APPLICATIONS OF COMPUTATIONAL AND GRAPHICAL METHODS FOR SEPARATION PROCESSES CALCULATIONS AND EDUCATION

by

Russell L. Natter

In partial fulfillment of the requirements
for the degree of

Master of Science

California Institute of Technology

1986

ACKNOWLEDGMENTS

A great deal of thanks is due to my advisor, Dr. Fred Shair. His limitless enthusiasm, constant supply of new ideas, and confidence in my abilities has helped my growth as a competent and socially aware engineer. Our relationship has grown from acquaintances, to friends, to an unbeaten doubles team alliance on the tennis court.

A debt of gratitude must also be paid to Betsy for not only helping directly on this report but for sharing her companionship and love over the past four years. As we leave Caltech, I look forward with a great deal of excitement to starting a new phase of our life together.

I wish to express a truly deep note of gratitude to my supportive and loving parents. Their trust and belief in me has been felt throughout my stay at Caltech and, indeed, throughout my life. Because they have taught me the values of caring, responsibility, and integrity, all of my successes must be shared with them.

The financial support of Dean Geoffrey Fox and the Educational Computing Department is also gratefully acknowledged.

R.L.N., Pasadena, May 1986

ABSTRACT

Two computer programs were developed to teach separation processes and to solve related problems. The first, SEPTECH, uses the Fenske-Underwood-Gilliland method to find stage requirements and column operating parameters of a multicomponent distillation tower given feed stream information and desired product purities. The second program was designed to achieve specific educational goals. A robust educational program was developed that was a general, flexible, fast, understandable, and easy to use tool for problem solving. This program, MCTHIELE, utilizing McCabe-Thiele graphical solution techniques, was used in a senior level chemical engineering course and proved to be a very successful learning tool.

TABLE OF CONTENTS

1.0 INTRODUCTION

As the use of digital computing systems increases in the chemical industry, the need to develop good computer tools for computational and educational use at the university level is created. The response of many chemical engineering departments to this need has ranged all the way from an earnest commitment to computer aided instruction to a firm belief in traditional teaching methods. The benefits of introducing state of the art computing technology into the coursework are numerous. With the increased computational speed, a large number of problems can be solved in a fraction of the time one solution would otherwise take. This ability allows students to have a greater exposure to a wide range of complex problem types. Also, the tedious preparation of engineering diagrams and repetitive calculations can easily be done on the computer. This frees the student to concentrate on the interplay of equations and problem specifications.

The explosion of computer technology has not only affected computing capability but also has generated

inexpensive, high resolution graphic display devices. The increased graphics capabilities affect chemical engineering education in two major ways. The first of these is the ability to convert the pages of numbers commonly generated by a process simulation program to a sequence of graphs showing trends and patterns in the solutions. The ability to see a problem's subtle dependencies and possible trouble spots is greatly enhanced by the presentation of data in a graphical format. Secondly, there are several graphical solution techniques used in chemical engineering that may be applied directly to a computer driven graphics screen. These techniques include the construction of boiling point, enthalpy, and phase diagrams. In process design there are several graphical methods particularly for heat exchanger network design. And in separation processes, the famous McCabe-Thiele, Ponchon-Savarit, and triangular phase diagrams can be implemented directly on a computer screen.

As little as two years ago, undergraduates from Caltech were graduating without any computer experience. This deficiency was recognized by the school and the department as a problem that needed to be remedied. Since then, a computer programming class has been added to the general requirements for graduation and the chemical engineering

department has reformatted the separation processes class to emphasize engineering computational skills. Assignments using developed software are given in the chemical process design, process control, and engineering economics courses. Computational elements are also being introduced into the transport phenomena, kinetics, and thermodynamics classes. At Caltech, the chemical engineering department has taken one of the leading positions on campus in the development and utilization of computer aided instruction.

Both of the projects detailed in this report are products of the chemical engineering department's emphasis on educational computing. The first program is a multicomponent, multistage distillation design tool that began as a class assignment. The second project involved the development of an educational tool for use in the senior level separations course.

The program SEPTECH computes the tray requirements, the condenser and reboiler temperatures, pressures and duties, the reflux ratio, and the product stream compositions of a multicomponent distillation tower. The idea for this project originated at a chemical engineering convention where several vendors were displaying engineering software. One of the packages was a design tool for multicomponent

distillation columns. This program calculated the compositions of the product streams given the feed composition and all other design parameters. To achieve a desired product purity, the user of this program would have to guess at a column size and operating condition and use the program to see if this guess would satisfy the constraints. This iterative guessing procedure was also required in other available distillation packages. A more reasonable design tool would take as its input the feed stream composition, and the desired product purities. From this information, the column size and necessary operating conditions would be generated. With these ideas in mind, the development of SEPTECH began.

SEPTECH utilizes composition dependent modified Raoult's law thermodynamics to calculate phase equilibrium information. The computational algorithm generally follows the Fenske-Underwood-Gilliland method for distillation column calculations. Unlike the rigorous computational techniques available in a main frame process simulator, this algorithm uses much less time to find an approximate solution to the design problem. The computational method and use of the program are covered in the next section.

The MCTHIELE program uses the McCabe-Thiele method for

design of distillation equipment. Because this program was designed as an educational tool, a great deal of extra thought and care was used in its development. The program design is general enough to solve over twenty five different problem types and flexible enough to allow a wide range of operating conditions. The program was designed to be self explanatory, easy to use, and simple to understand. The input routines are completely unstructured to allow the user total freedom of problem specification. Solutions to the problems are generated very rapidly to allow the exploration of various parametric dependencies.

Because this program is based on a graphical solution technique, full advantage is taken of the available graphics hardware. The calculation technique of the MCTHIELE program is completely visually based. The screen and a mouse pointing device are used as a computerized pad and pencil for constructing the separations diagrams. Full advantage is taken of this electronic draftsman's unmatched speed and accuracy.

The majority of this report will be devoted to the MCTHIELE program. The development of the McCabe-Thiele method is introduced and solution diagrams depicting various operating conditions are shown. Then, some of the early

thoughts on the learning process and on features of successful educational computing tools are discussed. A description of how to use the MCTHIELE program is given along with a summary of this program's use in a senior level chemical engineering course. In the appendix, a listing of the C language code for both programs is included.

2.0 MULTICOMPONENT DISTILLATION CALCULATIONS

2.1 FENSKE-UNDERWOOD-GILLILAND COMPUTATIONAL METHOD

Because of the computational complexity inherent in the multicomponent multistage separation problem, approximate calculation methods are widely used for preliminary design. One of the most powerful approximate methods for determining reflux and stage requirements of multicomponent distillation towers combines the independent works of Fenske, Underwood and Gilliland. A flow chart of the this method is shown in Figure (1).

To begin the problem solution procedure, the feed stream must be specified. Flow rates of each of the components in the feed stream are found from the mole or mass fraction information and the total feed stream flow rate given. Temperature and pressure data for the feed stream is used to find the thermal condition. And, for multicomponent feed mixtures, the two key components must be selected. The key components are those that the distillation process is designed to separate. The theoretical goal of the distillation column is to have all

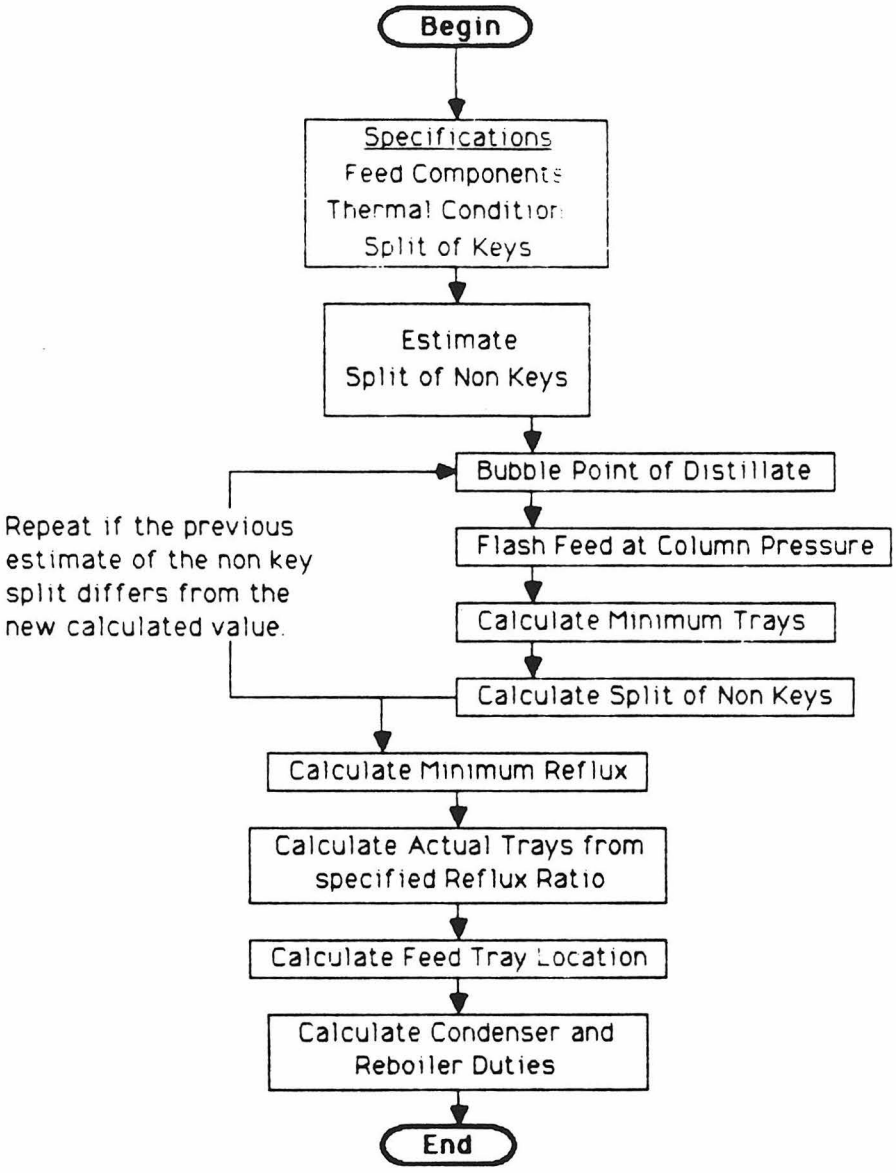

Figure (1) Fenske-Underwood-Gilliland Algorithm

of the light key and more volatile components leave through the top of the column and the remainder of the feed components leave through the bottom. Because this case of perfect separation can never occur in practice, the desired splits (or degree of separation) of the key components must also be given. This is usually specified by the percent recovery of the light key and percent contamination of the heavy key in the distillate stream.

With the feed composition, temperature, and pressure specified, the calculation enters the large, outer iteration loop. The splits of the non key components at a total reflux operating condition are calculated after every completion of the outer loop. These numbers are then compared to the previously calculated values to check for convergence. If the values differ by an amount less than the set tolerance, the outer loop is exited and the remainder of the algorithm is executed. If not, the compositions throughout the column are adjusted and the loop calculation begins again. The iteration process is begun by estimating the non key splits as the outer loop is entered for the first time.

The first calculation of the outer loop finds the bubble point of the distillate stream. Before bubble point

calculations can be made, some restrictions have to be placed on the condenser type. It is assumed in this version of the algorithm that a water-cooled, total condenser is used. The use of water as a coolant usually implies a minimum temperature of 120 degrees F in the condenser. Because a total condenser is assumed here, the distillate stream will be in the saturated liquid state. Now that the temperature, composition, and thermal condition of the distillate stream is known, the condenser pressure can be found by a bubble point calculation.

Once the condenser pressure is calculated, an estimate of the pressure at the feed tray can be found. By assuming a pressure drop of five psi through the condenser and another two and a half psi through the trays above the feed location, one finds the column pressure at the feed point. If the specified feed pressure is less than the column pressure, a compressor will be added to raise the feed stream pressure before it enters the column.

A flash calculation is done on the feed stream to find the compositions of the vapor and liquid phases after the sudden pressure drop upon entering the column. The compositions are found by iterating the Rachford-Rice flash equation. This inner iteration loop is solved with an

accelerated Newton-Raphson convergence method.

Knowing the contributions of the feed stream to the internal vapor and liquid flows of the column and recalling the distillate purity specifications, the minimum necessary number of trays can be found. This information, along with the definition of the phase equilibrium K-value and the mole fraction equality between stages, is used to derive the Fenske equation. From this equation, the tray requirement for a column operating at total reflux is found.

Once the minimum number of stages is determined, the Fenske equation can again be used to find the distribution of the non key components in the column. These values can then be compared to the last estimates of the non key splits that were used to start the outer iteration loop.

Assuming the tolerance condition has been met and the outer iteration loop is exited, the column's minimum reflux ratio is then calculated. If the assumption is made that all of the components will be present in the distillate and bottoms streams in at least trace amounts, the Underwood equation can be used. From compositions and flow rates of the feed, distillate, and bottoms streams, the Underwood equation is solved to find the minimum reflux ratio.

For a specified separation, the column will be operated

between the conditions of minimum reflux and minimum trays. The actual reflux ratio has a direct relation to the product stream flow rate, and the rate of product formation affects the economic feasibility of the project. Although the number of trays required in the column will affect the capital cost of the column, the operating cost and revenue of the column is often the the more important economic factor. Therefore, the actual reflux ratio of the column is set by economic considerations and the number of trays is calculated from the set reflux. A common economic rule of thumb for distillation column operation is to run the reflux ratio at approximately 1.3 times the minimum reflux ratio.

With the reflux ratio set, the number of theoretical trays needed for the desired separation can be found from the Gilliland correlation. For the preliminary design, this correlation is very useful. The number of trays that it predicts is remarkably close to that given by the rigorous calculation technique over a wide range of conditions. Implicit in the Gilliland correlation is the specification that the feed is introduced onto the optimal feed tray. With this information, the Kirkbride equation can then be used to find the feed tray location.

In the final step of the algorithm, a dew point

calculation is made to find the operating temperature in the reboiler. Knowing the temperature, pressure, and compositions in the condenser and reboiler, the energy requirements of the column are found.

2.2 COMPUTER IMPLEMENTATION, SEPTECH

A computer program was developed on an IBM PC/XT to utilize the computational procedure outlined above. The program, SEPTECH, was written in the C language with the HALO graphics package. This language and graphics software was recommended as a combination that will run on the greatest number of hardware arrangements. The fast growing popularity and the UNIX system compatibility of the C language were also factors in its selection. Development was done on the IBM PC/XT because of its wide spread popularity and availability.

During the development of this program, care was taken to make the input routines as easy to use and as self explanatory as possible. Titles are placed on the top of every screen directing the user how to enter the requested information at each step of the input procedure. Along the bottom of every screen is a help line that identifies and explains the functions of the active command keys on the

keyboard. All of the variable specifications are sorted and echoed back to the screen for the users inspection and verification.

As can be seen in the print out of the main program loop in Figure (2), SEPTECH closely follows the modified Fenske-Underwood-Gilliland method presented above. Because the necessary problem specifications are common to all columns designed with this method, the input routines for the program are marched through in a linear fashion. A list of feed and operating variables must be given to start the computational method and the program asks for these in a logical manner.

The feed components are chosen from an internal database of twenty compounds. All of the necessary thermodynamic information for this collection of light hydrocarbons is coded into the program. The user chooses the components by moving across the color screen with the cursor arrow key until the desired compound is highlighted. By pressing the return key, the currently highlighted compound is chosen. The user continues moving around the compounds with the cursor arrows, choosing or unchoosing with the return key, until all of the desired compounds have been selected. The escape key ends the component selection

```
#include (global.h)
#include (stdio.h)

main()
{
        init_halo(4);                   /* initialize HALO graphics      */
        _setworld();                    /* set world coordinates         */

        comp();                         /* choose feed components        */

        feedconc();                     /* specify feed conditions       */

        therm();                        /* check thermal condition       */

        split();                        /* specify desired split         */

        while (!converge)
        {
                bubble();       /* bubble point calculation of distillate */

                flashcol();     /* flash feed stream at column pressure   */

                min_stage();    /* calculate minimum theoretical stages   */

                TR_dist();      /* calc distribution at total reflux      */
        }

        putchar(7);             /* signal the end of main iterations      */

        min_rflx();             /* calc minimum reflux ratio              */

        actual();               /* calc actual # of stages & reflux ratio */

        feedstage();            /* calc optimal feed stage location       */

        boiler();               /* calc reboiler conditions               */

        if (comprs)
                compress();     /* notify if compressor is needed         */

        results2();             /* displays results on graphics screen    */
```

Figure (2) SEPTECH Main Program Loop

procedure.

The screen is then refreshed and the chosen components are rewritten onto the screen. Flow rates of each of the components are requested, as are the temperature and pressure of the feed stream. From this screen, the feed stream conditions are completely specified.

The identification and desired splits of the light and heavy key components are asked for next. As the screen is again refreshed, a list of the compounds in order of decreasing volatility is shown. From this, the two adjacent keys are chosen together. The desired percentage of the light key recovered and the allowed percentage of the heavy key contaminant in the distillate stream is then specified.

Looking back at the main program in Figure (2), one can see that the program now follows the computational procedure outlined above. Because the computational time can range from two seconds to two minutes, a small column diagram is drawn on the screen after every iteration around the outer loop to signify that calculations are proceeding. A beep from the computer signals that the outer loop has converged and the column results are on their way.

If the final column pressure was greater than the feed stream pressure and a compressor was needed, a screen

alerting the user of the new feed conditions before column entry is shown. The results of the program fill two screens. On the first screen the product stream compositions and the feed condition are shown numerically and graphically. The second screen displays the column operating conditions, tray requirement, and optimal feed location. These two screens of results can be flipped back and forth by hitting the space bar.

The development of this program is still in progress. Currently, some numerical stability problems exist for close boiling mixtures of over five components. In these cases, oscillations about the desired solutions sometimes occur. Problems can also arise in the calculation of mixtures with one or two wide boiling components. Often numerical deviations will begin and then settle to an unphysical solution. It is suspected that multiple numerical steady states are present in these instances.

Although the program still has these rough spots, the solution to most problems can be found. A listing of the latest version (Developmental Version 3.40) is included as an appendix.

3.0 GRAPHICAL SOLUTION TECHNIQUES, MCCABE-THIELE METHODS

3.1 PHASE EQUILIBRIA

One of the fundamental assumptions in the McCabe-Thiele graphical technique is a constant pressure throughout the column. Although a pressure drop through the column could be included in the technique, it would introduce another level of iterations into the calculations. This would complicate the solution procedure and, in most cases, would change the diagram only slightly. Therefore, a representation of the isobaric phase behavior of the binary feed mixture is needed.

The boiling point diagram shown in Figure (3) shows the constant pressure phase equilibria behavior of the two components, A and B. Temperature is plotted on the ordinate and concentration, in mole percent, is plotted on the abscissa. Component A which is more volatile than B, has the lower boiling temperature of t_A. Component B boils at a temperature of t_B. On the diagram, two curves begin and end at the two boiling temperatures. The upper line represents the loci of temperatures at which the first drop of liquid

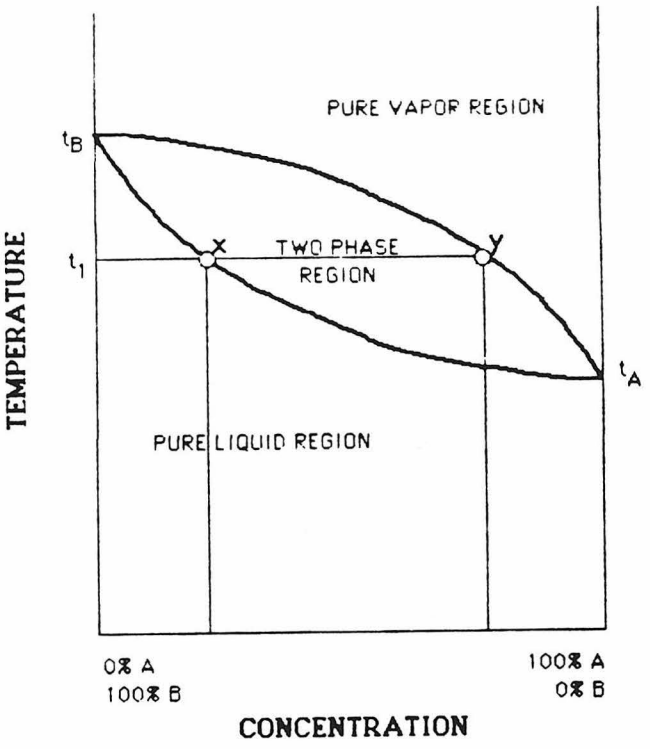

Figure (3) Boiling Point Diagram

would condense from a vapor mixture. For instance, the point y on the figure represents the composition of the vapor mixture that would just begin to condense at the temperature t_1. The concentration of that first drop of liquid is shown by the point x. All of the points on the lower curve represent the compositions of the liquid mixtures that would just begin to boil at the corresponding temperatures. The composition of the first bubble of vapor

that would form from a liquid mixture of composition x, would be represented by the point y. As this mixture continued to heat, the temperature would rise and the compositions would both move toward the point t_B. The upper line is referred to as the dew point line and the lower, as the bubble point line. The two points, x and y, on the same horizontal line represent the concentrations of the liquid and vapor phases at equilibrium at the specified conditions.

By collecting a set of the x and y points of vapor-liquid phase equilibria from the boiling point diagram, an x-y composition diagram can be found. The x-y composition diagram, or equilibrium line, is the plot of these x and y equilibrium pairs on a square diagram. This diagram is the starting point of the McCabe-Thiele diagram construction. Examples of the transfer from a boiling point diagram to a x-y composition diagram for both a Raoult's law mixture and an azeotropic mixture are shown in Figure (4).

There are several forms of phase equilibria equations that can be used to analytically solve for the equilibrium line. The simplest form of equilibrium information is the case of constant relative volatility. This condition is rather rare in real world thermodynamics because it assumes

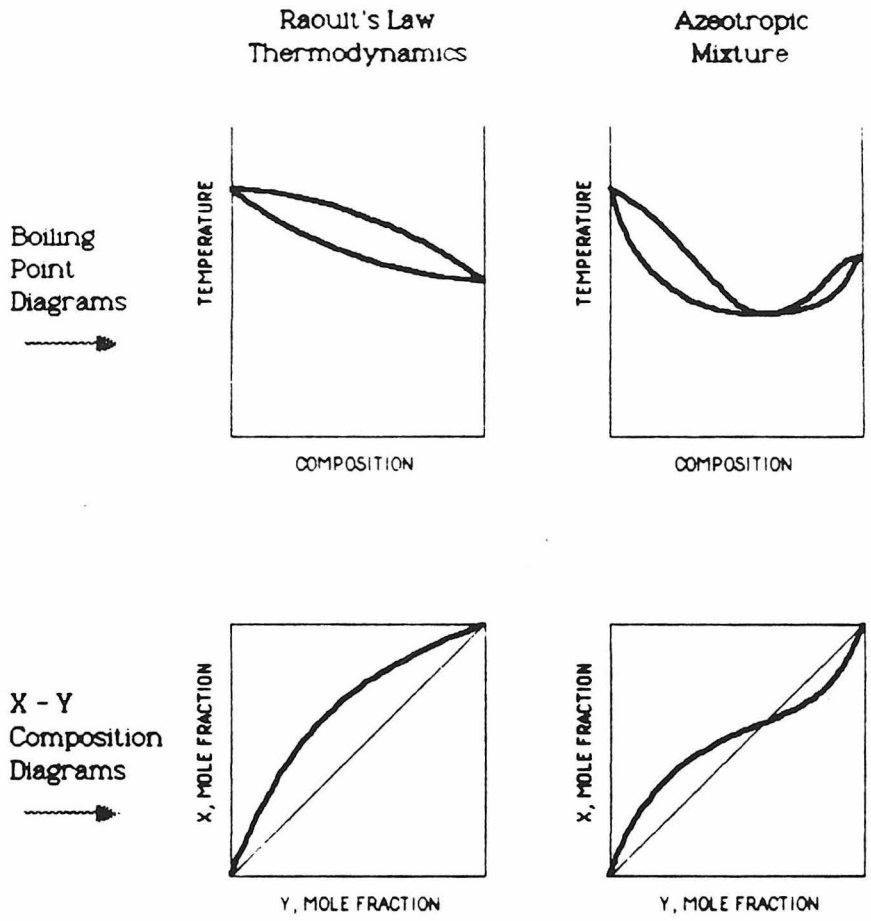

Figure (4) Boiling Point to x-y Diagrams

ideal solution behavior and equal molal latent heats of vaporization. Although both of these conditions are seldom met, the case of constant relative volatility is often used for instructional purposes with the McCabe-Thiele diagram. Using the relative volatility, the equilibrium line is found from a simple one parameter equation.

The vapor-liquid phase equation must be used for a more rigorous thermodynamic representation of the binary mixture.

The phase equation itself has a rather simple form, but difficulties arise as the activity and fugacity coefficients of the two phases are calculated.

