEW 5

char DOS_command[80];

struct Text_screen {
                    short x,   /* Position of first suggested answer */
                          description_x, /* First description line   */
                          y,
                          items,         /* Number of items to read */
                          status;        /* OLD or NEW */
                    char text[30][80],
                         description[30][80],
                         prototype[20];
                    double real[30];
                    long number[30];
                   }
                   Inscreen;

short Edit_string();

void Highlight(),
     Lowlight(),
     Mark(),
     Unmark(),
     Screen_input(),
     Screen_list(),
     Screen_msg();

char *Screen_file(),
     *Skip_extension();


void Screen_input( char screen_name[])
{
 FILE *in;
 short key,
       i;
 char string[80];

 Open_screen();
 if( (in = fopen( screen_name, "r")) == NULL )
 {
  printf("\nCould not open file %s\n",screen_name);
  exit(0);
 }
 Locate(1,1);
 while( fgets( string, 78, in ) != NULL ) /* Draw background */
  Printret(string);
 fclose(in);

 sprintf(Inscreen.text[Inscreen.items],"OK");
 Locate(30,10);
 Print("You may update highlighted answer.");
 Locate(31,10);
 Print("Move among answers:   ARROW KEYS");
 Locate(32,10);
 Print("Delete characters:    SPACE KEY");
 Locate(33,10);
 Print("Accept entire screen: Move to OK and press ENTER KEY");
 for( i = 0; i <= Inscreen.items; i++ )
 {
  Locate(Inscreen.y + 2 * i, Inscreen.x);
  Print(Inscreen.text[i]);
  Inscreen.real[i] = atof( Inscreen.text[i] );
  Inscreen.number[i] = atol( Inscreen.text[i] );
 }
 i = Inscreen.items;

 while( 1 )
 {
  key = Edit_string( Inscreen.y + 2 * i, Inscreen.x, Inscreen.text[i] );
  Inscreen.real[i] = atof( Inscreen.text[i] );
  Inscreen.number[i] = atol( Inscreen.text[i] );
  switch(key)
  {
   case UPKEY:
    if( i > 0 ) i--;
    break;

   case DOWNKEY:
    if( i < Inscreen.items ) i++;
    break;

   case RETURNKEY:
    if( i == Inscreen.items ) goto Leave;
  }
 }
 Leave:
 Close_screen();
}



void Screen_msg( char screen_name[])
{
 FILE *in;
 short key,
       i;
 char string[80];

 Open_screen();
 Clear_screen();
 if( (in = fopen( screen_name, "r")) == NULL )
 {
  printf("\nCould not open file %s\n",screen_name);
  exit(0);
 }
 Locate(1,1);
 while( fgets( string, 78, in ) != NULL ) /* Draw background */
  Printret(string);
 fclose(in);

 Locate(33,25);
 Print("To continue press ENTER");

 if(Inscreen.items>0)
 {
  for( i = 0; i < Inscreen.items; i++ )
  {
   Locate(Inscreen.y + 2 * i, Inscreen.x);
   Print(Inscreen.text[i]);
  }
 }
 Wait_key();
 Close_screen();
}
```

Appendix H: SCREEN.H

```
void Screen_list( char screen_name[] )
{
 FILE *in;
 short key,
       i;
 char string[80];

 Open_screen();
 if( (in = fopen( screen_name, "r")) == NULL )
 {
  printf("\nCould not open file %s\n",screen_name);
  exit(0);
 }
 Locate(1,1);
 while( fgets( string, 78, in ) != NULL ) /* Draw background */
  Printret(string);
 fclose(in);

 sprintf(Inscreen.text[Inscreen.items],"OK");
 Locate(30,10);
 Print("You may update highlighted answer.");
 Locate(31,10);
 Print("Move among answers:   ARROW KEYS");
 Locate(32,10);
 Print("Delete characters:    SPACE KEY");
 Locate(33,10);
 Print("Accept entire screen: Move to OK and press ENTER KEY");
 for( i = 0; i < Inscreen.items; i++ )
 {
  Locate(Inscreen.y + 2 * i, Inscreen.description_x);
  Print(Inscreen.description[i]);
  Locate(Inscreen.y + 2 * i, Inscreen.x);
  Print(Inscreen.text[i]);                /* Default answer  */
  Inscreen.real[i] = atof( Inscreen.text[i] );
  Inscreen.number[i] = atol( Inscreen.text[i] );
 }
 i = Inscreen.items;

 while( 1 )
 {
  key = Edit_string( Inscreen.y + 2 * i, Inscreen.x, Inscreen.text[i] );
  Inscreen.real[i] = atof( Inscreen.text[i] );
  Inscreen.number[i] = atol( Inscreen.text[i] );
  switch(key)
  {
   case UPKEY:
    if( i > 0 ) i--;
    break;

   case DOWNKEY:
    if( i < Inscreen.items ) i++;
    break;

   case RETURNKEY:
    if( i == Inscreen.items ) goto Leave;
  }
 }
 Leave:
 Close_screen();
}
```

Appendix H: SCREEN.H