Fugacity coefficients may be obtained from an equation of state. Because there does not exist a universally applicable equation of state, judgment must be used in the selection of which equation to use. The most popular and widely used of these is the virial equation of state. Because of its simple mathematical form, the large amount of published first and second virial coefficient data, and the convenient methods for mixture calculations, the virial equation is easier to use than the more complex types of equations.

Cubic equations of state, such as the API-Soave and the Peng-Robinson forms, use component interaction parameters to follow experimental data over larger temperature and pressure ranges. These equations have the great accuracy of some of the more rigorous equation types without the great complexities or computer time demands.

The major limitation of the Benedict-Webb-Rubin type equations is the small number of components for which the necessary coefficients have been found. The large number of constants and the increased complexity involved in these

equations often produce little or no advantage over the Soave or Peng-Robinson equation types. Although the Lee-Kester type corresponding states equations are the most accurate generalized equations available, they also suffers from inherent complexity.

In the liquid phase, it is sometimes not possible to calculate appropriate fugacity coefficients from the equations of state. For these cases, the introduction of activity coefficients and the Poynting correction factor are necessary. The activity coefficients can generally be found from any one of a number of correlations. The Poynting factor is used to transform the liquid phase fugacity from the vapor pressure to the system pressure.

The mathematical simplicity of the Margules, van Laar and related algebraic forms of activity coefficient correlations simplifies the necessary calculations. These forms can also handle some rather nonideal binary mixtures. But, for multicomponent mixtures, ternary or higher interaction parameters are needed. Although it is not directly applicable to immiscible liquid-liquid mixtures, the Wilson equation represents vapor-liquid equilibria very well. One of the biggest advantages of the Wilson equation is its ability to represent vapor-liquid equilibria for

binary or multicomponent mixtures with only binary interaction parameters. This gives the Wilson equation some the of the advantages of the more complicated correlations without any extra work. The NRTL and UNIQUAC correlations provide greater accuracy for liquid-liquid equilibria along with a multitude of additional constants and parameters. But distillation processes are primarily interested in vapor-liquid phase equilibria, therefore the NRTL and UNIQUAC methods provide little or no advantage over the Wilson correlation.

Because the McCabe-Thiele diagram is constructed on a two dimensional surface, only binary mixtures of components are generally considered. But, in practical use, multicomponent streams can be approximated as binary mixtures in situations where the light and heavy key components make up the majority of the feed stream. For more evenly distributed multicomponent mixtures, several methods of thermodynamic property averaging can be used to create a psuedo-binary mixture. Here, the light key and all the other more volatile components are lumped together to form a low boiling group. And, the heavier components are lumped to form another group. The thermodynamic parameters of the two groups can then be averaged to find a set of

effective parameters. These techniques work best on mixtures whose phase equilibria behavior can be described by a single parameter thermodynamic model. For more nonideal mixtures where binary, and possibly ternary, interaction parameters are needed, the appropriate averages are difficult to determine.

3.2 COUNTERCURRENT MULTISTAGE CONTACTING

Now that a graphical representation of the phase equilibria relations have been found, a corresponding representation of the distillation column operating conditions are needed. By developing the generalized material balance equations for countercurrent contacting equipment, an overall graphical description of the operating conditions can be found. These conditions are used with the equilibrium line information for separation process design and analysis.

In a multistage countercurrent operation, the two streams contact as they pass in opposite directions through a cascade of equilibrium stages. The typical arrangement would have a liquid stream falling down the sequence of stages with the vapor stream percolating up through the liquid. This general arrangement is used in distillation, absorption, or extraction operations. Figure (5) represents a generalized cascade of equilibrium stages.

In the above diagram, the variables L and V represent the molal flow rates of the liquid and vapor streams respectively between the stages. The x and y variables denote the mole fraction of the lighter component in the liquid and vapor streams. Because the McCabe-Thiele method is only used for binary or psuedo-binary mixtures, all of the stream composition information can be found from the

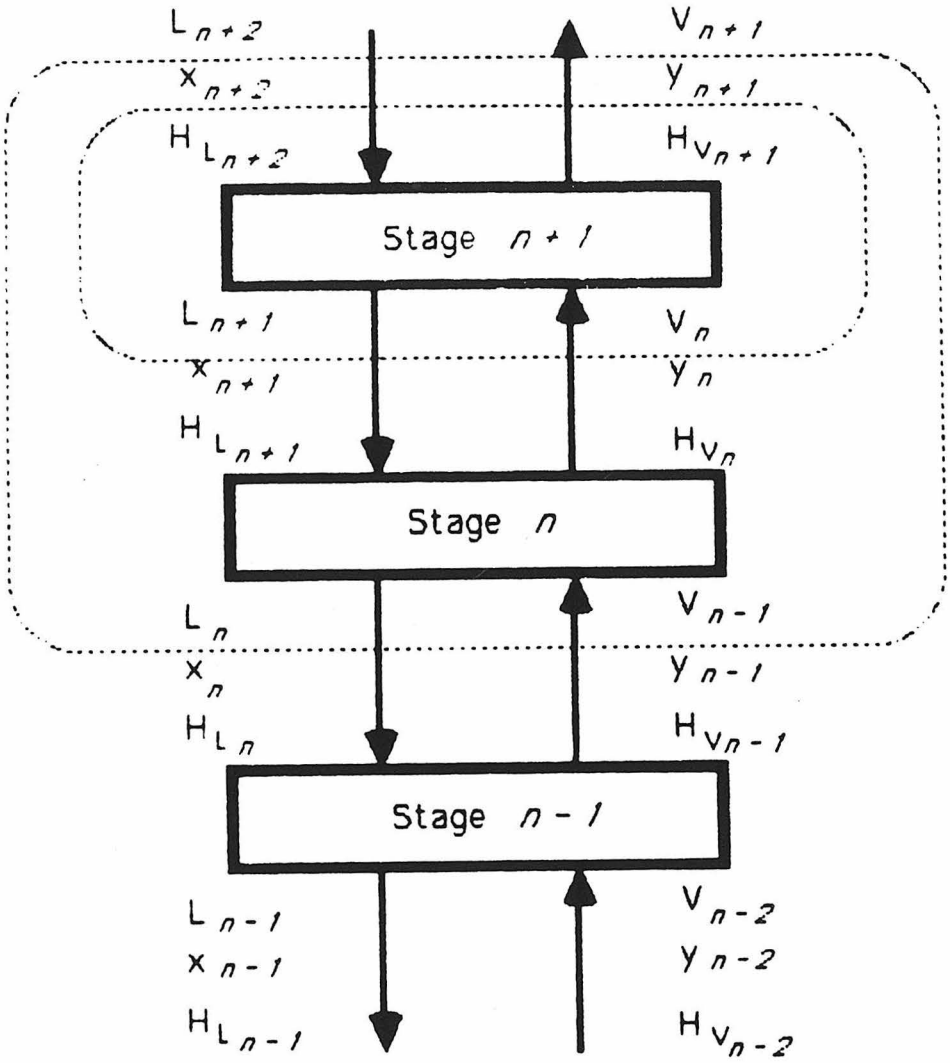

Figure (5) Generalized Cascade of Stages

mole fractions of the lighter component in the each phase. The enthalpy of the streams is represented by the variable H, with the liquid stream enthalpy subscripted with an L and the vapor stream with a V.

Each of the equilibrium stages can be thought of as a continuous stirred tank reactor (CSTR). The compositions of the liquid and vapor phases on each of the stages is assumed

to be homogeneous throughout the stage. Additionally, it is assumed that the residence time of each of the stages is such that a condition of thermal and phase equilibrium is achieved. Therefore, the composition of each stream is the same as that in the stage from which the stream originates.

As a first step in developing a graphical representation of the operating conditions, a material balance is taken around the top tray of the cascade in Figure (5), Stage n+1. The control volume for the balance is shown by the inner dashed box on the diagram. Streams L_{n+2} and V_n enter the control volume while streams L_{n+1} and V_{n+1} leave. The material balance becomes

$$L_{n+2}\, x_{n+2} + V_n\, y_n = V_{n+1}\, y_{n+1} + L_{n+1}\, x_{n+1} \qquad (i)$$

Or, solving for the vapor composition entering the control volume

$$y_n = \frac{L_{n+1}}{V_n}\, x_{n+1} + \frac{V_{n+1}\, y_{n+1} - L_{n+2}\, x_{n+2}}{V_n} \qquad (ii)$$

The numerator in the last term of equation (ii) is the net flow of the lighter component out the top of the cascade. Because this expression will be used later, the variable, NF, will be defined as the net flow of the lighter

component out the top.

$$NF \triangleq V_{n+1} y_{n+1} - L_{n+2} x_{n+2} \qquad (iii)$$

With this new definition, the light component material balance becomes

$$y_n = \frac{L_{n+1}}{V_n} x_{n+1} + \left[\frac{NF}{V_n}\right] \qquad (iv)$$

Another material balance is found for the second control volume shown on Figure (5). This volume includes the first two stages at the top of the cascade. Here, streams L_{n+2} and V_{n-1} enter the control volume and streams L_n and V_{n+1} leave.

$$L_{n+2} x_{n+2} + V_{n-1} y_{n-1} = V_{n+1} y_{n+1} + L_n x_n \qquad (v)$$

Again, solving for the vapor composition entering the control volume and using the defined variable NF, we obtain

$$y_{n-1} = \frac{L_n}{V_{n-1}} x_n + \left[\frac{NF}{V_{n-1}}\right] \qquad (vi)$$

With equations (iii), (iv) and (vi), the locations of

the following points can be found on the x-y equilibrium diagram.

(y_n , x_{n+1}) (y_{n-1} , x_n)

These points represent the compositions of countercurrent streams passing each other between the stages in a cascade. From these points, and other points found in a similar manner, an operating line can be constructed. All of the points representing the compositions of passing streams will lie on the operating line.

The operating line can be straight or slightly curved to represent varying ratios of flow between the passing streams. Although the curved operating lines depict the more general situation, their use introduces an additional level of complexity into the calculations. As with the approximation of no pressure drop through the column, the assumption of constant molal overflow is made to simplify the calculations. Constant molal overflow corresponds to the case of a constant total molal vapor flow rate and a constant total molal liquid flow rate leaving all of the stages in a given section. This simplification provides solutions that are approximately correct for most cases. To find the implications of the constant molal overflow approximation, the energy balance around the stages must be

considered.

Returning to Figure (5), an energy balance for the inner control volume can be found. The energy associated with the streams entering the control volume minus the energy of those leaving gives the following balance.

$$L_{n+2} H_{L_{n+2}} + V_n H_{V_n} = L_{n+1} H_{L_{n+1}} + V_{n+1} H_{V_{n+1}} \qquad (vii)$$

Or, after a slight rearrangement

$$V_{n+1} H_{V_{n+1}} - V_n H_{V_n} = L_{n+2} H_{L_{n+2}} - L_{n+1} H_{L_{n+1}} \qquad (viii)$$

Calculations can be made from equation (viii) to find molar flow rates of the passing streams. The solution of this equation, with its composition dependent enthalpy terms, takes a great deal of computational time and energy. It is here that the approximation of constant molar overflow provides a great simplification.

It can be seen, by considering the energy balance equations, that the internal flow rates are mathematically coupled to the enthalpies of the two phases at each stage. Although the constant flow rate assumption simplifies the calculations, it also places some constraints on the enthalpies of the phases. These restrictions are found as we continue to consider the energy equations.

An expression of the change of the vapor flow rates from stage to stage can be found from the energy equations. First, by adding and subtracting terms on each side of equation (viii), the following expression is found.

$$H_{V_{n+1}}[V_{n+1} - V_n] - V_n[H_{V_n} - H_{V_{n+1}}] = H_{L_{n+2}}[L_{n+2} - L_{n+1}] - L_{n+1}[H_{L_{n+1}} - H_{L_{n+2}}] \qquad (ix)$$

Combining this with the overall material balance around the same control volume from Figure (5),

$$V_{n+1} - V_n = L_{n+2} - L_{n+1} \qquad (x)$$

we obtain

$$V_{n+1} - V_n = \frac{V_n[H_{V_n} - H_{V_{n+1}}] - L_{n+1}[H_{L_{n+1}} - H_{L_{n+2}}]}{H_{V_{n+1}} - H_{L_{n+2}}} \qquad (xi)$$

By letting the left hand side of this equation go to zero (allowing no change in vapor flow rates from stage to stage), the consequences of the constant molar overflow simplification can be found. If the left side of the

equation is equal to zero then the right side must, of course, also equal zero. Since the denominator of the right side can not go to infinity, the numerator must go to zero for the case of constant molal overflow. This can only happen in one of the two following cases.

1) H_V = constant, and H_L = constant

2) $$\frac{H_{V_n} - H_{V_{n+1}}}{H_{L_{n+1}} - H_{L_{n+2}}} = \frac{L_{n+1}}{V_n}$$

A similar derivation using the change in the stage to stage liquid flow rates arrives at the same conclusions.

From the first condition, it can been seen that the constant molal overflow simplification will be exact when the molar latent heats of vaporization of the two components are equal, the sensible heat changes due to temperature changes from stage to stage are negligible, and there is no enthalpy change from mixing effects.

The second condition indicates that the simplification will be exact if the ratio of the stage to stage change in the vapor molal enthalpy to the stage to stage change in the liquid molal enthalpy is constant and equal to the ratio of liquid to vapor flow rates. This corresponds to the situation where the change in the vapor molal enthalpy with

respect to vapor composition is equal to the change in liquid molal enthalpy with respect to the liquid composition. With the above conditions and all of the limitations of the first condition satisfied but the reference enthalpies of the saturated pure liquid A and B were not equal, there would be constant molal overflow and the following expression would be true.

$$\frac{y_n - y_{n-1}}{x_{n+1} - x_n} = \frac{L}{V} = \frac{H_{V_n} - H_{V_{n+1}}}{H_{L_{n+1}} - H_{L_{n+2}}} \qquad (xii)$$

Despite all of the restrictions, the simplification of constant molal overflow is close to being exactly correct for many cases. It is especially good for close boiling and nearly ideal mixtures; and in almost all cases, it does not severely affect the diagram. The computational ease that this approximation permits tends to outweigh the small error it introduces.

With the vapor and liquid flow rates constant, all of the points that represent the compositions of the passing streams will lie on the same straight operating line. From any pair of points on the operating line the slope can be found. The slope will be equal to the ratio of the liquid flow rate to the vapor flow rate (L/V).

Now, the entire operating line with the L/V ratio data

and the composition data for all possible countercurrent streams can be found from a minimum of information. The operating line can be constructed from (1) knowledge of the L/V ratio and the compositions of one set of passing streams or (2) knowledge of the compositions of two sets of passing streams. The passing stream information can come from the inlet and outlet compositions at one end of the cascade of stages. Or, in the context of a series of distillation trays, the compositions of a distillate or bottoms stream.

With the phase equilibria equations and the operating equations now expressed in graphical form, a representation of the feed stream's thermal condition and physical location on the column is now needed. On the partial McCabe-Thiele diagram in Figure (6), the operating conditions for a single feed distillation column have been set up. All of the operating lines of the column lie below the curved equilibrium line. The top of the column is represented by the rectifying section operating line. From the diagram, we find that the distillate stream is saturated liquid with a mole fraction of the light component x_D. The slope of the rectifying line gives the ratio of the internal flows (L/V) in the upper section of the column. The bottom of the column is represented by the Stripping section operating line. This line shows the bottoms stream being saturated liquid with x_B as the mole fraction of the lighter

component. The slope of the line gives the ratio of internal flows in the stripping section of the column (L/V). The Feed line on the figure shows the total mole fraction (mole fraction of all phases combined) of the feed to be z_F. From the slope of the feed line, the thermal condition of the feed stream can be found.

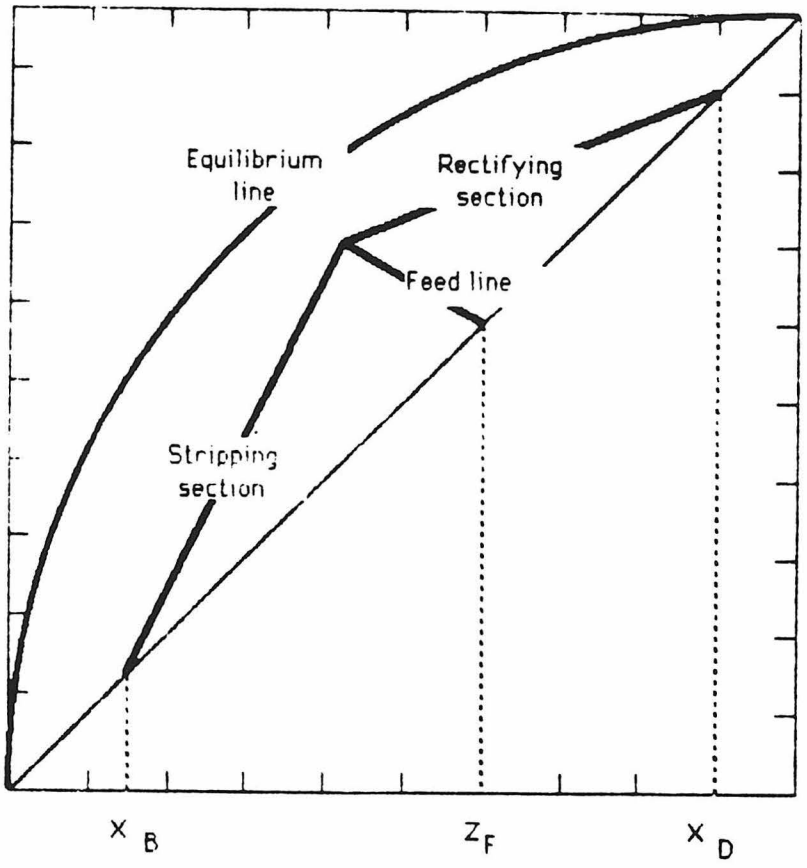

Figure (6) Partial McCabe-Thiele Diagram

The relation between the feed thermal condition and the slope of the feed line can be found by taking an energy and material balance around the feed tray in the column. Assuming that the molal enthalpies of the vapor and liquid phases on the trays above and below the feed tray are equal to those on the feed tray, these equations give a correlation between the thermal condition of the feed and the magnitude of the variable q. The definition of the variable q is as follows.

$$q = \frac{H_{V_f} - H_F}{H_{V_f} - H_{L_f}} \qquad \text{(xiii)}$$

Equation (xiii) states that the variable q is equal to the amount of enthalpy necessary to bring the feed to the saturated vapor state divided by the feed stream's latent heat of vaporization. Also one finds that the slope of the Feed line is equal to q/(q-1). Feed lines for various magnitudes of q are shown in Figure (7).

Now that the phase equilibrium line, the feed line, and the operating lines for the rectifying and stripping sections have been defined, the number of theoretical trays necessary for the specified separation can be found. First, recall that the equilibrium line contains all of the phase information for the binary mixture at its equilibrium state.

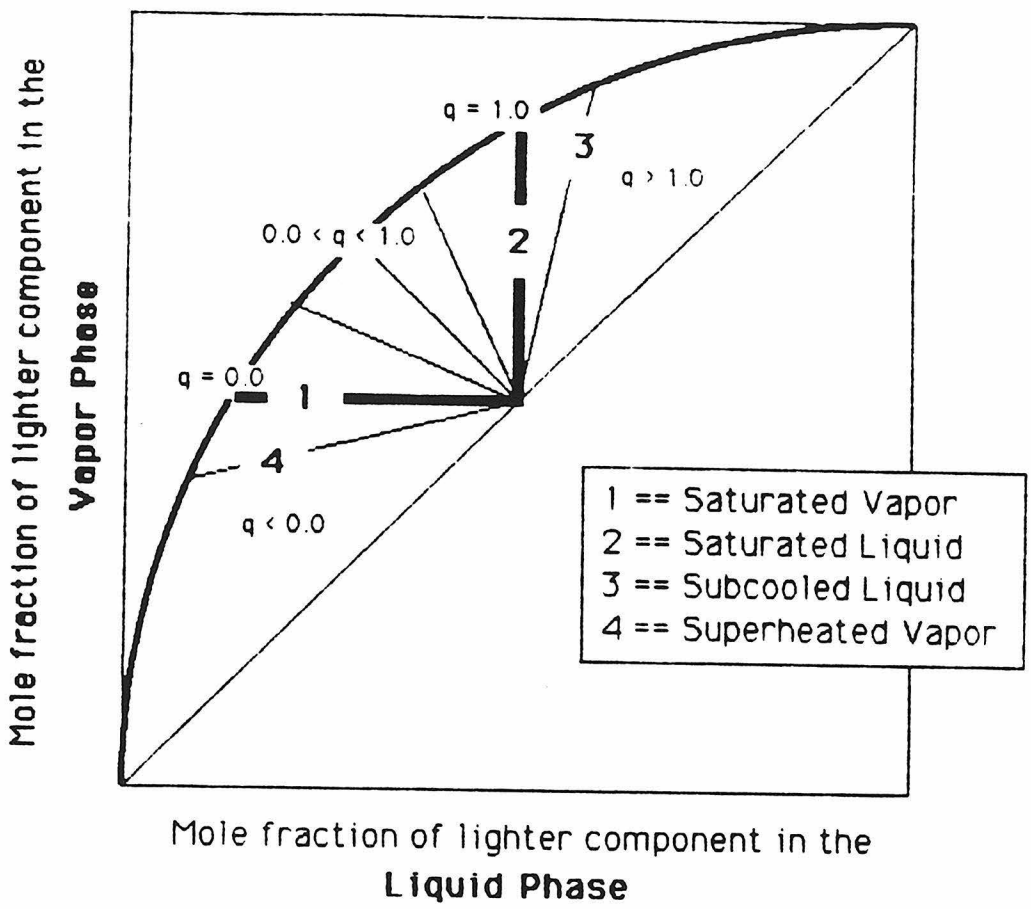

Figure (7) Feed Lines for Various Values of q

Therefore, the points on the equilibrium line will correspond to all of the possible phase compositions on any of the trays. Also recall that the operating lines are defined in such a way that they contain all of the possible compositions of countercurrent passing streams between the trays. By stepping off the trays like a stairway between the equilibrium and operating lines (as shown in the print out from the MCTHIELE program, Figure (8)), the number of

theoretical trays needed for the desired separation is found.

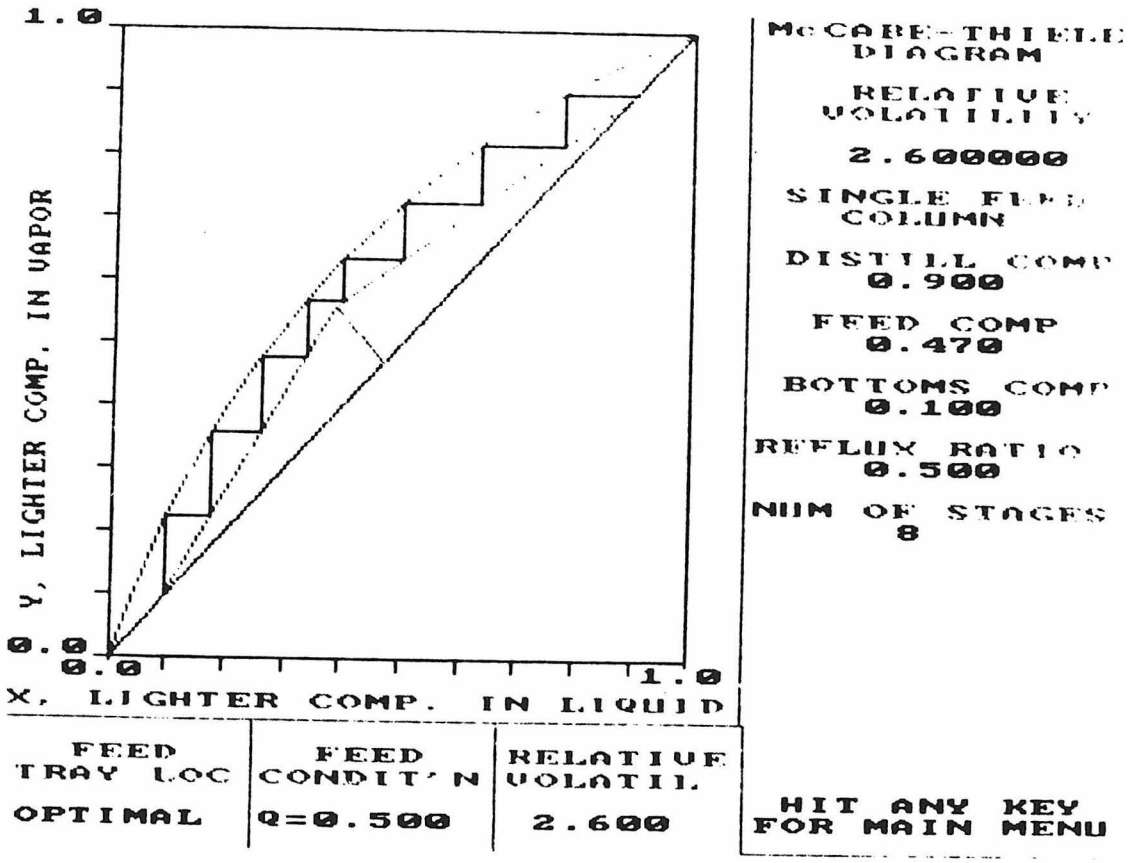

Figure (8) Optimal Feed Tray Location -- MCTHIELE Printout

To step off the stage requirement, start at the distillate point at the top of the rectifying section line and move horizontally across to the equilibrium line. This point represents the composition of the mixture in the top tray of the column. Moving vertically down to the operating line, the point corresponding to the composition of the countercurrent passing streams between the top two trays is

reached. Then, moving across to the equilibrium line, the mixture composition in the second tray is found. Moving down to the operating line again, one finds the composition of the passing streams between the second and third trays. Stages continue to be stepped off until the bottoms composition is reached or overshot. For the column in Figure (8), it is seen that eight theoretical trays are needed.

In practical application, the condenser on the top of a distillation column will be considered the top theoretical tray. Depending on the type of heating system, it has been reported that the reboiler can act as one to two theoretical stages.

The point where the stages change from stepping off on the rectifying section operating line to the stripping section line marks the location of the feed tray. From Figure (8), the optimal feed stage location for this column is on the fifth stage from the top. If a non optimal feed tray location is used, the column will need more trays to accomplish the same separation. This is shown the following two figures. In Figure (9), the feed is introduced to the column at the seventh tray from the top of the column. In Figure(10), the feed tray is located on the fourth tray from the top (seventh from the bottom). In both of these situations, the required number of trays to affect the same

separation increases from eight to ten.

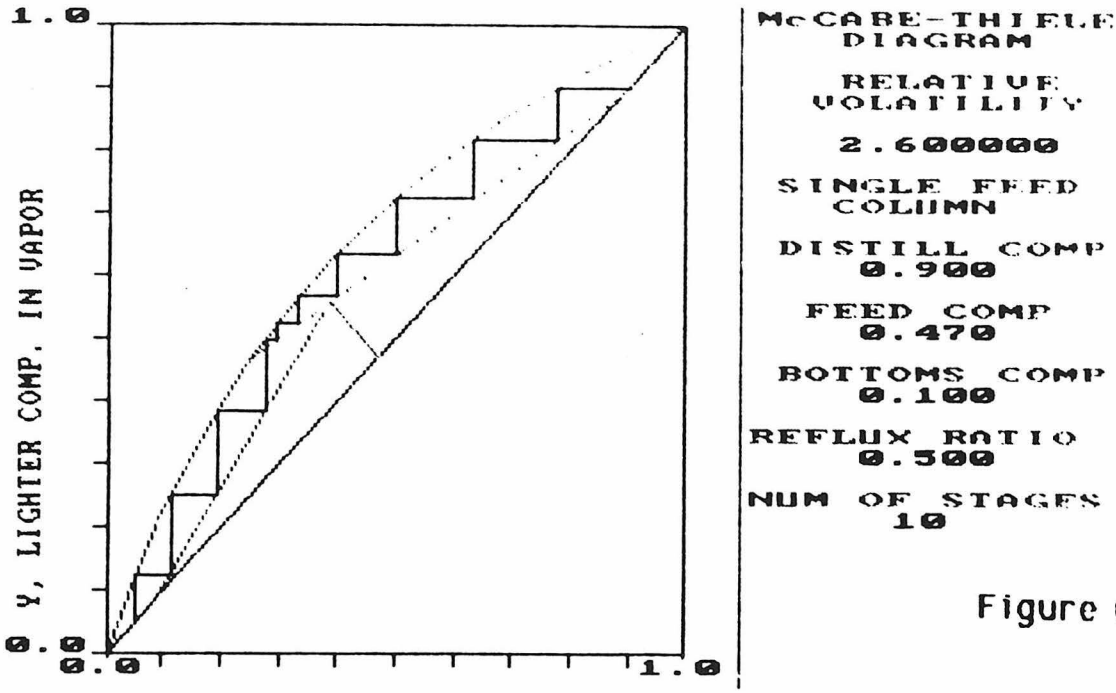

Figure (9)

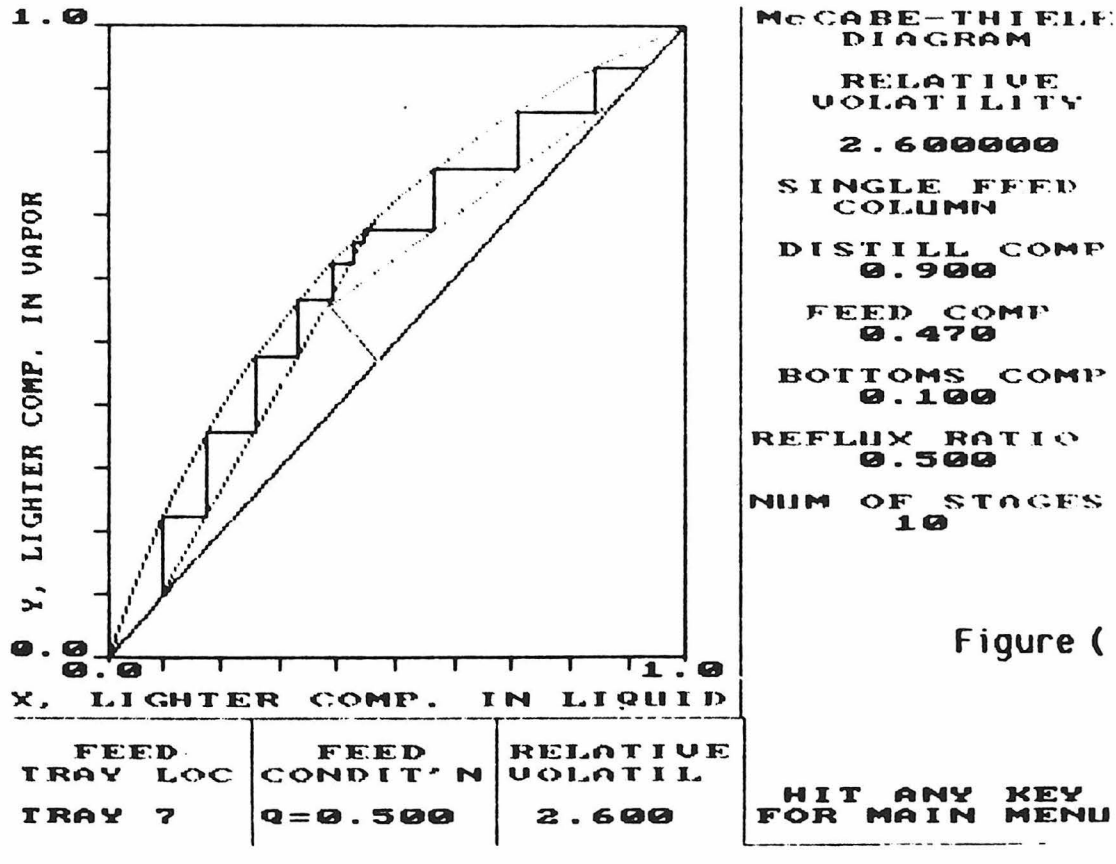

Figure (10)

The operation of all columns, theoretical and real, lie between two limiting operational conditions. The first of these limiting conditions is operation with the minimum number of stages. In order to minimize stage requirements the reflux ratio is adjusted to allow the greatest possible area between the equilibrium and operating lines. This occurs with an internal reflux ratio (L/V) equal to one. Under these conditions, the operating line falls on the 45 degree, x = y line. Operating at this condition, called total reflux, there is no product stream withdrawn from the condenser or reboiler. An example of this situation is shown in Figure (11). Note that the reflux ratio given in the figure is the external reflux ratio (distillate stream flow rate divided by the returned liquid stream flow rate). An external reflux ratio (D/L) of zero corresponds to a internal reflux ratio (L/V) of one (no distillate stream flow).

The other limiting condition is operation with the minimum internal reflux ratio. Because the operating lines can never go above the equilibrium line, the minimum internal reflux (or minimum rectifying section line slope) condition will occur when the intersection of the operating line and the feed line lie on the equilibrium line. Or, in the case of a complex equilibrium line, the operating line becomes tangent to the equilibrium line. The first method

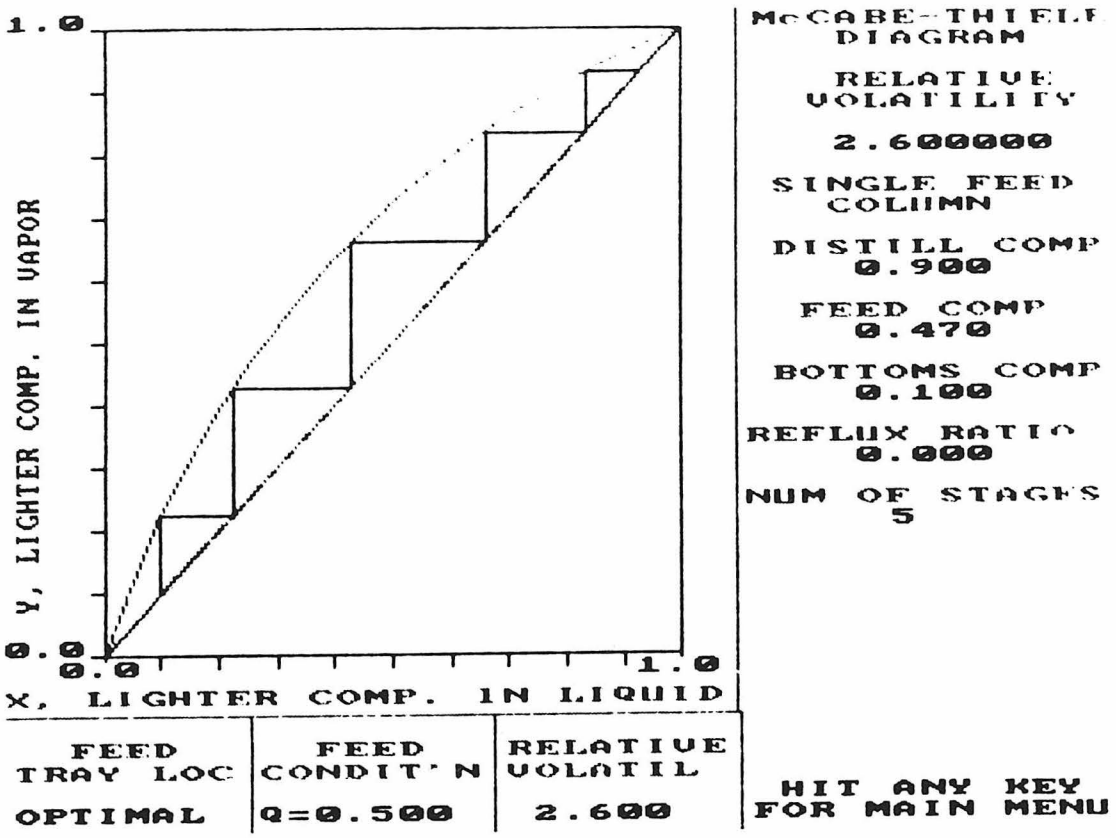

Figure (11) Total Reflux Condition -- MCTHIELE Printout

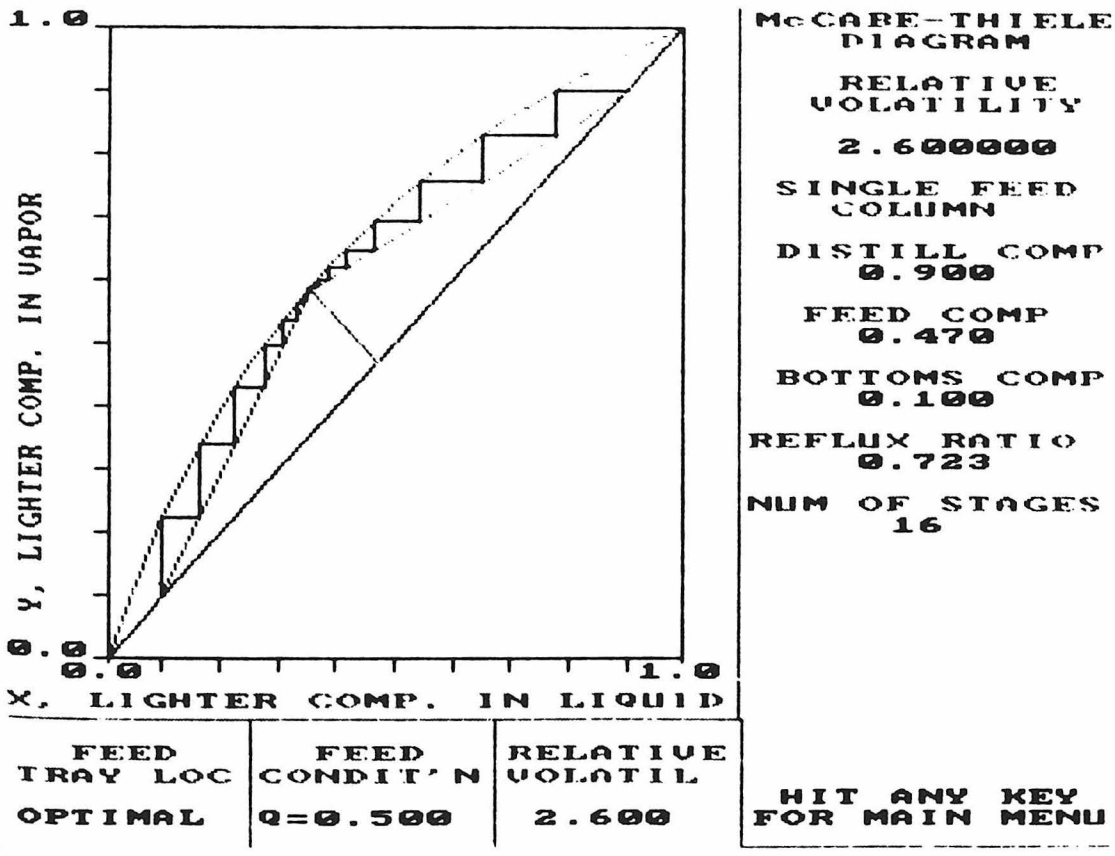

Figure (12) Minimum Reflux Condition -- MCTHIELE Printout

of minimum reflux occurs much more frequently. If the operating and equilibrium lines actually touched, it would theoretically take an infinite number of trays to past the pinch point between the two lines. A near minimum internal reflux ratio condition is illustrated in Figure (12). Again, remember that the reflux ratio on the figure is the external reflux ratio.

The limiting condition of minimum stages requires an infinite (or total) reflux ratio and the condition of minimum internal reflux requires a infinite number of stages. As is seen in Figure (13), the optimal conditions to operate a distillation column will lie somewhere between these two extremes.

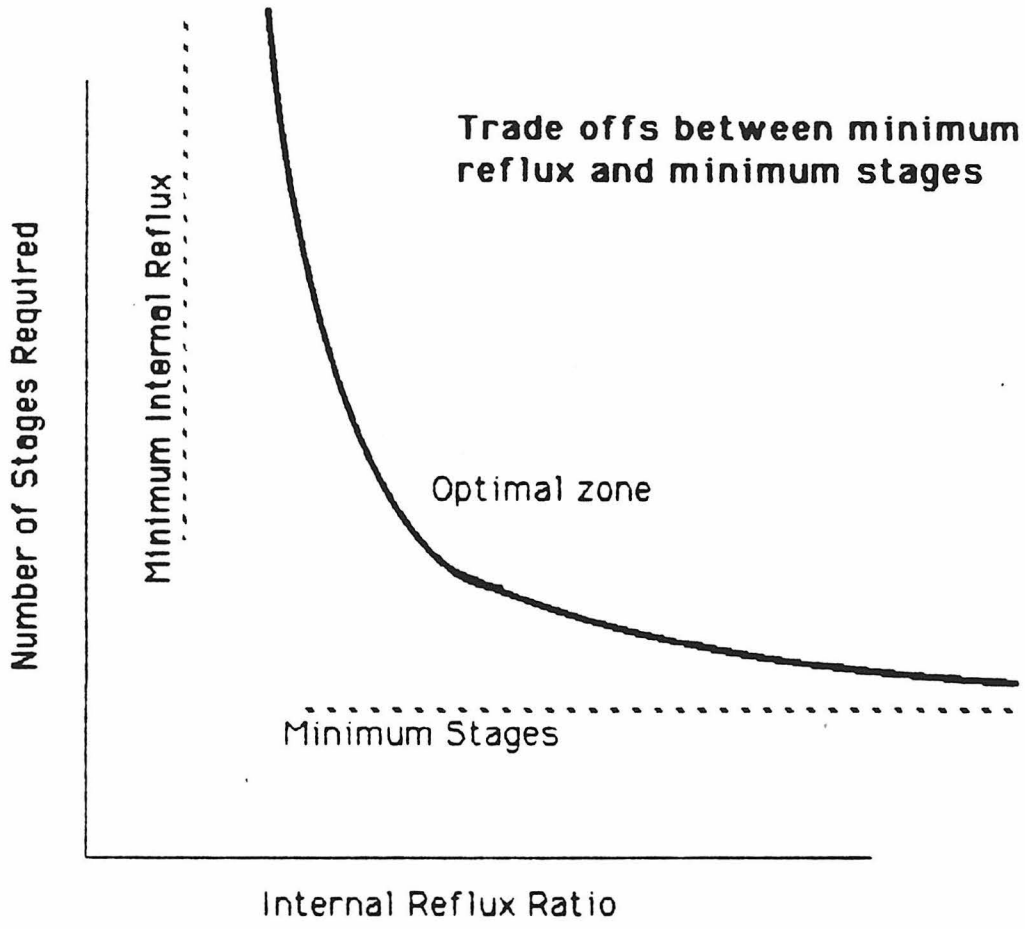

Figure (13) Minimum Stages vs. Minimum Reflux

3.3 EDUCATIONAL SOFTWARE DEVELOPMENT

The major goal in the development of the McCabe-Thiele design program was to create a good educational tool. Because of the graphical nature of this design method, the instructional possibilities of this computer application were promising. Because of the mixed acceptance of computer aided instruction tools in the past, a great deal of time was spend defining the features that make an educational program successful. Before this could happen, the learning process itself had to be considered. For if the methods of learning and information transfer could be investigated and categorized, the goals and features of a good educational program could better be defined.

Psychologists studying the learning process have defined two different methods of information transfer. The first of these learning modes involves written or verbal communication. This would include learning and understanding a subject through the use of books or verbal explanation. The second mode of learning uses visual and observational skills. These skills are used in learning a new task by a visual experience or pictorial representation of the action being performed. This idea of dual mode learning is also supported by biologists studying the fundamental division of the brain. They have proposed that the left brain is more analytical and language based whereas

the right brain is more abstract and visually based.

In this case, the goal is to teach chemical engineering students the fundamentals of multistage separation processes. In the teaching of any complex subject such as this, different levels of learning are experienced. The first level of information transfer is the explanation of the mathematical and scientific knowledge necessary to rigorously solve the separation problem. This involves the development of phase equilibrium relations, the formulation and solution of the material and energy balance equations, and the mathematical expression of operating conditions and restrictions. By the careful combination of these expressions, the solution to most equilibrium stage separation problems is found. The second aim is to develop the students' sense of intuition. One of the traditional ways to accomplish this is to cover, in homework and lectures, a large number of problems that show variations of the process. But when the problems become more complex, it is more difficult to cover a wide range of problem types. Unfortunately, it is in these complex problems that the development of a sense of intuition is perhaps most important.

Written and mathematical languages are certainly the best way to address the needs of the first level of learning. By deriving the equations that govern the physics

of the problem, the process is converted to a common mathematical language. In this form, the somewhat abstract concepts can be manipulated and sorted into a structure that the language based mind can understand and work with. Combining phase equilibrium equations, energy and material balances, and operation condition descriptions in the mathematical language, the solution to every definable problem can theoretically be found to an arbitrary precision.

Although an extremely rigorous and exact solution can be found to any problem using the language based methods, it is not always easy to understand the implications of the results when they are conveyed as a list of numbers. Often, more important than the exact answer, is the answer's functional dependence on problem variables. Although analytic solutions can be found, the solution's dependence on input variables is difficult to draw from the highly nonlinear equations that arise in multistage separation processes.

At this point, the second level of learning, the development of an intuitive feel for the problem, needs to take place. As the complexity of the system increases, a certain degree of engineering intuition into the problem becomes necessary. For example, consider running a chemical plant flow sheet on a mainframe process simulation program.

If, after the first run through the program, the desired product purity specification is not met, one must decide which parameters to change in the plant to increase the purity without adversely affecting the rest of the plant. Although the local consequences of each parameter change may be known, the overall effects are often unclear. A certain degree of engineering intuition must be exercised to make a intelligent guess at the appropriate changes.

In the context of distillation columns, one might wonder what the effect of changing the reflux ratio would be. Although all of the relevant equations could be known, a student may not have the "feel" for the problem that an experienced engineer would have. Students familiar with a graphical solution technique such as the McCabe-Thiele method can complement their knowledge of the equations with a graphical representation of the problem. With this type of thought process, the student gains insight by learning to think in the visual language of the McCabe-Thiele diagram.

From these considerations of the learning process, a number of conclusions and goals can be defined with regards to the development of an educational program. First of all, the graphical and visual elements of the McCabe-Thiele method should be emphasized and exploited. The option should be available to make all column specifications directly on the diagram to encourage the mental translation

of language based concepts to visually based representations. Secondly, all of the mathematical calculations made during program execution should be echoed on the displayed diagram to reinforce the relationship between the graphical manipulations and the mathematical calculations. Thirdly, all of the information shown on the diagram should be backed up by the concurrent display of its language based counterpart. This will help avoid any misunderstanding of the visual information.

Along with the above goals, there are several other key features of good educational software that were included in the McCabe-Thiele program. Although there have been numerous attempts to produce computer learning tools, many of these are not used or accepted because of their shortcomings in several areas. The five key features of any educational program are that it should be: general, flexible, fast, understandable, and easy to use.

The first of these features is the development of a general educational tool. Rather than being specific to a single problem type, the program should be general enough to solve a large family of possible problems. One of the outstanding features of the MCTHIELE program is the ability to solve for a number of the specified variables of the diagram yielding over twenty five different problem types.

The second key feature of a good educational program is

flexibility. The user should not be restricted to the programmer's idea of the "correct" technique for problem solution. The program should be flexible enough to handle methods of solution that the programmer has not thought of. For graphical solution techniques, the computer screen should be used like a pencil and pad of paper. For example, in this McCabe-Thiele program, the user can point and move lines on the screen much like he can do with a graphics editor. After the desired equilibrium and operating lines are drawn, no matter how unexpected or strange looking, the solution to the defined problem will be found.

The third consideration is computational speed. The calculation and display of problem solutions needs to be fast enough to allow the user to see a wide variety of problem types in a short amount of time. In this way, a number of solutions to problems with incremental changes in a single variable can be viewed in order to see the dependence of the solution on any of the variables. In this manner, the trends and patterns of various factors can be seen. In the context of the McCabe-Thiele program, the solution procedure is fast enough that an entire sequence of diagrams showing the full range of operating conditions all the way from minimum to total reflux can be generated in seconds.

A good educational program should be easy to

understand. After all, the audience may know little about the program's topic. Remember, teaching the subject is the whole idea behind an educational program. But, on the other hand, the educational program's audience may consist of experienced users. Therefore, the program should be easy to understand but it should also avoid being simple-minded. A good educational program should grow with its user and allow advanced problem types without demanding them. Also important for clear understanding, is the presence of complete and informative error messages. The user should not need any "inside" information to decipher help messages.

And, finally, an educational program has to be easy to use. The users should not find themselves at any point in a program where the next move is not clearly spelled out on the screen. And, preferably, the screen should suggest a variety of options if appropriate. Additionally, the use of a mouse or other pointing device, greatly improves the ease of use of any program. If a program is not easy to use, it will not be used.

To recap, a good educational program must be general, flexible, fast, understandable, and easy to use. All of these features combine to generate a supportive computing and learning environment. Other, less crucial, factors add to the comfortable computing atmosphere created by the five main features above. All possible user errors should be

explained and then a second chance should courteously be extended. There should be no unrecoverable errors. Also, the program should not punish its users for any wrong move. If the programmer feels compelled to exercise the sound capability of the hardware, the "beeps" should be reserved for congratulations on a job well done rather than used as a method to scold users.

3.4 MCTHIELE PROGRAM

Utilizing the ideas about the learning process and the features of a good educational program presented above, the McCabe-Thiele program, MCTHIELE, was created. Like the multicomponent multistage distillation program, SEPTECH, before it, this program was developed on an IBM PC/XT under the C language with HALO graphics support. This language and graphics software were recommended as a combination that would be compatible with the widest variety of hardware arrangements. The upward mobility of the C language under the UNIX systems was also considered. Because of its general availability the IBM personal computer was used for development. The first versions of the MCTHIELE program were written to take advantage of the high resolution, sixteen color modes of the AMDEK graphics board. Later versions are available for use on IBM standard and extended graphics boards.

Because the MCTHIELE program was designed as an educational tool, the flow of calculations, the input routines, and the problem set ups are not constrained to a predetermined order. Rather than following a linear flow through the input procedure as was done in the SEPTECH program, specifications are made with the use of menus and option blocks. Because of this lack of an inherent computational flow, the program can follow any order

preferred by individual users.