```
Screen_choice( char screen_name[])
{
 FILE *in;
 short key,
       i;

 char string[80];

 Open_screen();

 if( (in = fopen( screen_name, "r")) == NULL )
 {
  printf("\nCould not open file %s\n",screen_name);
  exit(0);
 }
 Locate(1,1);
 while( fgets( string, 78, in ) != NULL ) /* Draw background */
  Printret(string);
 fclose(in);

 Locate(30,10);
 Print("Move among answers:   ARROW KEYS");
 Locate(31,10);
 Print("Change an item: Move to it and press ENTER KEY");
 Locate(32,10);
 Print("Accept entire screen: Move to OK and press ENTER KEY");
 sprintf(Inscreen.text[Inscreen.items],"OK");
 Inscreen.items++;

 for( i = 0; i < Inscreen.items; i++ )
 {
  Locate(Inscreen.y + 2 * i, Inscreen.x);
  Print(Inscreen.text[i]);
 }
 i = Inscreen.items-1;
 Mark(Inscreen.x,Inscreen.y+2*i,Inscreen.text[i]);

 while((key=Wait_key()) != RETURNKEY)
 {
  Unmark(Inscreen.x,Inscreen.y+2*i,Inscreen.text[i]);
  switch(key)
  {
   case UPKEY:
    if( i > 0 ) i--;
    break;

   case DOWNKEY:
    if( i < Inscreen.items - 1 ) i++;
    break;
  }
  Mark(Inscreen.x,Inscreen.y+2*i,Inscreen.text[i]);
 }
 Accepted:
 Close_screen();
 return(i);
}
```

Appendix H: SCREEN.H

```
Screen_alternatives( char screen_name[])
{
 FILE *in;
 short key,
       i;
 char string[80];

 Open_screen();

 if( (in = fopen( screen_name, "r")) == NULL )
 {
  printf("\nCould not open file %s\n",screen_name);
  exit(0);
 }
 Locate(1,1);
 while( fgets( string, 78, in ) != NULL ) /* Draw background */
  Printret(string);
 fclose(in);

 Locate(31,10);
 Print("Move among answers:   ARROW KEYS");
 Locate(32,10);
 Print("Choose an item: Move to it and press ENTER KEY");
 for( i = 0; i < Inscreen.items; i++ )
 {
  Locate(Inscreen.y + 2 * i, Inscreen.x);
  Print(Inscreen.text[i]);
 }
 i = Inscreen.items-1;
 Mark(Inscreen.x,Inscreen.y+2*i,Inscreen.text[i]);

 while((key=Wait_key()) != RETURNKEY)
 {
  Unmark(Inscreen.x,Inscreen.y+2*i,Inscreen.text[i]);
  switch(key)
  {
   case UPKEY:
    if( i > 0 ) i--;
    break;

   case DOWNKEY:
    if( i < Inscreen.items - 1 ) i++;
    break;
  }
  Mark(Inscreen.x,Inscreen.y+2*i,Inscreen.text[i]);
 }
 Accepted:
 Close_screen();
 return(i);
}



char *Screen_file( char *path )
{
 short key,
       i,
       imax;
 char string[80],
      name[25][20];
 char result[25];
 struct ffblk current_file;

 Open_screen();
```

```
for( i = 0; i < Inscreen.items; i++ )
{
 Locate(Inscreen.y + 2 * i, Inscreen.x);
 Print(Inscreen.text[i]);
}
i=0;
Locate(1,10);