The overall structure of the MCTHIELE program is depicted in Figure (14). This diagram can be used as a "road map" to understand the flow of the program. The two main locations of the program's outer shell are the Main Menu and the Display screen. One also notices the central location of the Work Place Screen in the program. It is here that the majority of the computational time is spent constructing and solving various problems. The rest of the program acts as a supporting cast to the activities of the Work Place Screen.

DISPLAY

The Display screen is where the program begins when it is first accessed. This screen displays the current status of the McCabe-Thiele diagram under construction. The latest equilibrium and operating lines along with the most recent solution are shown here. To the right of the updated diagram, the numerical values of all of the variable are shown. Below the diagram, the status of the column conditions are updated. All of the printouts shown in the previous sections of this report are from the Display screen.

In the bottom right corner of the Display screen, the user is informed that hitting any key will bring up the Main

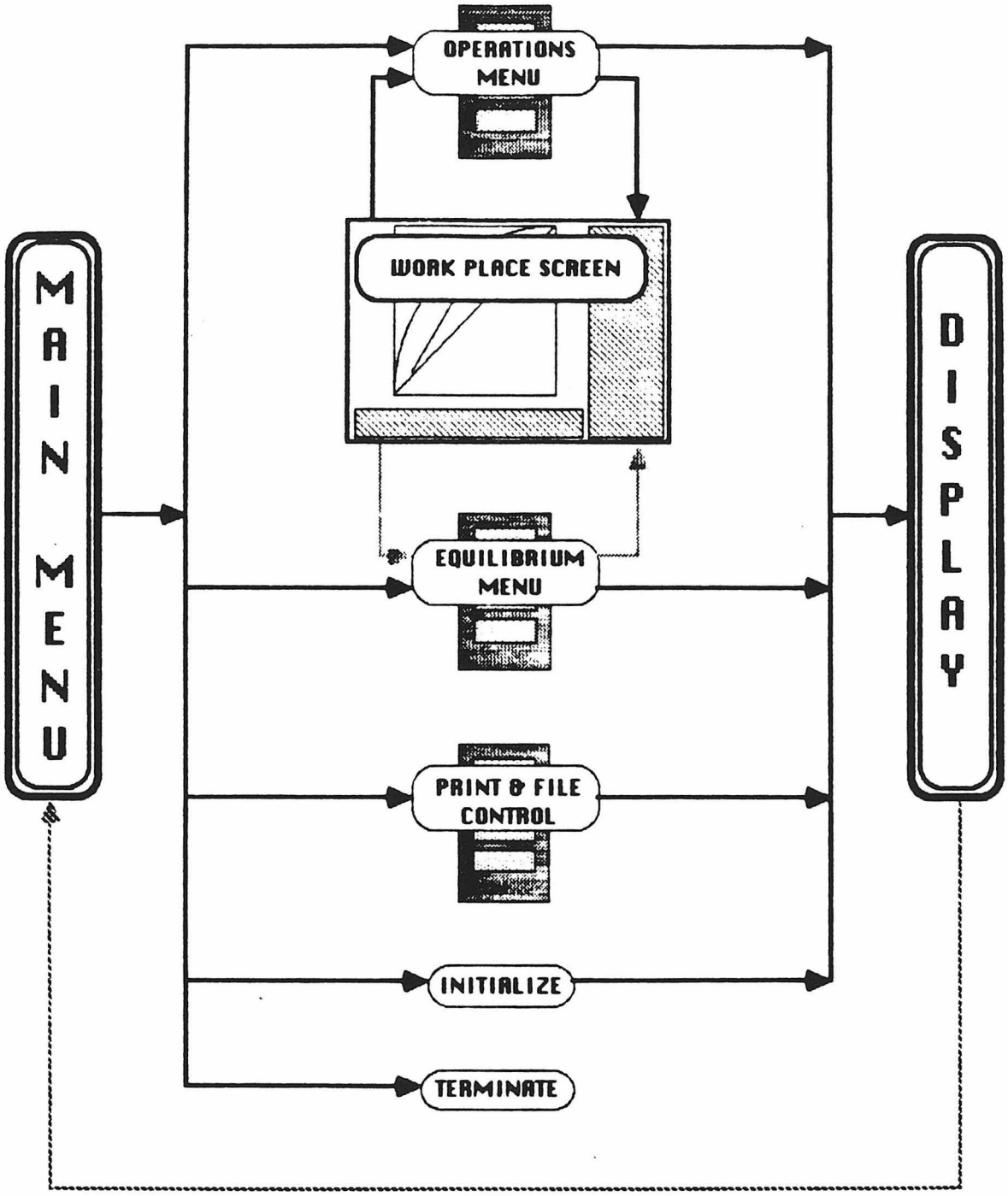

Figure (14) Road Map of MCTHIELE Program Operation

Menu. This is the second of the major, outer shell locations. As one would expect, this menu supplies the user with a variety of options. Looking back to Figure (14), it can be seen that the path back to the Display screen can follow a number of different paths. Unlike many other programs, the various possible tasks between the Main Menu and the Display screen are all in parallel, there are no serial elements to these paths. This allows the user total freedom of problem specification. The program does not force a predetermined order of task execution onto the user.

MAIN MENU

On the Main Menu there are five options: Operations Menu, Equilibrium Menu, Print and File Control, Initialize, and Terminate. As can be seen in Figure (14), the first three of these choices leads to another menu. The Initialize command performs a task before continuing to the Display screen and the Terminate option ends program execution.

*OPERATIONS MENU

The Operations Menu is the pathway to the Work Place Screen. On this menu, the options of three different column types are given: Rectifying Column, Stripping Column, and Single Feed Column. After the desired column type is chosen, the program enters the Work Place Screen

to begin diagram construction. Recall that the Rectifying and Stripping Column types are simply the top and bottom halves of the more general Single Feed Column.

*EQUILIBRIUM MENU

The equilibrium line is defined using the Equilibrium Menu. Although the Main Menu is the primary way to access this menu, it can also be called from the Work Place Screen. Although the Equilibrium Menu can be reach directly from the Work Place Screen, the opposite is not true.

**RELATIVE VOLATILITY

The first option of this menu is a Relative Volatility definition. For instructional use, this is the fastest and least troublesome way to specify an equilibrium line. After the relative volatility is entered through the keyboard, the appropriate equilibrium line appears on the diagram.

**USER DEFINED

The second option is a User Defined equilibrium line. Individual points of the desired line are entered either through the keyboard or with the mouse pointing device. Through the keyboard, the points are typed on the screen using x and y coordinate specifications. The numerical values are entered with the return key. With the mouse, the user moves a

crosshair cursor over the displayed diagram with the current x and y coordinates shown in the upper right corner of the screen. Once the desired coordinates have been located, a click on one of the mouse buttons will store the point. After using either method to select all of the points, the program then sorts the data, removes redundant points, and creates the user defined equilibrium line.

**THERMODYNAMIC DATABASE

The third of the equilibrium definition options is not currently available. The Thermodynamic Database has been constructed from the list of thermodynamic parameters listed in the back of Henley and Seader's book [7]. But, because of the absence of interaction parameters, a regular solution theory or a modified Raoult's law approach is the most sophisticated thermodynamic method that could be used with this data. Because the use of these methods should be restricted to fairly ideal mixtures and the database included many polar and hydrophobic species, it was felt that equilibrium curves derived from these methods could be rather incorrect and misleading. Rather than providing curves that would be approximately correct a fraction of the time, it was decided to postpone the introduction of this option until a more thorough

database could be constructed.

*PRINT AND FILE CONTROL

The third option on the Main Menu is the Print and File Control. From here, all of the file manipulations and diagram printing takes place.

**SAVE DIAGRAM

The Save Diagram option is the first on this menu. With this option, the present diagram and all of its variable information are saved in a disk file. The diagram can be saved at any level of completion. Files can be saved on any disk drive the computer has access to. Warnings will be sent back if the specified disk is too full or if a file with the specified save name already exists.

**RETRIEVE DIAGRAM

The second option is Retrieve Diagram. As its name would suggest, this option will recover a previously saved diagram from a disk file. The diagram to be retrieved could be a completed or partially completed diagram saved during the present or any previous MCTHIELE session. If the specified file is not found, the option to search other disk drives will be offered.

**PRINT DIAGRAM

The Print Diagram option sends the current diagram

to the graphics printer. When a diagram is to be printed, the program asks the user for confirmation that the printer is turned on and attached. The purpose of the conformation step is to help users avoid unnecessary down time while unwanted diagrams are printed. As with most graphics printers, once the printing has been started, the is no civilized way to stop the procedure. Currently the Epson, IBM, Okidata, and compatible graphics printers are supported by MCTHIELE.

**PRINT WITH GRID

The fourth option, Print With Grid, also allows hardcopies to be made of the diagrams. With this option, the diagram will include a grid with lines at one tenth of a mole fraction intervals.

*INITIALIZE

To begin a new problem by starting over from scratch, the Initialize option is selected from the Main Menu. The Initialize command will not only unspecify all of the operation variables, but it will also undefine the equilibrium line and set the operation conditions to their initial values. Once this command has been executed, all onscreen information is lost. Of course, all data saved in a disk file will still be accessible.

*TERMINATE

This final option from the Main Menu will halt program execution. The graphics screen will be closed and the system prompt will return to the text screen.

WORK PLACE SCREEN

A pictorial representation of the screen fields in the Work Place are shown in Figure (15). In the upper left portion of the screen is the largest of the three fields, the Diagram Field. This is where the McCabe-Thiele diagram construction takes place. The status of the of the current problem is always displayed here and all iterative calculations are shown to illustrate the convergence to the solution. The Diagram Field is also used for variable specification. If a mouse is in use, the option will be given during every variable specification to use the mouse to point at values directly on the diagram. Distillate, feed, and bottoms compositions, operating line slopes, and equilibrium line points can be selected by using the mouse to "point and pick".

Directly below the Diagram Field, lies the Condition Field. Access to this field is made by selecting the Change Conditions option from the Option Field. In the three boxes of the Condition Field, the defining diagram information is found. The first box contains the feed tray location information. If the optimal feed tray is being used in the

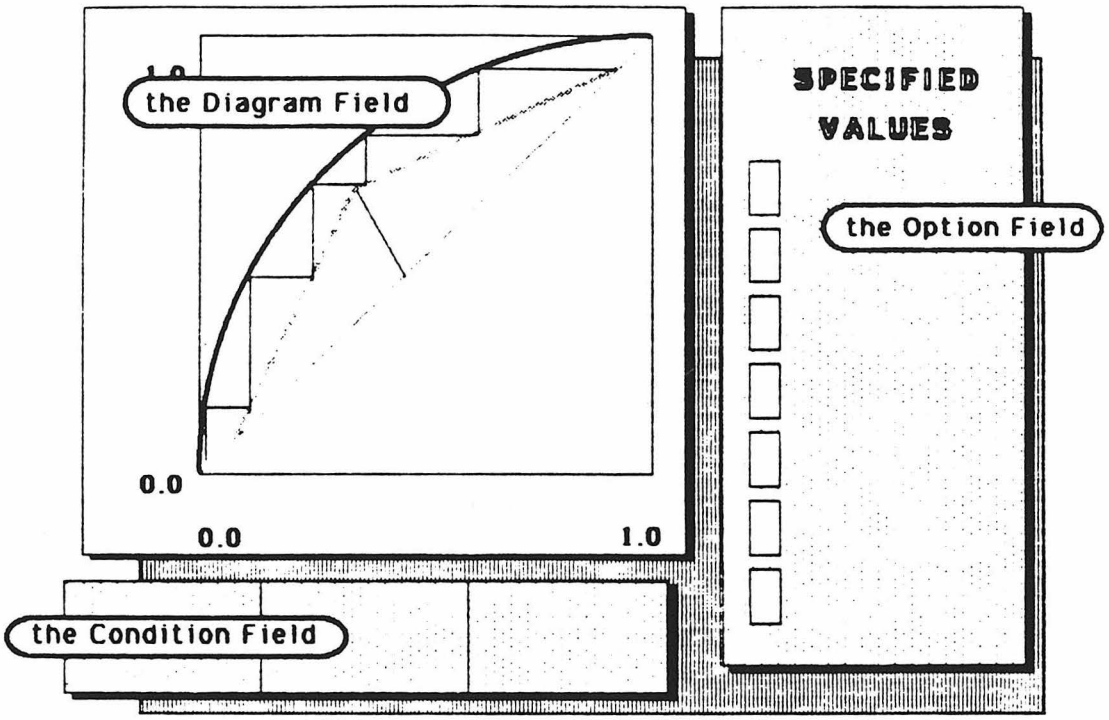

Figure (15) Work Place Screen -- MCTHIELE Program

calculations, the word "optimal" will appear under the "feed location" title. If a specified tray is being used for the column, the tray location will be displayed. The feed tray number can be specified using a top to bottom or bottom to top numbering convention.

The second box of the Conditions Field contains the thermal feed condition data. The variable, q, as defined earlier in this report, is used as a measure of the feed

condition. The initial value of the q variable is zero corresponding to a saturated vapor feed condition. This variable can be changed to reflect any phase condition from subcooled liquid to superheated vapor.

The equilibrium line definition appears in the third box. Relative Volatility, User Defined, and Thermo Database equilibrium types are identified here. The numerical value of the relative volatility is also displayed when appropriate. By choosing this box after the Change Condition option, the program advances to the Equilibrium Menu. When the Equilibrium Menu is entered in this fashion, the Work Place Screen will be returned to after the desired changes have been made.

The Option Field occupies the right third of the screen. It is here that all of the variable specifications are made. In the case of the Single Feed Column type, this field will contain the five diagram variables, the Begin Calculations, the Change Condition, and the Return options.

After the equilibrium line, the feed condition, and the feed location type have been specified, the five variables of the Option Field are all that is left to totally specify the Single Feed Column diagram. Of these five variables (distillate, feed, and bottoms composition, reflux ratio, and number of trays), at least four must be specified before any calculations can begin. The specification of the three

external compositions can be made through the keyboard or by pointing to the desired values on the Diagram Field with a mouse. As the mouse is being used as a pointer, the numerical value of the mouse's current diagram location is given to aid in precise variable specification. Either the internal or external reflux ratio definitions can be used to chose the diagram reflux ratio.

As each of the variables is specified, the value is shown on the Option Field directly under the variable title. Also, a blue asterisk appears next to the selection box of the variable to again indicate that the value has already been chosen. After four of the variables have been specified, the calculation of the remaining value can begin. Once a solution has been found, the Diagram Field will display the current diagram and the value of the variable will appear in the Option Field as the final variable specification.

At any time during this procedure, any of the values of the variables can be changed. If an already specified variable is chosen, the following three options will be offered: Change Value, Unspecify Value, or No Change. The option to unspecify variables allows the user to explore many different situations with a minimum of extra effort. For instance, a first problem solution can ask for the number of trays necessary to achieve a desired separation at

a chosen operating condition. After the number of trays has been solved for, the reflux ratio can be changed and the Number Of Stages variable can be unspecified. Now the new problem with a slightly different operating condition is ready for solution. In this way, a range of solutions with varying reflux ratios can generated quickly and easily. Or, as a second example, the solution for the necessary number of trays can be found and then the number of trays can be reduced to simulate the effect of tray fouling and the distillate composition can be unspecified. The solution to this problem would show the reduced distillate product purity caused by tray fouling of the previously designed column.

This freedom in variable and problem specification allows the solution of a wide variety of problem types. Parametric studies of different distillation variables are easily performed. The trends and patterns of various separations processes become clear and the exploration of new areas becomes invitingly easy.

3.5 CLASSROOM EXPERIENCE

The MCTHIELE program was used to supplement instruction in a senior level chemical engineering class at Caltech. This class, titled Separation Processes, was offered during the fall term of the 1985-1986 academic year.

During the previous year, the separations course had been targeted for the introduction of computer aided instruction. Traditionally, this class had been a problem oriented introduction to separation processes. But last year the course included instruction in the C programming language and computer homework sets were assigned. The purpose of the introduction of computers to this course was to teach problem statement, organized solution procedure, and computational skills. Along with the computer homework assignments, a computational term project in the field of separation processes was required.

Aside from the term projects, all of the computer programs used in the course during the previous year were developed by the students for individual homework assignments. Although these short, disposable programs were able to teach concepts pertaining to the narrow area of chemical engineering that they were designed for, the need for some well developed educational tools became apparent. A mature program that had extensive error checking for robust performance with novice users and that covered a wide

range of problem types was needed. The MCTHIELE program was designed to fill this need.

During the second year of the computer enhanced course, the MCTHIELE program was supplied to all of the students for use on a variety of homework problems. After one laboratory session for explanation of the program's use, disks with the program and pre-stored partial diagrams were handed out. The homework assignment that was assigned that week involve the pen and paper construction of two McCabe-Thiele type diagrams along with the computer construction of approximately thirty other diagrams.

The assignment of the two diagrams to be drawn by hand was to insure that the concepts of the McCabe-Thiele diagram were understood. This was to eliminate the possibility of the ignorant user; one who can use the program to generate numbers without bothering to understand the techniques involved. The thirty computer diagrams included sequences of solutions with incremental changes in certain variables to understand their importance. Although print outs of solutions were not required, for this would take a great amount of time, the numerical answers were turned in.

The students' reaction to the MCTHIELE program was very enthusiastic. Many people commented on the ease of use of the program. Using the mouse as a pointing device for drawing the onscreen diagram was found to be very helpful.

It was also mentioned that the program was still quite easy to use when a mouse was unavailable. The extensive warning and help messages also made the program more convenient. Although a third of the class missed the program instructional session, no one had any problems understanding the program by themselves.

Another indirect effect of the program's introduction was seen through the term projects. After the students had been exposed to a well written and more general type of problem solving tool, the quality of projects over the previous year was dramatically improved. Much more thought was invested into the projects' logical flow, use of graphics, and "user friendliness". The projects also solved more general problems and several students designed programs that they hoped could be used for instruction in the future.

But, perhaps the most powerful statement of the MCTHIELE program's success as an educational tool was that when the homework sets were turned in, the students had not only constructed the thirty two diagrams that were assigned but they had used this tool to explore additional conditions and features. The returned homework sets had an average of five to ten additional problem solutions. The comfortable atmosphere, the fast solution of problems, and the ease of use had encouraged the students to explore additional areas and to ask new questions.

4.0 REFERENCES

[1] Holland, C.D. and Liapis, A.I. Computer Methods for Solving Dynamic Separation Problems, McGraw-Hill, (1983)

[2] Myers, A.L. and Seider, W.D. Introduction to Chemical Engineering and Computer Calculations, Prentice-Hall, (1976)

[3] Skjellum, A. "Software Design Issues and Proposals for Engineering Applications", Masters Report, California Institute of Technology, (1985)

[4] Natter, R.L. "Tools For Chemical Engineering Learning", 2nd University AEP Conference, San Diego, (1986)

[5] Edgar, T.F., Mah, R.S.H., and Reklaitis, G.V. "Use of Computers in Chemical Engineering Education", Chemical Engineering Progress, 81, 9, (1985)

[6] Calo, J.M. and Andres, R.P. "Use of Interactive Graphics Based Software for Teaching Chemical Engineering Process Principles", Computers and Chemical Engineering, 5, 197, (1981)

[7] Henley, E.J. and Seader, J.D. Equilibrium Stage Separation Operations in Chemical Engineering, John Wiley & Sons, (1981)

[8] King, C.J. Separation Processes, McGraw-Hill, (1971)

[9] McCabe, W.L. and Smith, J.C. Unit Operations of Chemical Engineering, McGraw-Hill, (1956)

[10] Hengstebeck, R.J. Distillation: Principles and Design Procedures, Reinhold Book, (1961)

[11] Walas, S.M. Phase Equilibria in Chemical Engineering, Butterworth, (1985)

[12] Prausnitz, J.M. Molecular Thermodynamics of Fluid Phase Equilibria, Prentice-Hall, (1969)

[13] Benedek, P. and Olti, F. Computer Aided Chemical Thermodynamics of Gases and Liquids, John Wiley & Sons, (1985)

[14] Tufte, E.R. The Visual Display of Quantitative Information, Graphic Press, (1983)

5.0 APPENDIX

LISTING OF SEPTECH, DEVELOPMENTAL VERSION 3.40

GLOBAL.H

SEPTECH.C

N_MIN.C

FEED.C

THERM.C

RESULTS2.C

LISTING OF MCTHIELE, VERSION 2.00

MC.H

COLOR.H

MC1.C

MC2.C

MC3A.C

MC3B.C

MC4.C

MC67.C

MCHALO.C

```
int h_key, l_key;

float tempF, fpress, p_column, p_bubble, p_dew, alpha_hk[21], feed[21];

float old_fp, p_boiler, t_boiler, p_cond;

float k[21], dist[21], bott[21], liq[21], vap[21];

int num_comp, string();

char *tx[] ={
     "nul",
     "1) Methane",
     "2) Methanol",
     "3) Ethylene",
     "4) Ethane",
     "5) Ethanol",
     "6) Acetone",
     "7) Propane",
     "8) Iso-Butane",
     "9) n-Butane",
     "10) Iso-Pentane",
     "11) n-Pentane",
     "12) Benzene",
     "13) Cyclo-Hexane",
     "14) n-Hexane",
     "15) Toluene",
     "16) n-Heptane",
     "17) Styrene",
     "18) Ethylbenzene",
     "19) n-Octane",
     "20) n-Nonane"
};

int compnts[21];

float database[21][6] ={
     {0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
     {673.1, 5.680, 52.0, 5.14135, 1742.64, 452.974},
     {1153.6, 14.510, 40.5, 7.5133, 6468.1, 396.2},
     {742.2, 5.801, 61.0, 5.27791, 2569.0, 433.92},
     {709.8, 6.050, 68.0, 5.3839, 2847.9, 434.9},
     {925.3, 12.915, 58.4, 7.4344, 6162.36, 359.4},
     {693.7, 9.566, 73.5, 6.244412, 5356.7, 397.53},
     {617.4, 6.4, 84.0, 5.3534, 3371.1, 414.5},
     {529.1, 6.730, 105.5, 5.6118, 3870.4, 409.95},
     {550.7, 6.634, 101.4, 5.7416, 4125.4, 409.518},
     {483.0, 7.020, 117.4, 5.5000, 4221.1, 387.3},
     {489.5, 7.020, 116.1, 5.854, 4598.3, 394.41},
     {714.2, 9.158, 89.4, 5.658, 5307.8, 379.5},
     {591.5, 8.193, 108.7, 5.473, 5030.3, 371.27},
     {440.0, 7.266, 131.6, 6.039, 5085.8, 382.8},
     {587.8, 8.914, 106.8, 5.944, 5836.3, 374.75},
     {396.9, 7.340, 147.5, 5.986, 5278.9, 359.5},
     {559.0, 9.211, 115.0, 6.071, 6329.6, 358.6},
     {540.0, 8.783, 123.1, 5.747, 5862.9, 349.8},
```

```
    {362.1, 7.551, 163.5, 6.414, 5947.5, 360.3},
    {331.0, 7.649, 179.6, 6.222, 6662.6, 331.0}
};


float N_min, R_min, N_actual, R_actual, N_strip;

int converge, superheat, subcool, comprs;
```

```
/*

     SEPTECH2.C

     Developemental Version 3.40

     Support libraries: Halo, Lc, Curses

*/


#include (global.h)
#include (stdio.h)

main()
{
     init_halo(4);           /* initialize HALO graphics   */
     _setworld();            /* set world coordinates      */

     comp();                 /* choose feed components     */

     feedcond();             /* specify feed conditions    */

     therm();                /* check thermal condition    */

     split();                /* specify desired split      */

     while (!converge)
     {
          bubble();     /* bubble point calculation of distillate */

          flashcol();   /* flash feed stream at column pressure   */

          min_stage();  /* calculate minimum theoretical stages   */

          TR_dist();    /* calc distribution at total reflux      */
     }

     putchar(7);         /* signal the end of main iterations      */

     min_rflx();         /* calc minimum reflux ratio              */

     actual();           /* calc actual # of stages & reflux ratio */

     feedstage();        /* calc optimal feed stage location       */

     boiler();           /* calc reboiler conditions               */

     if (comprs)
          compress();    /* notify if compressor is needed         */

     results2();         /* displays results on graphics screen    */

}
```

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

    This file contains the following routines:

        min_stage()   calculates the minimum number of stages
                      at an infinite reflux ratio

        TR_dist()  find distribution of components at conditions
                      of total reflux

        min_rflx()    calculate minimum reflux with an infinite
                      number of stages

        actual()   finds actual number of stages and the
                      corresponding reflux ratio

        feedstage()   finds optimal feed stage location

        bubble()   does bubble point calculation on distillate

        flashcol()    determines results of flashing feed at
                      column pressure

        k_value()  calculates k values from the composition
                      dependent modified Raoult's equation


    NOTE:    These routines MUST be linked with the "halo" file because
        psuedo-halo commands are used.

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



#include (extern.h)
#include (math.h)

/*
    This routine calculates the minimum number of stages required to
    effect the desired split with total reflux.

                                                */


min_stage()
{
    float     alpha_ij,        /* relative volatilities of i and j */
        var1, var2, var3;      /* generic variables          */


    alpha_ij = k[l_key] / k[h_key];

    var1 = dist[l_key] / dist[h_key];
```

```
    var2 = bott[h_key] / bott[l_key];

    N_min = log(var1 * var2) / log( alpha_ij);
}


/*
    The following routine finds the distribution of components at
    conditions of total reflux
                                          */

TR_dist()
{
    float    var1,          /* generic variable      */
        hk_ratio,      /* comp to heavy key ratio    */
        old_bott; /* old bottoms flow rate */

    int  i;        /* counting variable          */


    k_value(feed, tempF, 1); /* computes k values     */


    old_bott = bott[0];

    hk_ratio = dist[h_key] / bott[h_key];

    /* For lighter components, calculate bottoms flows and use material
       balance to find distillate flow.                        */

    for (i=1;i<=l_key;++i)
    {
        if (feed[i])
        {

        bott[i] = feed[i]/(1.0+hk_ratio* pow(alpha_hk[i], N_min));
        dist[i] = feed[i] - bott[i];
        }
    }

    /* For heavier components, calculate distillate flows and use
       material balance to find bottoms flow.                  */

    for (i=h_key;i<=20;++i)
    {
        if (feed[i])
        {

        var1 = 1.0 + hk_ratio* pow( alpha_hk[i], N_min);
        dist[i] = feed[i]*hk_ratio* pow( alpha_hk[i], N_min) / var1;

        bott[i] = feed[i] - dist[i];
        }
    }
```

```
     bott[0] = 0.0;
     dist[0] = 0.0;

     /* store total flow rates in the zero elements of the arrays     */

     for (i=1;i<=20;++i)
     {
          if (feed[i])
          {
               bott[0] = bott[0] + bott[i];
               dist[0] = dist[0] + dist[i];
          }
     }

     /* test for convergence of non-key component splits         */

     converge = 0;

     if (( fabs(old_bott - bott[0]) / bott[0]) < .0001)
          converge = 1;

}


/*
     The minimum reflux ratio for an infinite number of stages is
     found by this routine.
                                             */

min_rflx()
{

     float     L_min,         /* minimum liquid phase returned to column */
          var1, var2;    /* generic variables                  */


     var1 = alpha_hk[l_key] * dist[h_key] / liq[h_key];

     var2 = liq[0] * ((dist[l_key] / liq[l_key]) - var1);

     L_min = var2 / (alpha_hk[l_key] - 1.0);

     R_min = L_min / dist[0]; /* minimum reflux ratio  */

}




/*
     This routine finds the actual number of stages necessary at the
     appropriate reflux ratio.
                                             */

#define   GILL_COR  1.30 /* Gilliland Correlation number         */
```

```
actual()
{

     float     var1, var2, var3,   /* generic variables          */
          _x;            /* dimensionless parameter    */


     R_actual = R_min * GILL_COR;  /* actual reflux ratio        */

     _x = (R_actual - R_min)/(R_actual + 1.0);

     var1 = (_x - 1.0) / pow(_x, 0.5);

     var2 = (1.0 + 54.4 * _x) / (11.0 + 117.2 * _x);

     var3 = 1.0 - exp(var1 * var2);

     N_actual = (var3 + N_min) / (1.0 - var3);    /* actual stages */


}


/*
     The optimal feed stage is found in this routine.
                                             */

feedstage()
{
     float var1, var2, var3, var4;      /* generic variables     */


     var1 = bott[l_key]/dist[h_key];

     var1 = pow(var1,2.0);

     var2 = feed[h_key] / feed[l_key];

     var3 = var1 * var2 * bott[0] / dist[0];

     var4 = pow(var3, 0.206);

     N_strip = N_actual / (1.0 + var4);
                    /* number of stripping section stages   */

}


/*
     The bubble point of the distillate stream is found in the
     following routine.
                                             */


#define        R    1.987     /*  [=] cal/(gmol*K)          */
                    /* universal gas constant     */
```

```
#define        RFLX_T    120.0     /*  120.0 degree F           */
                    /* rule of thumb temp for cooling water use */

/*
    P bub = SUM ( gamma * z * p_sat )
*/


bubble()
{
     float     p_sat,              /* saturation pressure        */
          del_bar,       /* averaged solubility parameter */
          gamma,              /* liquid state fugacity coeff.  */
          var1, var2, var3,   /* generic variables          */
          tempK;              /* temp in degress K          */

     int  i;             /* counting variable     */

     working();

     var1 = 0.0;
     var2 = 0.0;
     var3 = 0.0;

     for (i=1;i(=20;++i) /* calculate del bar     */
     {
          if (dist[i])
          {
               var1 = dist[i]/dist[0] * database[i][2];
               var2 = var2 + var1 * database[i][1];
               var3 = var3 + var1;
          }
     }

     del_bar = var2 / var3;

     p_cond = 0.0;

     for (i=1;i(=20;++i) /* solve equation for bubble temperature */
     {
          if (dist[i])
          {
               tempK = 273.15 + ((RFLX_T-32.0)*5.0/9.0);

               var1 = (database[i][2]/(R*tempK));
               var2 = (database[i][1] - del_bar);
               var3 = var2 * var2;
               gamma = exp(var1 * var3);

               var1 = database[i][4]/(RFLX_T + database[i][5]);
                    /* 120.0 degree F */
               var2 = database[i][3] - var1;
               p_sat = database[i][0] * exp(var2);  /* [=] psia */

               p_cond = p_cond +(gamma*dist[i]*p_sat/dist[0]);
```

```
        }
    }

    p_column = p_cond + 7.5;

/*  This added 7.5, allows for a pressure drop of 5 psia through
    the condenser and another 5 psia for the column.                */


}



double forw_der(), derfxn(), fxn();
                        /* variable types of procedure returns  */

#define  INTRVL   0.0001    /* delta for the numerical derivitive */

#define  TOL      0.0001   /* the error tolerence between sucessive
                     iterations of psi              */


/* The next two routines calculate a numerical derivitive of an arbitrary
   function.  The first routine calculates a central derivitve with the
   delta defined above as INTRVL.  If the central derivitive is equal to
   zero, the forward derivitive is found.  This eliminates the possibility
   of division by zero if the Newton method is used.  The delta for the
   forward derivitive is also INTRVL.                              */


double derfxn(f,y)       /* the central derivitive  */
double (*f)();
double y;
{
    double  y_up, y_down, der;
    y_up = y + INTRVL;
    y_down = y - INTRVL;
    der = (((*f)(y_up) - (*f)(y_down))/(2.0*INTRVL));

    /* If the central derivitve is equal to zero, the forward
       derivitive is calulated.                         */

    if ( der == 0.0 )
    {
        der = forw_der(fxn,y);
    }
    return(der);
}

double forw_der(f,y)          /* the forward derivitive */
double (*f)();
double y;
{
    double y_up, der;
    y_up = y + INTRVL;
    der = (((*f)(y_up)-(*f)(y))/INTRVL);
```

```
    return(der);
}



/* This is the Rachford-Rice flash function.  The evaluation involves
   taking the sum of a finite number of terms.  The number of terms is
   equal to the number of components.                          */

double fxn(y)
double y;
{
    double summ;        /* the summing varible  */
    int i;              /* a counting varible   */


    summ = 0;           /* reset sum to zero    */
    for (i=1; i(=20; i++)
    {
        if (feed[i])
        summ = summ+(((feed[i]/feed[0])*(1-k[i]))/(1+y*(k[i]-1)));
    }
    return(summ);
}


/*
    The results of the flash vaporization of the feed stream to
    the operating conditions of the column.
                                          */

flashcol()
{

    float     old_liq,  /* storage of the previous iteration    */
        p_sat,          /* saturation pressure                  */
        psi,      /* root of the Rachford-Rice equation   */
        old_psi,  /* storage of the previous iteration    */
        var1, var2;     /* generic variables                    */

    int  i;         /* a counting variable              */


    comprs = 0;

    fpress = old_fp;

    if (fpress (= p_column)
    {
        comprs = 1;

        old_fp = fpress;
        fpress = p_column + 5.0;
    }
```

```
/* first guess of split to start iteration   */

for (i=1;i(=20;++i)
{
     if (feed[i])
          liq[i] = feed[i]/2;
}
psi = 0.0;


do     /* performs iterations on the total liquid flow rate */
{

     k_value(liq,tempF,0);

     old_liq = liq[0];
     /* store previous iteration of total liquid flow */

     do   /* performs iterations on psi      */
     {

          old_psi = psi; /* saves last iteration  */

          psi = psi - (fxn(psi)/derfxn(fxn,psi));
          /* the Newton's method equation */


     } while ( fabs(old_psi - psi) ) TOL);

     liq[0] = feed[0] * (1.0 - psi);


     for (i=1; i(=20; ++i)         /* calc flash split */
     {
          if (feed[i])
             liq[i]=liq[0]*((feed[i]/feed[0])/(1+psi*(k[i]-1)));
     }

} while (fabs((liq[0] - old_liq)/liq[0]) ) 0.001);
/* until liquid flow rate iterations are within 0.1 %  */


vap[0] = 0.0;

/* find vapor flows */
for (i=1;i(=20;++i)
{
     if (feed[i])
     {
          vap[i] = feed[i] - liq[i];
          vap[0] = vap[0] + vap[i];
     }


}
```

```
}



/*
     The equilibrium k values are found in this routine using the
     composition dependent modified Raoult's equation.
                                                    */


k_value(flow,temp,alpha)
float     flow[21], /* a dummy array to hold stream condition values   */
     temp;          /* stream temperature                              */
int  alpha;         /* on/off indicator for relative volitility option */
{

     float     tempK,              /* temperature in degrees K   */
          p_sat,              /* saturation pressure        */
          del_bar,       /* averaged solubility para   */
          gamma,              /* liquid fugacity coefficient     */
          var1, var2, var3;   /* generic variables          */

     int  i;             /* counting variable          */

     var1 = 0.0;
     var2 = 0.0;
     var3 = 0.0;

     /* calculate del bar     */
     for (i=1;i<=20;++i)
     {

          if (flow[i])
          {
               var1 = flow[i]/flow[0] * database[i][2];
               var2 = var2 + var1 * database[i][1];
               var3 = var3 + var1;
          }
     }

     del_bar = var2 / var3;


     tempK = 273.15 + (temp - 32.0)*5.0/9.0;

     /* find fugacity coefficient and saturation pressure to find
        an equilibrium k value                          */

     for (i=1;i<=20;++i)
     {
          if (flow[i])
          {
               var1 = (database[i][2]/(R*tempK));
               var2 = (database[i][1] - del_bar);
```

```
                var3 = var2 * var2;
                gamma = exp(var1 * var3);

                var1 = database[i][4]/(temp + database[i][5]);
                var2 = database[i][3] - var1;
                p_sat = database[i][0] * exp(var2);  /* [=] psia */

                k[i] = gamma * p_sat / p_column;
            }
        }

        if (alpha)      /* if relative volitility option ON     */
        {
            for (i=1;i(=20;++i)
            {
                if (flow[i])
                    alpha_hk[i] = k[i]/k[h_key];

            }
        }

}
```

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

     This file contains the following routines:

          comp()         allows choice of twenty components in database

          feedcond()     specification of feed composition, temperature
                         and pressure

          split()        specification of primary split and percents of
                         key components in distillate

          strng()        a routine for displaying keyboard input on
                         color monitor
                         ( original code from Anthony Sjkellum )

          _getch()       an "intelligent" getch command to support the
                         comp() and split() routines


     NOTE:     These routines MUST be linked with the "halo" file because
               psuedo-halo commands are used.

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




#include (math.h)
#include (extern.h)
#include (curses.h)


/*
     This routine allows specification of the feed stream compositions,
     temperature, and pressure
                                             */


feedcond()
{

     char lambda[12];    /* dummy character string          */

     int  i, j,          /* counting varibles               */
          numcomp;  /* number of components           */

     float tcury;        /* text cursor y value             */

     _setcolor(0);
     clr();              /* clears screen to black          */

     diagram(0.0,0.0,200.0,1000.0,6,0,0,1);
          /* (xlow,ylow,xhigh,yhigh,color,trays,truefeed,diafeed) */
```

```
_setext(2,1,0,1);              /* (height,width,path,mode)   */
_inittcur(2,4,11);             /* (height, width, color)     */
_movtcurabs(10.0, 10.0);
_settextclr(0,2);              /* (foreground,background)    */
text(" Type in numbers  ;  'RETURN' to enter ");
                         /* command line text     */
_setext(2,1,0,0);
_movtcurabs(250.0,950.0);
_settextclr(4,0);
text("SPECIFY FEED COMPOSITION");  /* title of page    */

_setext(1,1,0,0);
_movtcurabs(300.0, 900.0);
_settextclr(4,0);
text("[=] LBMOLES / HR");

_setext(2,1,0,0);
_settextclr(11,0);
numcomp = 0;
for (i=1;i(=20;++i)     /* displays names of chosen components */
{
     if (compnts[i] == 1)
     {
          ++numcomp;     /* counts number of components     */

          _movtcurabs(200.0, 800.0-(numcomp * 50.0));
          text(tx[i]);
     }
}
num_comp = numcomp;

_movtcurabs(200.0, 800.0-(numcomp + 2) * 50.0);
text("Temp [=] deg F");
_movtcurabs(200.0, 800.0-(numcomp + 3) * 50.0);
text("Pressure [=] psia");


for (i=1;i(=20;++i)      /* accepts input of flow rates      */
{                  /* flow rates stored in feed[21] */
     if (compnts[i])
     {

          --numcomp;
          tcury = 800.0-(num_comp-numcomp)*50.0;
          _movtcurabs(200.0, tcury);
          _settextclr(16,0);
          text(tx[i]);

          strng(650.0, tcury, lambda, 1, 10);

          feed[i] = atof(lambda);

          _movtcurabs(200.0, tcury);
          _settextclr(11,0);
```

```
            text(tx[i]);
        }
    }

    _inittcur(1,1,0);

    for (i=1;i(=20;++i) /* sets feed[0] equal to total feed rate */
    {
        feed[0] = feed[0] + feed[i];
    }

    /* accepts input of feed temperature    */

    _settextclr(16,0);
    strng(700.0, 800.0-(num_comp + 2) * 50.0, lambda, 1, 10);
    sscanf(lambda, "%f", &tempF);

    /* accepts input of feed pressure  */

    strng(700.0, 800.0-(num_comp + 3) * 50.0, lambda, 1, 10);
    sscanf(lambda, "%f", &fpress);

    old_fp = fpress;

    if (tempF (= 130.0)
        preheat();


}



/*
    This routine allows selection of components from list in database.
                                        */


comp()
{

    int  old_num, new_num,   /* variables for tracking cursor */
        hue;          /* color indicator       */

    char c;           /* a dummy character variable      */


    diagram(0.0,0.0,200.0,1000.0,5,0,0,1);
        /* (xlow,ylow,xhigh,yhigh,color,trays,truefeed,diafeed) */


    _setext(2,1,0,1);           /* (height,width,path,mode)   */
    _inittcur(1,1,0);           /* (height, width, color)     */
    _movtcurabs(10.0, 10.0);
    _settextclr(0,2);           /* (foreground,background)    */
    text("CUR ARROWS-move, RETURN-enter, ESC-quit");
                        /* command line text          */
```

```
_setext(2,1,0,0);
_movtcurabs(300.0,950.0);
_settextclr(4,0);
text(" SELECT COMPONENTS ");  /* title of page        */

/* listing of all twenty components in database       */

_settextclr(11,0);
_movtcurabs(200.0, 800.0);
text(tx[1]);
_movtcurabs(200.0, 750.0);
text(tx[2]);
_movtcurabs(200.0, 700.0);
text(tx[3]);
_movtcurabs(200.0, 650.0);
text(tx[4]);
_movtcurabs(200.0, 600.0);
text(tx[5]);
_movtcurabs(200.0, 550.0);
text(tx[6]);
_movtcurabs(200.0, 500.0);
text(tx[7]);
_movtcurabs(200.0, 450.0);
text(tx[8]);
_movtcurabs(200.0, 400.0);
text(tx[9]);
_movtcurabs(200.0, 350.0);
text(tx[10]);
_movtcurabs(600.0, 800.0);
text(tx[11]);
_movtcurabs(600.0, 750.0);
text(tx[12]);
_movtcurabs(600.0, 700.0);
text(tx[13]);
_movtcurabs(600.0, 650.0);
text(tx[14]);
_movtcurabs(600.0, 600.0);
text(tx[15]);
_movtcurabs(600.0, 550.0);
text(tx[16]);
_movtcurabs(600.0, 500.0);
text(tx[17]);
_movtcurabs(600.0, 450.0);
text(tx[18]);
_movtcurabs(600.0, 400.0);
text(tx[19]);
_movtcurabs(600.0, 350.0);
text(tx[20]);

/* redraw first component in white */

_movtcurabs(200.0,800.0);
_settextclr(16,0);
text(tx[1]);
old_num = 1;
```

```
noecho(); /* stops keyboard input echo on monochrome (curses) */

while (1)
{
     int chr;  /* control varible */

     chr = _getch();

     if (chr == 27)      /* if ESC key  */
          break;

     if (chr == 333)          /* if cursor arrow right */
     {
          if (old_num (= 10)
          {
               if (compnts[old_num] == 1)
                    hue = 1;
               else
                    hue = 11;

               _movtcurabs(200.0, 850.0-old_num*50.0);
               _settextclr(hue,0);
               text(tx[old_num]);

               old_num = old_num + 10;
               _movtcurabs(600.0, 850-(old_num-10)*50.0);
               _settextclr(16,0);
               text(tx[old_num]);
          }
     }

     if (chr == 331)          /* if cursor arrow left       */
     {
          if (old_num ) 10)
          {
               if (compnts[old_num] == 1)
                    hue = 1;
               else
                    hue = 11;

               _movtcurabs(600.0, 850.0-(old_num-10)*50.0);
               _settextclr(hue,0);
               text(tx[old_num]);

               old_num = old_num - 10;
               _movtcurabs(200.0, 850.0 - old_num*50.0);
               _settextclr(16,0);
               text(tx[old_num]);
          }
     }


     if (chr == 328)          /* if cursor arrow up         */
     {
          if ((old_num != 1) && (old_num != 11))
          {
```

```
            if (compnts[old_num] == 1)
                 hue = 1;
            else
                 hue = 11;

            if (old_num < 11)
               _movtcurabs(200.0,850.0-old_num*50.0);
            else
               _movtcurabs(600.0,850.0-(old_num-10)*50.0);
            _settextclr(hue,0);
            text(tx[old_num]);

            old_num = old_num - 1;
            if (old_num < 11)
               _movtcurabs(200.0,850.0-old_num*50.0);
            else
               _movtcurabs(600.0,850.0-(old_num-10)*50.0);
            _settextclr(16,0);
            text(tx[old_num]);
       }
  }


  if (chr == 336)          /* if cursor arrow down       */
  {
       if ((old_num != 10) && (old_num != 20))
       {
            if (compnts[old_num] == 1)
                 hue = 1;
            else
                 hue = 11;

            if (old_num < 11)
               _movtcurabs(200.0,850.0-old_num*50.0);
            else
               _movtcurabs(600.0,850.0-(old_num-10)*50.0);
            _settextclr(hue,0);
            text(tx[old_num]);

            old_num = old_num + 1;
            if (old_num < 11)
               _movtcurabs(200.0,850.0-old_num*50.0);
            else
               _movtcurabs(600.0,850.0-(old_num-10)*50.0);
            _settextclr(16,0);
            text(tx[old_num]);
       }
  }

  if (chr == 13)      /* if RETURN key          */
  {
        /* toggle compnts[] to specify as chosen or unchosen */

       if (compnts[old_num] == 0)
            compnts[old_num] = 1;
       else
```

```
                compnts[old_num] = 0;

            if (compnts[old_num] == 1)
                hue = 1;
            else
                hue = 11;

            if (old_num < 11)
               _moveabs(200.0,850.0-old_num*50.0);
            else
               _moveabs(600.0,850.0-(old_num-10)*50.0);
            _settextclr(hue,0);
            text(tx[old_num]);
        }

    }


    echo();         /* return keyboard input echo on monochrome (curses) */

}


/*
    This is an "intelligent" getch() command that adds 256 to the ASCII
    codes of control characters
                                                    */

_getch()
{
    int chr = getch();

    if (chr != 0)
        return(chr);

    chr = getch();
    return(256+chr);
}


/*
    The split() routine allows specification of primary split and
    percent of key components in distillate.  A preliminary guess
    is made at the split of non-key components.
                                                */


split()
{

    int  i, j,              /* counting variables        */
         compnums[21],      /* array of component numbers */
         old_num, numcomp,  /* tracking variables        */
         startcom,
         first, second;
```

```
int  place;              /* location on screen        */

float     var1,          /* generic variable               */
     tcury,         /* text cursor y value            */
     per;      /* percent of comp in distillate   */

char      lambda[12];    /* dummy character variable       */

_setcolor(0);
clr();

diagram(0.0,0.0,200.0,1000.0,13,0,0,1);
     /* (xlow,ylow,xhigh,yhigh,color,trays,truefeed,diafeed) */


_setext(2,1,0,1);     /* (height,width,path,mode)  */
_inittcur(1,1,0);     /* (height, width, color)         */
_movtcurabs(10.0, 10.0);
_settextclr(0,2);     /* (foreground,background)   */
text("  CUR ARROWS - move,  RETURN - enter   ");

_setext(2,1,0,0);
_movtcurabs(300.0,950.0);
_settextclr(4,0);
text("INDICATE PRIMARY SPLIT");

_settextclr(11,0);

numcomp = 0;
for (i=1;i(=20;++i) /* display chosen components on screen  */
{
     if (compnts[i])
     {
          ++numcomp;
          compnums[numcomp] = i;
          _movtcurabs(250.0, 800.0-(numcomp * 50.0));
          text(tx[i]);
     }
}
num_comp = numcomp;


noecho();


place = 1;
while (1)
{
     int  chr; /* control variable      */

     _setcolor(16);

     /* draws split cursor on screen localion "place"  */

     var1 = (place * 50.0);
```

```
        _movabs(240.0, 850.0- var1);
        _lnabs(200.0, 850.0- var1);
        _lnabs(200.0, 750.0- var1);
        _lnabs(240.0, 750.0- var1);
        _movabs(210.0, 850.0- var1);
        _lnabs(210.0, 750.0- var1);


        chr = _getch();

        if (chr == 336)          /* down cursor arrow     */
        {
             if (place ( (numcomp - 1))
                  ++place;
        }

        if (chr == 328)               /* up cursor arrow  */
        {
             if (place ) 1)
                  --place;
        }

        if (chr == 13)           /* RETURN key       */
        {
             l_key = compnums[place];
             h_key = compnums[(place + 1)];

             break;
        }

        /* undraw split cursor at old location "place"        */

        _setcolor(0);
        _movabs(240.0, 850.0- var1);
        _lnabs(200.0, 850.0- var1);
        _lnabs(200.0, 750.0- var1);
        _lnabs(240.0, 750.0- var1);
        _movabs(210.0, 850.0- var1);
        _lnabs(210.0, 750.0- var1);


   }

   echo();

   compnts[l_key] = 2;
   compnts[h_key] = 3;


   /* highlight key components   */

   _settextclr(1,0);

   _movtcurabs(250.0, 800.0 - (place * 50.0));
   text(tx[compnums[place]]);
```

```
_movtcurabs(250.0, 800.0 - (place + 1) * 50.0);
text(tx[compnums[(place + 1)]]);


/* undraw old command line and title          */

_setext(2,1,0,1);              /* (height,width,path,mode)    */
_movtcurabs(10.0, 10.0);
_settextclr(0,0);              /* (foreground,background)     */
text("  CUR ARROWS - move,  RETURN - enter   ");

_setext(2,1,0,0);
_movtcurabs(300.0,950.0);
_settextclr(0,0);
text("INDICATE PRIMARY SPLIT");


/* draw new command line and title       */

_setext(2,1,0,0);
_movtcurabs(250.0,950.0);
_settextclr(4,0);
text("ENTER PERCENT OF COMPONENT");
_movtcurabs(400.0, 900.0);
text("IN DISTILLATE");

_setext(2,1,0,1);              /* (height,width,path,mode)    */
_movtcurabs(10.0, 10.0);
_settextclr(0,2);              /* (foreground,background)     */
text("    Type in numbers,  RETURN - enter   ");


tcury = 800.0 - place * 50.0;

/* input percent of light key in distillate  */

_settextclr(16,0);
strng(700.0, tcury, lambda, 1, 10);
sscanf(lambda, "%f", &per);

dist[l_key] = feed[l_key] * per / 100.0;



tcury = 800.0 - (place + 1) * 50.0;

/* input percent of heavy key in distillate  */

strng(700.0, tcury, lambda, 1, 10);
sscanf(lambda, "%f", &per);

dist[h_key] = feed[h_key] * per/100.0;


_inittcur(1,1,0);
```

```
    /* make preliminary estimates of non-key component splits   */

    for (i=1;i(l_key;++i)
    {
        dist[i] = feed[i];
    }

    for (i=1;i(=20;++i)
    {
        bott[i] = feed[i] - dist[i];
    }

    /* let total flow rates equal zero elements of arrays       */

    for (i=1;i(=20;++i)
    {
        dist[0] = dist[0] + dist[i];
    }

    for (i=1;i(=20;++i)
    {
        bott[0] = bott[0] + bott[i];
    }



}


/*
    The next two routines allow keyboard input to be displayed on
    the color screen.  Original code by A. Skjellum.
                                                */


strng(tcurx,tcury,string,imin,imax)     /* sets text cursor position */
double tcurx,tcury;           /* then goes to _strng()     */
char *string;
int imin,imax;
{
    _setcolor(0);
    _inittcur(2,4,11);
    _setext(2,1,0,0);
    _settextclr(7,0);
    _movtcurabs(tcurx,tcury);
    return(_strng(string,imin,imax));
}

_strng(string,imin,imax) /* displays text on color screen and  */
char *string;            /* allows backspacing during input    */
int imin,imax;
{
    float o_curx,o_cury;
    float curx,cury;
    float delta;
    int dflag = 0;
```

```
int y,x;
int i = 0;
int done = 0;

inqtcur(&o_curx,&o_cury);
curx = o_curx;
cury = o_cury;
delta = 0.0;

while(!done)
{
     int chr;
     chr = getch(); /* get a character */
     switch(chr)
     {
          case '\b':     /* backspace     */
          case 0x07f:
               if(i)  /* decrement count */
               {
                    i--;
                    o_curx = curx;
                    curx -= delta;
                    _movtcurabs(curx,cury);
                    text(" ");
                    _movtcurabs(curx,cury);
               }
               break;

          case '\n':     /* RETURN or carriage feed    */
          case '\r':
              if(i )= imin) done = 1;
               break;

          default:  /* text to be displayed       */
               if(i ( imax)
               {
                    char str[2];
                    if(chr ( 32) break;
                    string[i++] = chr;
                    str[0] = chr;
                    str[1] = '\0';
                    text(str);
                    o_curx = curx;
                    o_cury = cury;
                    inqtcur(&curx,&cury);
                    if(dflag == 0)
                    {
                         delta = curx - o_curx;
                         dflag = 1;
                    }
               }
               break;
     }

} /* end while(!done) */
```

```
    string[i] = '\0';   /* eos */

    return(i);
}
```

```
#include (math.h)
#include (extern.h)

#define        R    1.987     /*  [=] cal/(gmol*K)  */

/*
    P bub = SUM ( gamma * z * p_sat )

    P dew = SUM ( gamma * p_sat / z )
*/
/* sets 'superheat' and 'subcool' flags */

therm()        /* temperature selected */
{
     float     invp_dew, p_sat, del_bar, gamma, var1, var2, var3, tempK;

     int  i;

     var1 = 0.0;
     var2 = 0.0;
     var3 = 0.0;

     superheat = 0;
     subcool = 0;

     p_bubble = p_dew = invp_dew = 0.0;


     for (i = 1; i (= 20; ++i)
     {
          if (feed[i])
          {
               var1 = feed[i]/feed[0] * database[i][2];
               var2 = var2 + var1 * database[i][1];
               var3 = var3 + var1;
          }
     }

     del_bar = var2 / var3;

     for (i = 1; i (= 20; ++i)
     {
          if (feed[i])
          {
               tempK = 273.15 + (( tempF -32.0)*5.0/9.0);
               var1 = (database[i][2]/(R*tempK));
               var2 = (database[i][1] - del_bar);
               var3 = var2 * var2;
               gamma = exp(var1 * var3);

               var1 = database[i][4]/(tempF + database[i][5]);
               var2 = database[i][3] - var1;
               p_sat = database[i][0] * exp(var2);  /* [=] psia */

               p_bubble = p_bubble + (gamma*feed[i]*p_sat/feed[0]);
```

```
            invp_dew = invp_dew + (feed[i]/(feed[0]*gamma*p_sat));
        }
    }

    p_dew = 1.0 / invp_dew;

    superheat = 0;
    subcool = 0;

    if (fpress < p_dew)
        superheat = 1;

    if (fpress > p_bubble)
        subcool = 1;

/**********************
    if ((superheat) || (subcool))
    {
        compress();         /* notify if compressor used  */
    }
**********************/

}


preheat()
{
    char     tempstr[100];


    _setcolor(0);
    clr();                  /* clears screen to black     */

    diagram(0.0,0.0,200.0,1000.0,5,0,0,1);
        /* (xlow,ylow,xhigh,yhigh,color,trays,truefeed,diafeed) */

    _setext(2,1,0,1);       /* (height,width,path,mode)   */
    _inittcur(2,4,11);      /* (height, width, color)     */
    _movtcurabs(10.0, 10.0);
    _settextclr(0,2);       /* (foreground, background)   */
    text("   to continue, hit the 'RETURN' key   ");

    _setext(2,1,0,0);
    _movtcurabs(325.0, 950.0);
    _settextclr(4,0);
    text("FEED PRE-HEATER NEEDED");

    _setext(2,1,0,0);
    _settextclr(11,0);

    _movtcurabs(200.0,700.0);
    text("To allow the use of water as the");
    _movtcurabs(200.0,650.0);
    text("coolant in the condenser, the ");
    _movtcurabs(200.0,600.0);
    text("feed must enter the column at a");
```

```
    _movtcurabs(200.0,550.0);
    text("minimum temperature of 130 dg F.");
    _movtcurabs(200.0,450.0);
    text("A feed pre-heater must be added");
    _movtcurabs(200.0,400.0);
    text("to increase the temperature .");
    _movtcurabs(200.0,300.0);
    text("FEED CONDITION AFTER PRE-HEATER");
    _movtcurabs(200.0,250.0);
    text("Temperature [=] deg F");
    _movtcurabs(800.0,250.0);
    text("130.00");
    _movtcurabs(200.0,200.0);
    text("Pressure [=] psia");
    _movtcurabs(800.0,200.0);
    sprintf(tempstr, "%.2g", fpress);
    text(tempstr);

    tempF = 130.0;

    while(1)
    {
        int chr =_getch();

        if (chr == 13)      /* 'RETURN' key              */
            break;
    }

}



char *words[] =
{
    "NULL",
    "COMPRESSOR NEEDED",
    " EXPANDER NEEDED"
};

compress()
{
    char     tempstr[100];

    _setcolor(0);
    clr();                  /* clears screen to black    */

    diagram(0.0,0.0,200.0,1000.0,5,0,0,1);
        /* (xlow,ylow,xhigh,yhigh,color,trays,truefeed,diafeed) */

    _setext(2,1,0,1);       /* (height,width,path,mode)  */
    _inittcur(2,4,11);      /* (height, width, color)    */
    _movtcurabs(10.0, 10.0);
    _settextclr(0,2);       /* (foreground, background)  */
    text("   to continue, hit the 'RETURN' key   ");

    _setext(2,1,0,0);
```

```
     _movtcurabs(375.0, 950.0);
     _settextclr(4,0);
/*******
     if (superheat)
     {
          text(words[1]);

          _setext(2,1,0,0);
          _settextclr(11,0);

          _movtcurabs(200.0,750.0);
          text("At the specified temperature and");
          _movtcurabs(200.0,700.0);
          text("pressure, the feed is below its  ");
          _movtcurabs(200.0,650.0);
          text("dew point pressure.");
          _movtcurabs(200.0,500.0);
          text("A compressor will be added to ");
          _movtcurabs(200.0,450.0);
          text("raise the pressure of the feed.");
          _movtcurabs(200.0,350.0);
          text("CONDITIONS AFTER COMPRESSOR");
          _movtcurabs(200.0,300.0);
          text("Temperature [=] deg F");
          _movtcurabs(800.0,300.0);
          sprintf(tempstr, "%.2g", tempF);
          text(tempstr);
          _movtcurabs(200.0,250.0);
          text("Pressure [=] psia");
          _movtcurabs(800.0,250.0);
          sprintf(tempstr, "%.2g", p_dew);
          text(tempstr);

          fpress = p_dew;

     }
     if (subcool)
     {
          text(words[2]);

          _setext(2,1,0,0);
          _settextclr(11,0);

          _movtcurabs(200.0,750.0);
          text("At the specified temperature and");
          _movtcurabs(200.0,700.0);
          text("pressure, the feed is above its ");
          _movtcurabs(200.0,650.0);
          text("bubble point pressure.");
          _movtcurabs(200.0,500.0);
          text("An expander will be added to ");
          _movtcurabs(200.0,450.0);
          text("lower the pressure of the feed.");
          _movtcurabs(200.0,350.0);
          text("CONDITIONS AFTER EXPANDER");
          _movtcurabs(200.0,300.0);
```

```
        text("Temperature [=] deg F");
        _movtcurabs(800.0,300.0);
        sprintf(tempstr, "%.2g", tempF);
        text(tempstr);
        _movtcurabs(200.0,250.0);
        text("Pressure [=] psia");
        _movtcurabs(800.0,250.0);
        sprintf(tempstr, "%.2g", p_bubble);
        text(tempstr);

        fpress = p_bubble;

    }
*********/
    if (compress)
    {
        text(words[1]);

        _setext(2,1,0,0);
        _settextclr(11,0);

        _movtcurabs(200.0,750.0);
        text("The specified feed pressure is   ");
        _movtcurabs(200.0,700.0);
        text("below the pressure calculated");
        _movtcurabs(200.0,650.0);
        text("for column operation.");
        _movtcurabs(200.0,500.0);
        text("A compressor will be added to ");
        _movtcurabs(200.0,450.0);
        text("raise the pressure of the feed.");
        _movtcurabs(200.0,350.0);
        text("CONDITIONS AFTER COMPRESSOR");
        _movtcurabs(200.0,300.0);
        text("Temperature [=] deg F");
        _movtcurabs(800.0,300.0);
        sprintf(tempstr, "%.2g", tempF);
        text(tempstr);
        _movtcurabs(200.0,250.0);
        text("Pressure [=] psia");
        _movtcurabs(800.0,250.0);
        sprintf(tempstr, "%.2g", fpress);
        text(tempstr);

    }

    while(1)
    {
        int chr =_getch();

        if (chr == 13)      /* 'RETURN' key          */
            break;
    }

}
```

```
/*
    P bub = SUM ( gamma * z * p_sat )
*/

boiler()
{
     float     p_sat, del_bar, gamma, var1, var2, var3, tempK;

     float     _tempF, times;

     int  i, past;

     past = 0;

     times = 1.0;

     var1 = 0.0;
     var2 = 0.0;
     var3 = 0.0;

     p_boiler = p_column + 2.5;

     _tempF = tempF;

     do
     {

          for (i = 1; i (= 20; ++i)
          {
               if (bott[i])
               {
                    var1 = bott[i]/bott[0] * database[i][2];
                    var2 = var2 + var1 * database[i][1];
                    var3 = var3 + var1;
               }
          }

          del_bar = var2 / var3;

          p_bubble = 0.0;

          for (i = 1; i (= 20; ++i)
          {
               if (bott[i])
               {
                    tempK = 273.15 + (( _tempF -32.0)*5.0/9.0);
                    var1 = (database[i][2]/(R*tempK));
                    var2 = (database[i][1] - del_bar);
                    var3 = var2 * var2;
                    gamma = exp(var1 * var3);

                    var1 = database[i][4]/(_tempF+database[i][5]);
                    var2 = database[i][3] - var1;
                    p_sat = database[i][0] * exp(var2);  /* psi */
```

```
                    p_bubble=p_bubble+(gamma*bott[i]*p_sat/bott[0]);

               }
          }

          if (p_bubble (= p_boiler)
          {
               if (!past)
                    _tempF = _tempF + 50.0;

               if (past)
               {
                    times = times + 1.0;
                    _tempF = _tempF + 50.0/(times);
               }
          }

          if (p_bubble ) p_boiler)
          {
               times = times + 1.0;
               _tempF = _tempF - 50.0/(times);
               past = 1;
          }

     } while (fabs(p_bubble - p_boiler) ) 1.0);

     t_boiler = _tempF;

}
```