findfirst(path,&current_file,0);
Skip_extension(current_file.ff_name);
strcpy(name[i],current_file.ff_name);
Print(name[i]);
i++;
while( findnext(&current_file) == 0 )
{
 Skip_extension(current_file.ff_name);
 strcpy(name[i],current_file.ff_name);
 Locate(2*i+1,10);
 Print(name[i]);
 i++;
}
imax = i;
Locate(10,38);
Print("Move among names:           ARROW KEYS");
Locate(12,38);
Print("Choice of highlighted name: ENTER KEY");
if(Inscreen.status == OLD)
{
 Locate(14,38);
 Print("To create a new name:       Choose NEW");
 strcpy(name[i],"NEW");
 Locate(2*i+1,10);
 Print(name[i]);
}
i = 0;
Mark(10,2*i+1,name[i]);
while( (key=Wait_key()) != RETURNKEY )
{
 Unmark(10,2*i+1,name[i]);
 switch(key)
 {
  case UPKEY:
   if( i > 0 ) i--;
   break;

  case DOWNKEY:
   if( i < imax ) i++;
   break;
 }
 Mark(10,2*i+1,name[i]);
}
```

```
 if( i == imax ) /* New name */
 {
  strcpy(result,"        ");
  Clear_screen();
  Locate(29,15);
  Print("Write the new name in the text field (max 8 characters)");
  Locate(30,15);
  Print("Move within text field:   ARROW KEYS");
  Locate(31,15);
  Print("Delete characters:        SPACE KEY");
  Locate(32,15);
  Print("Accept name:              ENTER KEY");
  Edit_string(25,20,result,0);
  Trim_space(result);
  Inscreen.status = NEW;
  sprintf(Inscreen.text[0],"Choose an old file as prototype");
  sprintf(Inscreen.text[1],"for %s",result);
  Inscreen.items = 2;
  Close_screen();
  strcpy(Inscreen.prototype,Screen_file(path));
  Inscreen.status = NEW;
 }
 else
 {
  Inscreen.status = OLD;
  strcpy(result,name[i]);
  Close_screen();
 }
 return(result);
}



short Edit_string( short y, short x, char *string )
{
 short pos = 0,
       oldpos = 0,
       key,
       len; /* character # */
 char c[2];

 c[1]=0;
 len = strlen(string);
 Mark(x,y,string);
 c[0]=string[pos];
 Lowlight(x,y,c);
 while( 1 )
 {
  key = Wait_key();
  if(   key == UPKEY || key == DOWNKEY || key == RETURNKEY
    ) break;

  switch( key )
  {
   case LEFTKEY:
    if( pos > 0 ) pos--;
    break;

   case RIGHTKEY:
    pos++;
    break;

   default:
    string[pos] = (char)key;
    pos++;
  }
  if( pos >= len )
  {
   len++;
   string[len]=(char)0;
  }
  c[0]=string[oldpos];
  Highlight(x+oldpos,y,c);
  c[0]=string[pos];
  Lowlight(x+pos,y,c);
  oldpos = pos;
 }
 Unmark(x,y,string);
 return( key );
}



void Highlight(short x,short y,char *string) /* One character */
{
 short x1,y1,l,c,b;
 Locate(y,x);
 l=textwidth(string);
 c=PixColor;
 b=Background;
 x1=getx();
 y1=gety();
 setfillstyle(SOLID_FILL,c);
 bar3d(x1,y1-1,x1+l,y1+9,0,1);
 Color(b);
 Print(string);
 Color(c);
}



void Lowlight(short x,short y,char *string)
{
 short x1,y1,l,c,b;
 Locate(y,x);
 l=textwidth(string);
 c=PixColor;
 b=Background;
 Color(b);
 x1=getx();
 y1=gety();
 setfillstyle(SOLID_FILL,b);
 bar3d(x1,y1-1,x1+l,y1+9,0,1);
 Color(c);
 Print(string);
}



void Mark(short x,short y,char *string) /* Whole string */
{
 short x1,y1,l,c,b;
 Locate(y,x);
 l=textwidth(string);
 c=PixColor;
 b=Background;
 x1=getx();
 y1=gety();
 setfillstyle(SOLID_FILL,c);
 bar3d(x1-5,y1-3,x1+l+5,y1+11,0,1);
 Color(b);
 Print(string);
 Color(c);
}
```

Appendix H: SCREEN.H

```
void Unmark(short x,short y,char *string)
{
 short x1,y1,l,c,b;
 Locate(y,x);
 l=textwidth(string);
 c=PixColor;
 b=Background;
 Color(b);
 x1=getx();
 y1=gety();
 setfillstyle(SOLID_FILL,b);
 bar3d(x1-5,y1-3,x1+l+5,y1+11,0,1);
 Color(c);
 Print(string);
}


char *Skip_extension(char file[])
{
 char *pos;
 if((pos=strchr(file,(int)'.')) != NULL) *pos = (char)0;
 return(pos);
}
```

www.ski.se