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

    This file contains the routine:

        results2()

    In this routine, the results of the SEPTEC simulation program
    are displayed on both the color and monochrome screens.


    NOTE:   This routine MUST be linked with the "halo" file because
        psuedo-halo commands are used.

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



#include (extern.h)


results2()
{
    char tempstr[100];

    int  enough,
         toggle,
         trays,             /* number of actual trays     */
         truefeed,          /* optimal feed stage location     */
         i, j;              /* counting variables         */

    float    var1, var2, var3;   /* generic variables          */



    trays = N_actual - 1.0;
    truefeed = N_strip;
    var1 = 0.25 + 0.5/trays/2.0 + 0.5/trays*(truefeed-1);

    toggle = 1;
    enough = 0;

    do
    {
        if (toggle)
        {
            _setcolor(0);
            clr();

            diagram(0.0,0.0,200.0,1000.0,1,trays,truefeed,0);
        /* (xlow,ylow,xhigh,yhigh,color,trays,truefeed,diafeed) */

            _setext(2,1,0,0);
            _settextclr(4,0);
            _movtcurabs(200.0, 950.0);
```

```
text("DISTILLATE AND BOTTOMS PRODUCTS");
_movtcurabs(375.0,900.0);
text("[=]  lbmoles / hr");

_settextclr(11,0);
_movtcurabs(600.0,800.0);
text("Distill");

_movtcurabs(825.0,800.0);
text("Bottoms");

j = 0;
for (i=1;i(=20;++i)
{
     if (feed[i])
     {
          ++j;

          _movtcurabs(200.0, 750.0-(j*50.0));
          text(tx[i]);

          _movtcurabs(625.0, 750.0-(j*50.0));
          if (dist[i] ( 0.001)
           text("0.000");
          else
          {
           sprintf(tempstr, "%.3g", dist[i]);
           text(tempstr);
          }

          _movtcurabs(825.0, 750.0-(j*50.0));
          if (bott[i] ( 0.001)
           text("0.000");
          else
          {
           sprintf(tempstr, "%.3g", bott[i]);
           text(tempstr);
          }
     }
}

j += 2;
_movtcurabs(200.0, 750.0-(j*50.0));
text("Temperature");
_movtcurabs(625.0, 750.0-(j*50.0));
text("120.0 F");
_movtcurabs(825.0, 750.0-(j*50.0));
sprintf(tempstr,"%.1g F", t_boiler);
text(tempstr);

++j;
_movtcurabs(200.0, 750.0-(j*50.0));
text("Pressure [=] psi");
_movtcurabs(625.0, 750.0-(j*50.0));
sprintf(tempstr,"%.1g",p_cond);
text(tempstr);
```

```
        _movtcurabs(825.0, 750.0-(j*50.0));
        sprintf(tempstr,"%.1g",p_boiler);
        text(tempstr);

        _setext(2,1,0,1);
        _inittcur(2,4,11);
        _movtcurabs(10.0, 10.0);
        _settextclr(0,2);
        text(" RETURN for Column data : ESC when done ");
    }

    if (!toggle)
    {
        _setcolor(0);
        clr();

        diagram(0.0,0.0,200.0,1000.0,6,trays,truefeed,0);
    /* (xlow,ylow,xhigh,yhigh,color,trays,truefeed,diafeed) */

        _setext(2,1,0,0);
        _settextclr(4,0);
        _movtcurabs(250.0, 950.0);
        text("COLUMN OPERATING CONDITIONS");

        _setext(2,1,0,1);
        _inittcur(2,4,11);
        _movtcurabs(10.0, 10.0);
        _settextclr(0,2);
        text(" RETURN for Stream data : ESC when done ");


        _settextclr(11,0);
        _movtcurabs(200.0,850.0);
        text("Theoretical Stages");
        _movtcurabs(250.0,800.0);
        text("Minimum");
        _movtcurabs(750.0,800.0);
        sprintf(tempstr,"%.3g",N_min);
        text(tempstr);
        _movtcurabs(250.0,750.0);
        text("Actual");
        _movtcurabs(750.0,750.0);
        sprintf(tempstr,"%.3g",N_actual);
        text(tempstr);

        _movtcurabs(200.0,675.0);
        text("Reflux Ratio");
        _movtcurabs(250.0,625.0);
        text("Minimum");
        _movtcurabs(750.0,625.0);
        sprintf(tempstr,"%.3g",R_min);
        text(tempstr);
        _movtcurabs(250.0,575.0);
        text("Actual");
        _movtcurabs(750.0,575.0);
        sprintf(tempstr,"%.3g",R_actual);
```

```
        text(tempstr);

        _movtcurabs(200.0,500.0);
        text("Optimal Feed Stage");
        _movtcurabs(750.0,500.0);
        sprintf(tempstr,"%d",truefeed);
        text(tempstr);

        _movtcurabs(200.0,425.0);
        text("Condenser Type");
        _movtcurabs(250.0,375.0);
        text("Bubble Point Product Total");
        _movtcurabs(250.0,300.0);
        text("Temperature");
        _movtcurabs(750.0,300.0);
        text("120.0 F");
        _movtcurabs(250.0,250.0);
        text("Pressure [=] psi");
        _movtcurabs(750.0,250.0);
        sprintf(tempstr,"%.1g",p_cond);
        text(tempstr);

        _movtcurabs(200.0,175.0);
        text("Reboiler Conditions");
        _movtcurabs(250.0,125.0);
        text("Temperature");
        _movtcurabs(750.0,125.0);
        sprintf(tempstr,"%.1g F",t_boiler);
        text(tempstr);
        _movtcurabs(250.0,75.0);
        text("Pressure [=] psi");
        _movtcurabs(750.0,75.0);
        sprintf(tempstr,"%.1g",p_boiler);
        text(tempstr);
    }

    while (1)
    {
        int chr = _getch();

        if (chr == 13)      /* RETURN key  */
        {
            if (toggle)
                toggle = 0;
            else
                toggle = 1;
            break;
        }

        if (chr == 27)      /* ESC key     */
        {
            enough = 1;
            break;
        }
    }
```

```
    } while (!enough);



    printf("*****************************************************\n");
    printf("*                                                   *\n");
    printf("*                  Results from                     *\n");
    printf("*                                                   *\n");
    printf("*                   SEPTECH                         *\n");
    printf("*                                                   *\n");
    printf("*                                                   *\n");
    printf("*            a computer simulation of a             *\n");
    printf("*          multi-component distillation column      *\n");
    printf("*                                                   *\n");
    printf("*                by Russell L. Natter               *\n");
    printf("*                                                   *\n");
    printf("*            original code written 12/11/84         *\n");
    printf("*                                                   *\n");
    printf("*****************************************************\n");
    printf("\n\n");

    printf(" ------  OPERATING CONDITIONS  ------\n\n");
    printf("\tOperating Pressure  :   %.2g  psia\n",p_column);
    printf("\tMinimum Theoretical Stages    :   %.3g\n",N_min);
    printf("\tMinimum Reflux Ratio          :   %.3g\n",R_min);
    printf("\tActual Theoretical Stages     :   %.3g\n",N_actual);
    printf("\tActual Reflux Ratio      :   %.3g\n",R_actual);
    printf("\tOptimal Feed Stage  :   %d\n",truefeed);
    printf("\n");
    printf("  CONDENSER\n");
    printf("\tBubble Point Product Total Condenser\n");
    printf("\tTemperature         :   120.0  F  (Water Cooled)\n");
    printf("\tPressure       :   %.1g  psia\n",p_cond);
    printf("  REBOILER\n");
    printf("\tTemperature         :   %.1g  F\n",t_boiler);
    printf("\tPressure       :   %.1g  psia\n",p_boiler);
    printf("\n");
    printf("---  DISTILLATE AND BOTTOMS STREAMS [=] lbmole/hr ---\n\n");
    printf("\t\t\tDistill\t\tBottoms\n\n");

    for (i=1;i(=20;++i) /* table of out flows    */
    {
        if ((dist[i]) || (bott[i]))
        {
            printf(tx[i]);
            printf("\t\t%.3g\t\t%.3g\n",dist[i],bott[i]);
        }
    }

    printf("\nTemperature");
    printf("\t\t120.0\t\t%.1g\n",t_boiler);
    printf("Pressure");
    printf("\t\t%.2g\t\t%.2g\n",p_cond,p_boiler);

}
```

```
/*
     Last Revision
     4/21/86

     MC.H

*/

#define   DIAX1      90.0
#define   DIAY1      225.0
#define   DIAX2      610.0
#define   DIAY2      975.0

#define COMX1   660.0
#define COMY1   0.0
#define COMX2   1000.0
#define COMY2   1000.0

#define   NORMENX1  0.0
#define   NORMENY1  0.5
#define   NORMENX2  0.66
#define   NORMENY2  1.0

#define   TXTX       200.0
#define   TITLEX     100.0
#define   TITLEY     950.0
#define   STITLEY    900.0
#define   TXT1Y      810.0
#define   STXT1Y     770.0
#define   TXT2Y      710.0
#define   STXT2Y     670.0
#define   TXT3Y      610.0
#define   STXT3Y     570.0
#define   TXT4Y      510.0
#define   STXT4Y     470.0
#define   TXT5Y      410.0
#define   STXT5Y     370.0
#define   TXT6Y      310.0
#define   STXT6Y     270.0
#define   TXT7Y      210.0
#define   STXT7Y     170.0
#define   TXT8Y      110.0
#define   STXT8Y     70.0

#define   TRI1Y      820.0
#define   STRI1Y     790.0
#define   VTRI1Y     760.0
#define   TRI2Y      720.0
#define   STRI2Y     690.0
#define   VTRI2Y     660.0
#define   TRI3Y      620.0
#define   STRI3Y     590.0
#define   VTRI3Y     560.0
#define   TRI4Y      520.0
#define   STRI4Y     490.0
#define   VTRI4Y     460.0
```

```
#define   TRI5Y     420.0
#define   STRI5Y    390.0
#define   VTRI5Y    360.0
#define   TRI6Y     320.0
#define   STRI6Y    290.0
#define   VTRI6Y    260.0
#define   TRI7Y     220.0
#define   STRI7Y    190.0
#define   VTRI7Y    160.0
#define   TRI8Y     120.0
#define   STRI8Y    90.0
#define   VTRI8Y    60.0

#define   RIGHT_BT    ((bt == 1) || (bt == 129))
#define   LEFT_BT     ((bt == 2) || (bt == 130))
#define   BOTH_BT     ((bt == 3) || (bt == 131))
#define   CLICK       (((bt)=129) && (bt(=131)) || ((bt)=1) && (bt(=3)))

#define   RECTIFY     1
#define   STRIP       2
#define   SINGLE      3
#define   GENERAL     4

#define   RVOLATILE   1
#define   USERSPEC    2
#define   THERMODATA  3

int  eqltype, probtype, ratiodef, nostages;
int  points, operpts;

float     rratio, opslope, x_dist;
float     q, alpha, x_bott, x_feed, opl[3][10];
float     nordiax1, nordiay1, nordiax2, nordiay2,
          norcomx1, norcomy1, norcomx2, norcomy2;

double    x, y, eql[3][30];

int  rectcalc(), strng(), mainmenu(), new_prob(), new_eql();
int  curr_val(), copline(), menubox(), i_input(), work();
int  _getch(), sleep(), alter_cond();

float     mouse_slope(), mouse_x(), equil_x(), equil_y();
float     feedcond(), key_x(), diax(), diay(), invdiax(), invdiay(), f_input();
float     oper_x(), oper_y(), psinterx(), trial();


float     gpts[3][12];
int       genpoints;
int       mouse_in_use, optimal, feedtray,top_to_bott, bott_to_top, grid;
float     ofx, ofy;
```

```
/*
     COLOR.H

     Allows use of color names rather than code numbers while
     programming.  This version for

     AMDEK  (mode 4)

*/


#define BLACK       0
#define BLUE        1
#define GREEN       2
#define CYAN        3
#define RED         4
#define MAGENTA         5
#define MAG         5
#define BROWN       6
#define WHITE       7
#define GRAY        8
#define GREY        8
#define L_BLUE      9
#define L_GREEN         10
#define L_CYAN      11
#define L_RED       12
#define L_MAGENTA   13
#define YELLOW      14
#define HI_WHITE    15
```

```
/*
     Last Revision
     5/2/86

     MC1.C

     This file contains the following procedures:

          main()          oper_x()
          mainmenu()      oper_y()
          init_op()       equil_x()
          opmenu()        equil_y()
          rectsec()       ask_for_mouse()
          rectcalc()
          stripsec()
          stripcalc()
          mouse_slope()
          mouse_x()

*/



#include (math.h)
#include (color.h)
#include (stdio.h)
#include (mc.h)


main()
{
     int  option;

     ask_for_mouse();
     _initgraphics(4);
     init_op();
     init_eql();
     normcoor();

     do
     {
          _setviewport(0.0, 0.0, 1.0, 1.0, -1, BLACK);
          _setworld(0.0, 0.0, 1000.0, 1000.0);
          draw_dia();
          draw_eql();
          draw_answer();

          sleep();

          option = mainmenu();

          switch(option)
          {
               case 1:             /* Equilibrium Line */
                    eqlmenu();
                    break;
```

```
            case 2:             /* Operating Line   */
                 opmenu();
                 break;
            case 3:             /* File & Print Control  */
                 prtmenu();
                 break;
            case 4:             /* Clear All Entries     */
                 init_eql();
                 init_op();
                 break;
            case 5:             /* Quit Program          */
                 option = 0;
                 break;
        }
    } while (option);

    closegraphics();
}

ask_for_mouse()
{
    int  let_me_out = 0,
         chr, getch();

    printf("\n\n\tAre you using a MOUSE for a pointing device? (y/n):");
    while (!let_me_out)
    {
        switch (chr = getch())
        {
        case 'y':
        case 'Y':
             mouse_in_use = 1;
             let_me_out = 1;
             _setlocator(3, 1);
             break;
        case 'n':
        case 'N':
             mouse_in_use = 0;
             let_me_out = 1;
             break;
        default:
             break;
        }
    }
}


int mainmenu()
{
    int  option;

    _setviewport(NORMENX1, NORMENY1, NORMENX2, NORMENY2, GREEN, GREY);
    _setworld(0.0, 0.0, 1000.0, 1000.0);

    _setext(1,1,0,1);
    _settextclr(WHITE, RED);
```

```
    attext(100.0, STITLEY, "  MAIN MENU  ");
    _settextclr(YELLOW, GREY);
    attext(TXTX, TXT1Y, "EQUILIBRIUM LINE");
    attext(TXTX, TXT2Y, "OPERATING LINE");
    attext(TXTX, TXT3Y, "FILE & PRINT CONTROL");
    attext(TXTX, TXT4Y, "CLEAR ALL ENTRIES");
    attext(TXTX, TXT5Y, "TERMINATE");
    deltcur();

    option = menubox(5, BLACK, MAGENTA);

    return(option);
}

init_op()
{
    optimal  = 1;
    feedtray = 0;
    q = 0.0;

    nostages = 0;
    probtype = 0;
    opslope  = -1.0;
    x_bott   = -1.0;
    x_feed   = -1.0;
    x_dist   = -1.0;
    rratio = -1.0;
}

opmenu()
{
    int option;

    do
    {
    _setviewport(norcomx1, norcomy1, norcomx2, norcomy2, GREEN, GREY);
    _setworld(0.0, 0.0, 1000.0, 1000.0);

        _setext(1,1,0,1);
        _settextclr(WHITE, RED);
        attext(TITLEX, TITLEY, " OPERATING ");
        attext(TITLEX, STITLEY, "  OPTIONS  ");
        _settextclr(YELLOW, GREY);
        attext(TXTX, TXT1Y, "RECTIFYING");
        attext(TXTX, STXT1Y, "SECTION");
        attext(TXTX, TXT2Y, "STRIPPING");
        attext(TXTX, STXT2Y, "SECTION");
        attext(TXTX, TXT3Y, "SINGLE");
        attext(TXTX, STXT3Y, "FEED COL");
        attext(TXTX, TXT4Y, "NO CHANGE");
        deltcur();

        option = menubox(4, BLACK, MAGENTA);

        switch(option)
        {
```

```
          case 1:            /* Rectifying Section    */
               if ((probtype == RECTIFY) || (!probtype))
                    rectsec();
               else
                    if (rorl("   INITIALIZE VARIABLES   ",
                         "  AND START NEW PROBLEM?  ",
                         "NEW PROBLEM", "RETURN"))
                    {
                         init_op();
                         rectsec();
                    }
               break;
          case 2:            /* Stripping Section     */
               if ((probtype == STRIP) || (!probtype))
                    stripsec();
               else
                    if (rorl("   INITIALIZE VARIABLES   ",
                         "  AND START NEW PROBLEM?  ",
                         "NEW PROBLEM", "RETURN"))
                    {
                         init_op();
                         stripsec();
                    }
               break;
          case 3:            /* Single Feed Column    */
               if ((probtype == SINGLE) || (!probtype))
                    singlecol();
               else
                    if (rorl("   INITIALIZE VARIABLES   ",
                         "  AND START NEW PROBLEM?  ",
                         "NEW PROBLEM", "RETURN"))
                    {
                         init_op();
                         singlecol();
                    }
               break;
          case 4:            /* No Change        */
               option = 0;
               break;
          }
     } while (option);

}

stripsec()
{
     char lambda[20];
     int  option,
          cond_option,
          values;

     probtype = STRIP;

     do
     {
     _setviewport(norcomx1, norcomy1, norcomx2, norcomy2, GREEN, GREY);
```

```
_setworld(0.0, 0.0, 1000.0, 1000.0);

     _setext(1,1,0,1);
     _settextclr(WHITE, BLUE);
     attext(TITLEX, TITLEY, " SPECIFIED ");
     attext(TITLEX, STITLEY, "  VALUES   ");
     _settextclr(YELLOW, GREY);
     attext(TXTX, TRI1Y, "BOTTOMS");
     attext(TXTX, STRI1Y, "COMPOSIT'N");
     attext(TXTX, TRI2Y, "FEED");
     attext(TXTX, STRI2Y, "COMPOSIT'N");
     attext(TXTX, TRI3Y, "OPERATING");
     attext(TXTX, STRI3Y, "SLOPE");
     attext(TXTX, TRI4Y, "NUMBER OF");
     attext(TXTX, STRI4Y, "STAGES");
     _settextclr(GREEN, GREY);
     attext(TXTX, TXT5Y, "BEGIN");
     attext(TXTX, STXT5Y, "CALCULAT'N");
     attext(TXTX, TXT6Y, "CHANGE");
     attext(TXTX, STXT6Y, "CONDITIONS");
     attext(TXTX, TXT7Y, "RETURN");

     attext(80.0, TXT8Y, "MUST SPECIFY");
     attext(160.0, STXT8Y, "THREE VALUES");
     deltcur();

     if (x_bott != -1.0)
     {
          _settextclr(WHITE, GREY);
          sprintf(lambda, "  %.3g", x_bott);
          attext(TXTX, VTRI1Y, lambda);
          _settextclr(L_BLUE, GREY);
          attext(0.0, TXT1Y, "*");
     }
     else
     {
          _settextclr(WHITE, GREY);
          attext(TXTX, VTRI1Y, " _____");
     }
     if (x_feed != -1.0)
     {
          _settextclr(WHITE, GREY);
          sprintf(lambda, "  %.3g", x_feed);
          attext(TXTX, VTRI2Y, lambda);
          _settextclr(L_BLUE, GREY);
          attext(0.0, TXT2Y, "*");
     }
     else
     {
          _settextclr(WHITE, GREY);
          attext(TXTX, VTRI2Y, " _____");
     }
     if (opslope != -1.0)
     {
          _settextclr(WHITE, GREY);
          sprintf(lambda, "  %.3g", opslope);
```

```
        attext(TXTX, VTRI3Y, lambda);
        _settextclr(L_BLUE, GREY);
        attext(0.0, TXT3Y, "*");
    }
    else
    {
        _settextclr(WHITE, GREY);
        attext(TXTX, VTRI3Y, " _____");
    }
    if (nostages != 0)
    {
        _settextclr(WHITE, GREY);
        sprintf(lambda, "  %d", nostages);
        attext(TXTX, VTRI4Y, lambda);
        _settextclr(L_BLUE, GREY);
        attext(0.0, TXT4Y, "*");
    }
    else
    {
        _settextclr(WHITE, GREY);
        attext(TXTX, VTRI4Y, " _____");
    }
    deltcur();

    option = menubox(7, BLACK, MAGENTA);

    switch(option)
    {
    case 1:                 /* Bottoms Comp.    */
        if (x_bott != -1.0)
        {
            sprintf(lambda, "%.3g", x_bott);
            option = curr_val(lambda);

            if (option == 2)
            {
                x_bott = -1.0;
                break;
            }
            if (option == 3)
                break;
        }
        if (mork())
            x_bott = mouse_x();
        else
            x_bott = key_x("MOLE FRACTION OF LIGHT",
                    "COMPONENT IN BOTTOMS");
        break;
    case 2:                 /* Feed Comp.       */
        if (x_feed != -1.0)
        {
            sprintf(lambda, "%.3g", x_feed);
            option = curr_val(lambda);

            if (option == 2)
            {
```

```
                x_feed = -1.0;
                break;
            }
            if (option == 3)
                break;
        }
        if (mork())
            x_feed = mouse_x();
        else
            x_feed = key_x("MOLE FRACTION OF LIGHT",
                    "COMPONENT IN FEED");
        x_dist = x_feed;
        break;
    case 3:                 /* Operating Slope  */
        if (opslope != -1.0)
        {
            sprintf(lambda, "%.3g", opslope);
            option = curr_val(lambda);

            if (option == 2)
            {
                opslope = -1.0;
                break;
            }
            if (option == 3)
                break;
        }
        if (mork())
            opslope = mouse_slope();
        else
            opslope = key_x("SLOPE OF OPERATING LINE",
                    "LIQUID / VAPOR  RATES");
        break;
    case 4:                 /* Number of Stages      */
        if (nostages != 0)
        {
            sprintf(lambda, "%d", nostages);
            option = curr_val(lambda);

            if (option == 2)
            {
                nostages = 0;
                break;
            }
            if (option == 3)
                break;
        }
        nostages = key_x("NUMBER OF THEORETICAL",
                "EQUILIBRIUM STAGES");
        break;
    case 5:                 /* Begin Calculations    */
        stripcalc();
        break;
    case 6:
        draw_cond(1);  /* draw in "menu" color scheme     */
        cond_option = alter_cond();
```

```
            if (cond_option == 1)    /* optimal feed tray     */
            {
                ermess(6,
                    "THE FEED TRAY",
                    " LOCATION FOR",
                    " A STRIPPING ",
                    "  COLUMN IS  ",
                    "  IMPLICITLY ",
                    "   DEFINED.  ");
            }
            if (cond_option == 2)    /* feed condition   */
            {
                q = feedcond();
            }
            if (cond_option == 3)    /* equilibrium      */
            {
                eqlmenu();
            }
            draw_cond(0);  /* draw in display color scheme    */
            break;
        case 7:                  /* No Change        */
            option = 0;
            break;
        }
    } while (option);

}

int stripcalc()
{
    float    cur_x, cur_y, mx, my;
    int  specified, solvfor, bt;

    specified = 0;

    if (x_bott == -1.0)
        solvfor = 1;
    else
        specified++;

    if (x_feed == -1.0)
        solvfor = 2;
    else
        specified++;

    if (opslope == -1.0)
        solvfor = 3;
    else
        specified++;

    if (nostages == 0)
        solvfor = 4;
    else
        specified++;

    if (!eqltype)
```

```
    {
        ermess(5,
            "     AN",
            " EQUILIBRIUM",
            "  LINE MUST  ",
            "   FIRST BE",
            "  SPECIFIED.");
        return(0);
    }
    if (specified (= 2)      /* under specified problem    */
    {
        ermess(5,
            "ALTEAST THREE",
            "   OF THE",
            "  VARIABLES",
            "   MUST BE",
            "  SPECIFIED.");
        return(0);
    }

    if (specified )= 4)      /* over specified problem     */
    {
        ermess(5,
            " ATLEAST ONE",
            "   OF THE",
            "  VARIABLES",
            "   MUST BE",
            " UNSPECIFIED.");
        return(0);
    }

    if ((x_bott )= x_feed) && (x_feed != -1.0))
    {
        ermess(7,
            " COMPOSITION",
            "OF THE LIGHT",
            "COMPONENT IN",
            "FEED MUST BE",
            "GREATER THAN",
            " THAT OF THE",
            "   BOTTOMS.");
        return(0);
    }

    if (nostages ( 0)
    {
        ermess(5,
            " A NEGATIVE",
            "  NUMBER OF",
            " STAGES HAS",
            "    BEEN",
            "  SPECIFIED.");
        return(0);
    }

    if ((opslope (= 1.0) && (opslope != -1.0))
```

```
    {
        ermess(5,
            " THE SLOPE OF",
            "THE OPERATING",
            " LINE MUST BE",
            " GREATER THAN",
            "     ONE.");
        return(0);
    }

    _setviewport(0.0, 0.0, 0.66, 1.0, -1, BLACK);
    _setworld(0.0, 0.0, 660.0, 1000.0);
    draw_dia();
    draw_eql();

    operpts = 2;

/* Solving for the Number of Trays in a Stripping Column    */
    if (solvfor == 4)
    {
        char lambda[20];
        int  tmp, stagetrk;

        if (!copline(1))
        {
            ermess(   4,
                "THE SPECIFIED",
                "  SYSTEM IS",
                " PHYSICALLY",
                " IMPOSSIBLE");
            return(0);
        }
        cur_x = opl[1][2];
        cur_y = opl[2][2];
        _setcolor(WHITE);
        _movabs(diax(cur_x), diay(cur_y)); /* Feed Location */

        stagetrk = 0;
        while (cur_x ) x_bott)
        {
            cur_x = equil_x(cur_y);
            _lnabs(diax(cur_x), diay(cur_y));
            cur_y = oper_y(cur_x);
            _lnabs(diax(cur_x), diay(cur_y));
            stagetrk++;
        }
        nostages = stagetrk;              /* The Answer  */

        sprintf(lambda, "%d STAGES", stagetrk);

    _setviewport(norcomx1, norcomy1, norcomx2, norcomy2, GREEN, GREY);
    _setworld(0.0, 0.0, 1000.0, 1000.0);

        _setext(2,1,0,1);
        _settextclr(WHITE, RED);
        attext(40.0, 850.0, " THE ANSWER ");
```

```
        _setext(1,1,0,1);
        _settextclr(YELLOW, GREY);
        attext(210.0, 700.0, "FOR THE");
        attext(210.0, 650.0, "DESIRED");
        attext(70.0, 600.0, "SEPARATION,");
        attext(140.0, 500.0, lambda);
        attext(70.0, 450.0, "ARE NEEDED.");

        _settextclr(L_GREEN, GREY);
        attext(160.0, 250.0, "CLICK ANY");
        attext(10.0, 200.0, "BUTTON ON THE");
        attext(160.0, 150.0, "MOUSE TO");
        attext(160.0, 100.0, "CONTINUE.");
        deltcur();

        sleep();

        return(1);
    }

/* Solve for the Bottoms Composition        */
    if (solvfor == 1)
    {
        int  i;
        float    cur_x, cur_y, overshot, upper, lower;


        upper = x_feed;
        lower = 0.0;

        do
        {
            x_bott = ((upper - lower)/2) + lower;

            if (copline(2))
            {
                overshot = trial();
                if (overshot ) 0.0)
                    upper = x_bott;
                if (overshot ( 0.0)
                    lower = x_bott;
                if (overshot == 0.0)
                    break;
            }
            else
            {
                lower = x_bott;
                overshot = 1.0;
            }
            ++i;
            if (i == 25)
            {
                ermess(  4,
                    "THE SPECIFIED",
                    "  SYSTEM IS",
                    " PHYSICALLY",
```

```
                    " IMPOSSIBLE");
                x_bott = -1.0;
                return(0);
            }

        } while (fabs(overshot) ) 0.001);

        draw_answer();
        return(1);
    }

/* Solve for Feed Composition      */
    if (solvfor == 2)
    {
        int  i;
        float     overshot, upper, lower;

        upper = 1.0;
        lower = x_bott;
        i = 0;
        do
        {
            x_feed = ((upper - lower)/2) + lower;
            i++;

            if (copline(2))
            {
                overshot = trial();
                if (overshot ) 0.0)
                    lower = x_feed;
                if (overshot ( 0.0)
                    upper = x_feed;
                if (overshot == 0.0)
                    break;
            }
            else
            {
                upper = x_feed;
                overshot = 1.0;
            }
            if (i == 25)
            {
                ermess(   4,
                    "THE SPECIFIED",
                    "  SYSTEM IS",
                    " PHYSICALLY",
                    " IMPOSSIBLE");
                x_feed = -1.0;
                return(0);
            }

        } while (fabs(overshot) ) 0.001);

        draw_answer();
        return(1);
    }
```

```
/* Solve for Operating Line Slope       */
     if (solvfor == 3)
     {
          int  i;
          float     overshot, upper, lower;

          upper = 3.0;
          lower = 1.0;

          do
          {
               opslope = ((upper - lower)/2) + lower;

               if (copline(2))
               {
                    overshot = trial();
                    if (overshot ) 0.0)
                         lower = opslope;
                    if (overshot ( 0.0)
                         upper = opslope;
                    if (overshot == 0.0)
                         break;
               }
               else
               {
                    upper = opslope;
                    overshot = 1.0;
               }
               if (++i == 25)
               {
                    ermess(   4,
                         "THE SPECIFIED",
                         "  SYSTEM IS",
                         " PHYSICALLY",
                         " IMPOSSIBLE");
                    opslope = -1.0;
                    return(0);
               }

          } while (fabs(overshot) ) 0.001);

          draw_answer();
          return(1);
     }
}


rectsec()
{
     char lambda[20];
     int  option,
          cond_option,
          values;

     probtype = RECTIFY;
```

```
do
{
_setviewport(norcomx1, norcomy1, norcomx2, norcomy2, GREEN, GREY);
_setworld(0.0, 0.0, 1000.0, 1000.0);

     _setext(1,1,0,1);
     _settextclr(WHITE, BLUE);
     attext(TITLEX, TITLEY, " SPECIFIED ");
     attext(TITLEX, STITLEY, "  VALUES   ");
     _settextclr(YELLOW, GREY);
     attext(TXTX, TRI1Y, "DISTILLATE");
     attext(TXTX, STRI1Y, "COMPOSIT'N");
     attext(TXTX, TRI2Y, "FEED");
     attext(TXTX, STRI2Y, "COMPOSIT'N");
     attext(TXTX, TRI3Y, "OPERATING");
     attext(TXTX, STRI3Y, "SLOPE");
     attext(TXTX, TRI4Y, "NUMBER OF");
     attext(TXTX, STRI4Y, "STAGES");
     _settextclr(GREEN, GREY);
     attext(TXTX, TXT5Y, "BEGIN");
     attext(TXTX, STXT5Y, "CALCULAT'N");
     attext(TXTX, TXT6Y, "CHANGE");
     attext(TXTX, STXT6Y, "CONDITIONS");
     attext(TXTX, TXT7Y, "RETURN");

     attext(80.0, TXT8Y, "MUST SPECIFY");
     attext(160.0, STXT8Y, "THREE VALUES");
     deltcur();
     if (x_dist != -1.0)
     {
          _settextclr(WHITE, GREY);
          sprintf(lambda, "  %.3g", x_dist);
          attext(TXTX, VTRI1Y, lambda);
          _settextclr(L_BLUE, GREY);
          attext(0.0, TXT1Y, "*");
     }
     else
     {
          _settextclr(WHITE, GREY);
          attext(TXTX, VTRI1Y, " _____");
     }
     if (x_feed != -1.0)
     {
          _settextclr(WHITE, GREY);
          sprintf(lambda, "  %.3g", x_feed);
          attext(TXTX, VTRI2Y, lambda);
          _settextclr(L_BLUE, GREY);
          attext(0.0, TXT2Y, "*");
     }
     else
     {
          _settextclr(WHITE, GREY);
          attext(TXTX, VTRI2Y, " _____");
     }
     if (opslope != -1.0)
     {
```

```
     _settextclr(WHITE, GREY);
     sprintf(lambda, "  %.3g", opslope);
     attext(TXTX, VTRI3Y, lambda);
     _settextclr(L_BLUE, GREY);
     attext(0.0, TXT3Y, "*");
}
else
{
     _settextclr(WHITE, GREY);
     attext(TXTX, VTRI3Y, " _____");
}
if (nostages != 0)
{
     _settextclr(WHITE, GREY);
     sprintf(lambda, "  %d", nostages);
     attext(TXTX, VTRI4Y, lambda);
     _settextclr(L_BLUE, GREY);
     attext(0.0, TXT4Y, "*");
}
else
{
     _settextclr(WHITE, GREY);
     attext(TXTX, VTRI4Y, " _____");
}
deltcur();

option = menubox(7, BLACK, MAGENTA);

switch(option)
{
case 1:                  /* Distillate Comp. */
     if (x_dist != -1.0)
     {
          sprintf(lambda, "%.3g", x_dist);
          option = curr_val(lambda);

          if (option == 2)
          {
               x_dist = -1.0;
               break;
          }
          if (option == 3)
               break;
     }
     if (mork())
          x_dist = mouse_x();
     else
          x_dist = key_x("MOLE FRACTION OF LIGHT",
                    "COMPONENT IN DISTILLATE");
     break;
case 2:                  /* Feed Comp.      */
     if (x_feed != -1.0)
     {
          sprintf(lambda, "%.3g", x_feed);
          option = curr_val(lambda);
```

```
            if (option == 2)
            {
                x_feed = -1.0;
                break;
            }
            if (option == 3)
                break;
        }
        if (mork())
            x_feed = mouse_x();
        else
            x_feed = key_x("MOLE FRACTION OF LIGHT",
                    "COMPONENT IN FEED");
        x_bott = x_feed;
        break;
    case 3:                 /* Operating Slope  */
        if (opslope != -1.0)
        {
            sprintf(lambda, "%.3g", opslope);
            option = curr_val(lambda);

            if (option == 2)
            {
                opslope = -1.0;
                break;
            }
            if (option == 3)
                break;
        }
        if (mork())
            opslope = mouse_slope();
        else
            opslope = key_x("SLOPE OF OPERATING LINE",
                    "LIQUID / VAPOR  RATES");
        break;
    case 4:                 /* Number of Stages      */
        if (nostages != 0)
        {
            sprintf(lambda, "%d", nostages);
            option = curr_val(lambda);

            if (option == 2)
            {
                nostages = 0;
                break;
            }
            if (option == 3)
                break;
        }
        nostages = key_x("NUMBER OF THEORETICAL",
                "EQUILIBRIUM STAGES");
        break;
    case 5:                 /* Begin Calculations    */
        rectcalc();
        break;
    case 6:
```

```
                draw_cond(1);  /* draw in "menu" color scheme     */
                cond_option = alter_cond();
                if (cond_option == 1)    /* optimal feed tray     */
                {
                     ermess(6,
                          "THE FEED TRAY",
                          " LOCATION FOR",
                          " A RECTIFYING",
                          "  COLUMN IS  ",
                          "  IMPLICITLY ",
                          "   DEFINED.  ");
                }
                if (cond_option == 2)    /* feed condition   */
                {
                     q = feedcond();
                }
                if (cond_option == 3)    /* equilibrium      */
                {
                     eqlmenu();
                }
                draw_cond(0);  /* draw in display color scheme    */
                break;
           case 7:                 /* No Change        */
                option = 0;
                break;
           }
     } while (option);

}

int rectcalc()
{
     float     cur_x, cur_y, mx, my;
     int  specified, solvfor, bt;

     if (x_dist == -1.0)
          solvfor = 1;
     else
          specified++;

     if (x_feed == -1.0)
          solvfor = 2;
     else
          specified++;

     if (opslope == -1.0)
          solvfor = 3;
     else
          specified++;

     if (nostages == 0)
          solvfor = 4;
     else
          specified++;

     if (!eqltype)
```

```
{
     ermess(5,
          "     AN",
          " EQUILIBRIUM",
          "  LINE MUST  ",
          "   FIRST BE",
          "  SPECIFIED.");
     return(0);
}

if (specified (= 2)      /* under specified problem    */
{
     ermess(5,
          "ALTEAST THREE",
          "   OF THE",
          "  VARIABLES",
          "   MUST BE",
          "  SPECIFIED.");
     return(0);
}

if (specified )= 4)      /* over specified problem     */
{
     ermess(5,
          " ATLEAST ONE",
          "   OF THE",
          "  VARIABLES",
          "   MUST BE",
          " UNSPECIFIED.");
     return(0);
}

if ((x_dist (= x_feed) && (x_dist != -1.0))
{
     ermess(7,
          " COMPOSITION",
          "OF THE LIGHT",
          "COMPONENT IN",
          "FEED MUST BE",
          "  LESS THAN",
          " THAT OF THE",
          " DISTILLATE.");
     return(0);
}

if (nostages ( 0)
{
     ermess(5,
          " A NEGATIVE",
          "  NUMBER OF",
          " STAGES HAS",
          "    BEEN",
          "  SPECIFIED.");
     return(0);
}
```

```
    if (opslope )= 1.0)
    {
        ermess(5,
            " THE SLOPE OF",
            "THE OPERATING",
            " LINE MUST BE",
            "  LESS THAN",
            "     ONE.");
        return(0);
    }

    _setviewport(0.0, 0.0, 0.66, 1.0, -1, BLACK);
    _setworld(0.0, 0.0, 660.0, 1000.0);
    draw_dia();
    draw_eql();

    operpts = 2;

/* Solving for the Number of Trays in a Rectifying Column   */
    if (solvfor == 4)
    {
        char lambda[20];
        int  stagetrk;

        if (!copline(1))
        {
            ermess(   4,
                "THE SPECIFIED",
                "  SYSTEM IS",
                " PHYSICALLY",
                " IMPOSSIBLE");
            return(0);
        }
        cur_x = op1[1][1];
        cur_y = op1[2][1];
        _setcolor(WHITE);
        _movabs(diax(cur_x), diay(cur_y)); /* Feed Location */

        stagetrk = 0;
        while (cur_x ( x_dist)
        {
            cur_y = equil_y(cur_x);
            _lnabs(diax(cur_x), diay(cur_y));
            cur_x = oper_x(cur_y);
            _lnabs(diax(cur_x), diay(cur_y));
            stagetrk++;
        }
        nostages = stagetrk;              /* The Answer  */

        sprintf(lambda, "%d STAGES", stagetrk);

    _setviewport(norcomx1, norcomy1, norcomx2, norcomy2, GREEN, GREY);
    _setworld(0.0, 0.0, 1000.0, 1000.0);

        _setext(2,1,0,1);
        _settextclr(WHITE, RED);
```

```
        attext(40.0, 850.0, " THE ANSWER ");
        _setext(1,1,0,1);
        _settextclr(YELLOW, GREY);
        attext(210.0, 700.0, "FOR THE");
        attext(210.0, 650.0, "DESIRED");
        attext(70.0, 600.0, "SEPARATION,");
        attext(140.0, 500.0, lambda);
        attext(70.0, 450.0, "ARE NEEDED.");

        _settextclr(L_GREEN, GREY);
        attext(160.0, 250.0, "CLICK ANY");
        attext(10.0, 200.0, "BUTTON ON THE");
        attext(160.0, 150.0, "MOUSE TO");
        attext(160.0, 100.0, "CONTINUE.");
        deltcur();

        sleep();

        return(1);
    }
/* Solve for the Distillate Composition      */
    if (solvfor == 1)
    {
        int  i;
        float     cur_x, cur_y, overshot, upper, lower;

        upper = 1.0;
        lower = x_feed;

        do
        {
            x_dist = ((upper - lower)/2) + lower;

            if (copline(2))
            {
                overshot = trial();
                if (overshot ) 0.0)
                    lower = x_dist;
                if (overshot ( 0.0)
                    upper = x_dist;
                if (overshot == 0.0)
                    break;
            }
            else
            {
                upper = x_dist;
                overshot = 1.0;
            }
            ++i;
            if (i == 25)
            {
                ermess(  4,
                    "THE SPECIFIED",
                    "  SYSTEM IS",
                    " PHYSICALLY",
                    " IMPOSSIBLE");
```

```
                    x_dist = -1.0;
                    return(0);
               }

          } while (fabs(overshot) ) 0.001);

          draw_answer();
          return(1);
     }
/* Solve for Feed Composition      */
     if (solvfor == 2)
     {
          int  i;
          float     overshot, upper, lower;

          upper = x_dist;
          lower = 0.0;
          i = 0;
          do
          {
               x_feed = ((upper - lower)/2) + lower;
               i++;
               if (copline(2))
               {
                    overshot = trial();
                    if (overshot ) 0.0)
                         upper = x_feed;
                    if (overshot ( 0.0)
                         lower = x_feed;
                    if (overshot == 0.0)
                         break;
               }
               else
               {
                    upper = x_feed;
                    overshot = 1.0;
               }
               if (i == 25)
               {
                    ermess(   4,
                         "THE SPECIFIED",
                         "  SYSTEM IS",
                         " PHYSICALLY",
                         " IMPOSSIBLE");
                    x_feed = -1.0;
                    return(0);
               }

          } while (fabs(overshot) ) 0.001);

          draw_answer();
          return(1);
     }
/* Solve for Operating Line Slope       */
     if (solvfor == 3)
     {
```

```
        int  i;
        float     overshot, upper, lower;

        upper = 1.0;
        lower = 0.0;

        do
        {
             opslope = ((upper - lower)/2) + lower;

             if (copline(2))
             {
                  overshot = trial();
                  if (overshot ) 0.0)
                       upper = opslope;
                  if (overshot ( 0.0)
                       lower = opslope;
                  if (overshot == 0.0)
                       break;
             }
             else
             {
                  lower = opslope;
                  overshot = 1.0;
             }
             if (++i == 25)
             {
                  ermess(   4,
                       "THE SPECIFIED",
                       "  SYSTEM IS",
                       " PHYSICALLY",
                       " IMPOSSIBLE");
                  opslope = -1.0;
                  return(0);
             }

        } while (fabs(overshot) ) 0.001);

        draw_answer();
        return(1);
    }
}


float mouse_slope()
{
    char lambda[8];
    float     mx, my, slope;
    int  bt;

    _setviewport(0.0, 0.0, 0.66, 1.0, -1, BLACK);
    _setworld(0.0, 0.0, 660.0, 1000.0);
    draw_dia();
    draw_eql();

    ftinit();
```

```
    _ftsize(1,10);
    _setcolor(WHITE);
    _ftcolor(WHITE, BLACK);
    _movabs(diax(0.0), diay(0.0));
    _orglocator(diax(0.7), diay(0.3));
    do
    {
        readlocator(&mx, &my, &bt);
        if (bt )= 128)
        {
            if ((mx )= diax(0.0)) && (my )= diay(0.0)))
            {
                if (invdiax(mx) == 0.0)
                    slope = 0.0;
                else
                    slope = invdiay(my) / invdiax(mx);
                _rlnabs(mx, my);
                sprintf(lambda, "%10.3f", slope);
                _ftlocate(20, 15);
                ftext(lambda);
            }
        }
    } while (!CLICK);

    return(slope);
}


float mouse_x()
{
    char lambda[8];
    float    mx, my;
    int  bt;

    _setviewport(0.0, 0.0, 0.66, 1.0, -1, BLACK);
    _setworld(0.0, 0.0, 660.0, 1000.0);
    draw_dia();
    draw_eql();

    ftinit();
    _ftsize(1,10);
    _setcolor(WHITE);
    _ftcolor(WHITE, BLACK);
    _orglocator(diax(0.5), diay(0.5));
    do
    {
        readlocator(&mx, &my, &bt);
        if (bt )= 128)
        {
            if ((mx (= diax(1.0)) && (mx )= diax(0.0)))
            {
                _movabs(mx, diay(0.0));
                _rlnabs(mx, diay(1.0));
                sprintf(lambda, "%.3f", invdiax(mx));
                _ftlocate(20, 20);
                ftext(lambda);
```

```
            }
        }
    } while (!CLICK);

    return(invdiax(mx));
}


float oper_y(x)
double x;
{
    int  i;
    double    low_x, high_x;
    double    low_y, high_y;
    float     y;

    if (probtype == STRIP)
        x_dist = x_feed;
    if (probtype == RECTIFY)
        x_bott = opl[1][1];

    if ((x ( x_bott) || (x ) x_dist))
    {
        y = (float)(x);
        return(y);
    }

    if ((x ( 0.0) || (x ) 1.0))
    {
        printf("\n\n\tINVALID ARGUEMENT IN OPER_Y\n\n");
        exit(1);
    }

    for (i = 1; i (= operpts; i++)
    {
        low_x  = opl[1][i];
        high_x = opl[1][i+1];

        if ((low_x (= x) && (high_x )= x))
        {
            low_y  = opl[2][i];
            high_y = opl[2][i+1];
            break;
        }
    }

    y = (float)(low_y + ((x-low_x)/(high_x-low_x))*(high_y-low_y));

    if (probtype == STRIP)
        x_dist = -1.0;
    if (probtype == RECTIFY)
        x_bott = -1.0;

    return(y);
}
```

```
float oper_x(y)
double y;
{
     int  i;
     double    low_x, high_x;
     double    low_y, high_y;
     float     x;

     if ((y < opl[2][1]) || (y > opl[2][operpts]))
     {
          x = (float)(y);
          return(x);
     }

     if ((y < 0.0) || (y > 1.0))
     {
          printf("\n\n\tINVALID ARGUEMENT IN OPER_X\n\n");
          exit(1);
     }

     for (i = 1; i <= operpts; i++)
     {
          low_y  = opl[2][i];
          high_y = opl[2][i+1];

          if ((low_y <= y) && (high_y >= y))
          {
               low_x  = opl[1][i];
               high_x = opl[1][i+1];
               break;
          }
     }

     x = (float)(low_x + ((y-low_y)/(high_y-low_y))*(high_x-low_x));

     return(x);
}

/* linear interpolation of equilibrium line: */

float equil_y(x)
double x;
{
     int  i;
     double    low_x, high_x;
     double    low_y, high_y;
     float     y;

     if ((x < 0.0) || (x > 1.0))
     {
          printf("\n\n\tINVALID ARGUEMENT IN EQUIL_Y\n\n");
          exit(1);
     }

     for (i = 0; i <= points; i++)
     {
```

```
          low_x  = eql[1][i];
          high_x = eql[1][i+1];

          if ((low_x (= x) && (high_x )= x))
          {
               low_y  = eql[2][i];
               high_y = eql[2][i+1];
               break;
          }
     }

     y = (float)(low_y + ((x-low_x)/(high_x-low_x))*(high_y-low_y));

     return(y);
}



float equil_x(y)
double y;
{
     int  i;
     double    low_x, high_x;
     double    low_y, high_y;
     float     x;

     if ((y ( 0.0) || (y ) 1.0))
     {
          printf("\n\n\tINVALID ARGUEMENT IN EQUIL_X\n\n");
          exit(1);
     }

     for (i = 0; i (= points; i++)
     {
          low_y  = eql[2][i];
          high_y = eql[2][i+1];

          if ((low_y (= y) && (high_y )= y))
          {
               low_x  = eql[1][i];
               high_x = eql[1][i+1];
               break;
          }
     }

     x = (float)(low_x + ((y-low_y)/(high_y-low_y))*(high_x-low_x));

     return(x);
}
```

```
/*
     Last Revision
     4/21/86

     MC2.C

     This file contains the following procedures:

          init_eql()     savedia()
          eqlmenu()      retrieve()
          user_sup()     ctl_char()
          no_avail()     strng()
          getrvol()      _strng()
          relvol()       sleep()
          draw_eql()
          prtmenu()
          ermess()
          psinterx()
          feedcond()
          work()
          rorl()
          key_x()
          normcoor()


*/

#include (math.h)
#include (color.h)
#include (stdio.h)
#include (mcext.h)


init_eql()
{
     eqltype     = 0;
     eql[1][0] = -1.0;
     alpha     = -1.0;
}

eqlmenu()
{
     int option;

     do
     {
     _setviewport(norcomx1, norcomy1, norcomx2, norcomy2, GREEN, GREY);
     _setworld(0.0, 0.0, 1000.0, 1000.0);

          _setext(1,1,0,1);
          _settextclr(WHITE, RED);
          attext(100.0, TITLEY, " EQUILIBRIUM ");
          attext(100.0, STITLEY, " DEFINITION  ");
          _settextclr(YELLOW, GREY);
          attext(TXTX, TXT1Y, "RELATIVE");
          attext(TXTX, STXT1Y, "VOLATILITY");
```

```
        attext(TXTX, TXT2Y, "USER");
        attext(TXTX, STXT2Y, "SPECIFIED");
        attext(TXTX, TXT3Y, "THERMO");
        attext(TXTX, STXT3Y, "DATABASE");
        attext(TXTX, TXT4Y, "NO CHANGE");
        deltcur();
        option = menubox(4, BLACK, MAGENTA);

        switch(option)
        {
        case 1:             /* Relative Volatility   */
            if ((eqltype == RVOLATILE) || (!eqltype))
                getrvol();
            else
                if (rorl("       REDEFINE THE       ",
                     "     EQUILIBRIUM LINE?    ",
                     "REDEFINITION", "RETURN"))
                {
                    init_eql();
                    getrvol();
                }
            break;
        case 2:             /* User Defined          */
            if ((eqltype == USERSPEC) || (!eqltype))
                user_sup();
            else
                if (rorl("       REDEFINE THE       ",
                     "     EQUILIBRIUM LINE?    ",
                     "REDEFINITION", "RETURN"))
                {
                    init_eql();
                    user_sup();
                }
            break;
        case 3:             /* Thermo Database  */
            no_avail();
            break;
        case 4:             /* No Change        */
            option = 0;
            break;
        }
    } while (option);
}

user_sup()
{
    float    omega, raw[3][30];
    int  elim = 0, done = 0,
        k = 0,          j = 0,
        half = 0, screens = 0,
        num = 0,  i = 0,         offset = 0;
    int  use_mouse = 0,      finis =0;
    char     lambda[20];

    eqltype = USERSPEC;
```

```
    raw[1][0] = raw[2][0] = eq1[1][0] = eq1[2][0] = 0.0;

    if (mouse_in_use)
    {
         use_mouse = mork();
    }
    click_buffer();

    if (use_mouse)
    {
         _setviewport(0.0, 0.0, 1.0, 1.0, -1, BLACK);

         _setviewport(norcomx1,norcomy1,norcomx2,norcomy2,GREEN,GREY);
         _setworld(0.0, 0.0, 1000.0, 1000.0);
         _setext(1,1,0,1);
         _settextclr(WHITE, BLUE);
         attext(50.0, 975.0, " EQUILBRIUM  ");
         attext(50.0, 950.0, " LINE POINTS ");
         _settextclr(WHITE, RED);
         attext(50.0, 75.0, "CHOOSE (0, 0)");
         attext(50.0, 50.0, "   TO QUIT   ");
         _settextclr(YELLOW, GREY);
         attext(50.0, 900.0, "  X       Y  ");
         attext(50.0, 875.0, "====== ======");

         _settextclr(YELLOW, GREY);
      do
      {
         i++;
         mouse_xy(&raw[1][i], &raw[2][i]);
         if ((raw[1][i] == 0.0) && (raw[2][i] == 0.0))
         {
              finis = 1;
              i--;
         }
         else
         {
         _setviewport(norcomx1,norcomy1,norcomx2,norcomy2,-1,-1);
         _setworld(0.0, 0.0, 1000.0, 1000.0);
         sprintf(lambda, "%.2f   %.2f", raw[1][i], raw[2][i]);
         _settextclr(YELLOW, GREY);
         attext(50.0, 850.0-(i*25.0), lambda);
         }
      } while (!finis);

         points = i;
         i = 0;
    }
    else
    {

         _setviewport(0.0, 0.0, 0.66, 1.0, GREEN, BLUE);
         _setworld(0.0, 0.0, 660.0, 1000.0);
         _setext(1,1,0,1);
         _settextclr(WHITE, RED);
         attext(5.0, 950.0, " ENTER THE COORDINATES OF ");
```

```
        attext(5.0, 900.0, "  THE EQUILIBRIUM LINE.   ");
        _settextclr(RED, BLUE);
        attext(100.0, 800.0, "X VALUE");
        attext(375.0, 800.0, "Y VALUE");
        _setcolor(RED);
        _movabs(100.0, 790.0);
        _lnabs(275.0, 790.0);
        _movabs(375.0, 790.0);
        _lnabs(550.0, 790.0);
        _settextclr(L_GREEN, BLUE);
        attext(10.0, 10.0, "WHEN DONE, HIT (RETURN)");
        _settextclr(WHITE, BLUE);

    /*  This while loop ends when a zero length string is returned
        from the strng() function. (i.e., (CR) w/ no number )
    */
    while (strng(75.0 + offset*275.0, 725.0 - (25.0*i),lambda,0, 10) != 0)
        {
            if (offset == 0)
            {
                offset = 1;
                omega = atof(lambda);
                num++;
                raw[1][num] = (float)omega;
                half = 1;

            }
            else
            {
                half = 0;
                offset = 0;
                omega = atof(lambda);
                raw[2][num] = (float)omega;
                i++;

            }

            if (i == 30)
            {
                i = 0;
                screens++;
            }
        }

        if (half == 0)
            num++;
        raw[1][num] = raw[2][num] = -1.0;


        /*  Number of points read through the keyboard.  */
        points = num - 1;
    }


    elim = 0;
    /*  These tests eliminate x and y values greater than or
```

```
        equal to one and values less than or equal to zero.  The
        values for x = 0 and x = 1 are automatically added later.
    */
    for (j = 1; j (= points; j++)
    {
        if ((raw[1][j] )= 1.0) || (raw[1][j] (= 0.0))
        {
            raw[1][j] = 100.0;
            elim++;
        }
        else
        {
            if ((raw[2][j] (= 0.0) || (raw[2][j] ) 1.0))
            {
                raw[1][j] = 100.0;
                elim++;
            }
        }
    }

    /*  This nested test eliminates duplicate x values.  If two
        points are given with the same x value, the first of the
        two is filtered out.
    */
    for (j = 1; j (= points; j++)
    {
        for (i = j; i (= points; i++)
        {
            if (j != i)
            {
                if (raw[1][j] != 100.0)
                {
                    if (raw[1][j] == raw[1][i])
                    {
                        elim++;
                        raw[1][j] = 100.0;
                    }
                }
            }
        }
    }
    done = 0;
    if (elim )= points)
    {
        ermess(5,
            " NONE OF THE",
            "  SPECIFIED",
            " EQUILIBRIUM",
            " LINE VALUES",
            "  ARE VALID");
        done = 1;
    }
    k = 0;
    while (!done)
    {
        float     small_x;
```

```
        int  best;

        small_x = 2.0;      /* an impossibly large x */

        /*   The x values are ordered in this for loop    */
        k++;
        for (j = 1; j (= points; j++)
        {
             if (raw[1][j] ( small_x)
             {
                  small_x = raw[1][j];
                  best = j;
             }

        }
        eql[1][k] = raw[1][best];
        eql[2][k] = raw[2][best];
        raw[1][best] = 3.0;

        if ((k + elim) )= points)
             done = 1;
    }
    /* The new number of points after elimination of bad points and
       the addition of the point (1,1).
    */
    points = k + 1;
    eql[1][points] = eql[2][points] = 1.0;
    eql[1][points+1] = eql[2][points+1] = -1.0;

    _setviewport(0.0, 0.0, 0.66, 1.0, -1, BLACK);
    _setworld(0.0, 0.0, 660.0, 1000.0);
    draw_dia();
    draw_eql();
}


no_avail()
{
    float    mx, my;
    int  bt;

    _setviewport(norcomx1, norcomy1, norcomx2, norcomy2, GREEN, BLUE);
    _setworld(0.0, 0.0, 1000.0, 1000.0);

    _setext(2,1,0,1);
    _settextclr(YELLOW, BLUE);
    attext(400.0, 700.0, "NOT");
    attext(160.0, 600.0, "AVAILABLE");
    attext(10.0, 500.0, "AT THIS TIME.");

    if (mouse_in_use)
    {
         _setext(1,1,0,1);
         _settextclr(L_GREEN, BLUE);
         attext(160.0, 250.0, "CLICK ANY");
         attext(10.0, 200.0, "BUTTON ON THE");
```

```
        attext(160.0, 150.0, "MOUSE TO");
        attext(160.0, 100.0, "CONTINUE.");
    }
    else
    {
        _setext(1,1,0,1);
        _settextclr(L_GREEN, BLUE);
        attext(10.0, 200.0, "  HIT ANY KEY");
        attext(10.0, 150.0, "  TO CONTINUE");
    }
    deltcur();

    sleep();

}


sleep()
{
    float    mx, my;
    int  bt;

    if (mouse_in_use)
    {
        for(;;)
        {
            readlocator(&mx, &my, &bt);
            if (CLICK)
                break;
        }
    }
    else
    {
        int  chr = _getch();
    }
}



getrvol()
{
    char     lambda[100];

    eqltype = RVOLATILE;

    _setviewport(norcomx1, norcomy1, norcomx2, norcomy2, GREEN, GREY);
    _setworld(0.0, 0.0, 1000.0, 1000.0);

    _setcolor(BLUE);
    clr();

    _setext(2,1,0,1);
    _settextclr(YELLOW, BLUE);
    attext(80.0, 800.0, "WHAT IS THE");
    attext(160.0, 600.0, "RELATIVE");
    attext(80.0, 500.0, "VOLATILITY?");
```

```
     _settextclr(WHITE, BLUE);
     strng(200.0, 300.0, lambda, 1, 10);
     alpha = atof(lambda);
     deltcur();

     relvol();
}

relvol()
{
     int  i;
     double    interval;

     points = 20;

     eql[1][0] = eql[2][0] = 0.0;

     /* The points on the equilibrium line are found   */
     for (i = 1; i <= points; i++)
     {
          interval = i/((float)points);

          eql[1][i] = interval;
          eql[2][i] = alpha/((1.0/interval) + alpha - 1.0);
     }
     _setviewport(0.0, 0.0, 0.66, 1.0, -1, BLACK);
     _setworld(0.0, 0.0, 660.0, 1000.0);
     draw_dia();
     draw_eql();

}


draw_eql()          /* Equilibrium line is drawn here  */
{
     int  i;

     if (eqltype)
     {
          _setcolor(RED);
          _movabs(diax(eql[1][0]), diay(eql[2][0]));

          for (i = 1; i <= points; i++)
          {
               _lnabs(diax(eql[1][i]), diay(eql[2][i]));
          }
     }
}

draw_brown_eql()    /* Brown equilibrium line is drawn here */
{
     int  i;

     if (eqltype)
     {
```

```
          _setcolor(BROWN);
          _movabs(diax(eql[1][0]), diay(eql[2][0]));

          for (i = 1; i (= points; i++)
          {
               _lnabs(diax(eql[1][i]), diay(eql[2][i]));
          }
     }
}

prtmenu()
{
     int option;

     do
     {

     _setviewport(norcomx1, norcomy1, norcomx2, norcomy2, GREEN, GREY);
     _setworld(0.0, 0.0, 1000.0, 1000.0);

          _setext(1,1,0,1);
          _settextclr(WHITE, RED);
          attext(TITLEX, TITLEY, "  DISPLAY  ");
          attext(TITLEX, STITLEY, "  OPTIONS  ");
          _settextclr(YELLOW, GREY);
          attext(TXTX, TXT1Y, "SAVE");
          attext(TXTX, STXT1Y, "DIAGRAM");
          attext(TXTX, TXT2Y, "RETRIEVE");
          attext(TXTX, STXT2Y, "DIAGRAM");
          attext(TXTX, TXT3Y, "SEND TO");
          attext(TXTX, STXT3Y, "PRINTER");
          attext(TXTX, TXT4Y, "PRINT w/");
          attext(TXTX, STXT4Y, "BKGD GRID");
          attext(TXTX, TXT5Y, "NO CHANGE");
          delcur();
          option = menubox(5, BLACK, MAGENTA);


          switch(option)
          {
          case 1:             /* Save Diagram          */
               savedia();
               break;
          case 2:             /* Retrieve Diagram */
               retrieve();
               break;
          case 3:             /* Send to Printer  */
               if (rorl("  IS THE PRINTER ATTACHED  ",
                    "    AND READY TO PRINT?    ",
                    "READY TO PRINT", "ABORT PRINT"))
               {
                    dotmatrix();
               }
               break;
          case 4:             /* Print w/ Grid    */
               if (rorl("  IS THE PRINTER ATTACHED  ",
```

```
                    "    AND READY TO PRINT?    ",
                    "READY TO PRINT", "ABORT PRINT"))
               {
                    grid = 1;
                    dotmatrix();
                    grid = 0;
               }
               break;
          case 5:             /* No Change        */
               option = 0;
               break;
          }
     } while (option);

}

savedia()
{
     char      lambda[15], picfile[15], datfile[15];
     int  j, i, good;
     FILE *fopen(), *fp;

     _setviewport(NORMENX1, NORMENY1, NORMENX2, NORMENY2, GREEN, GREY);
     _setworld(0.0, 0.0, 1000.0, 1000.0);

     do
     {
          _setcolor(GREY);
          clr();
          _setext(2,1,0,1);
          _settextclr(GREEN, GREY);
          attext(240.0, 700.0, "FILENAME:");
          _settextclr(L_GREEN, GREY);
          strng(280.0, 400.0, lambda, 1, 12);

          if (isdigit(lambda[0]))
          {
               _setcolor(GREY);
               clr();
               _settextclr(WHITE, RED);
               attext(240.0, 600.0, "INVALID ENTRY");
               good = 0;
          }


/*   Unless a drive has been specified, truncate the
     filename to eight characters.
*/
          if (lambda[1] != ':')
               lambda[8] = '\0';

/*   Copy the filename into the arrays 'picfile' and 'datfile' and
     give the filenames the extensions '.pic' and '.dat' respectively.
*/
          i = 0;
          while (((picfile[i] = datfile[i] = lambda[i]) != '.') &&
```

```
        ((picfile[i] = datfile[i] = lambda[i]) != '\0'))
    {
        i++;
    }
    picfile[i] = '.';
    datfile[i] = '.';
    i++;
    picfile[i] = 'p';
    datfile[i] = 'd';
    i++;
    picfile[i] = 'i';
    datfile[i] = 'a';
    i++;
    picfile[i] = 'c';
    datfile[i] = 't';
    i++;
    picfile[i] = datfile[i] = '\0';


    if ((fp = fopen(datfile, "r")) != NULL)
    {
        char epsilon[5];
        int  cont;

        do
        {
            _setcolor(GREY);
            clr();
            _setext(2,1,0,1);
            attext(80.0, 800.0, "A FILE WITH THAT NAME");
            attext(80.0, 700.0, "   ALREADY EXISTS.");
            _setext(1,1,0,1);
            attext(40.0,400.0,"DO YOU WISH TO OVERWRITE");
            attext(40.0,350.0,"THE OLD FILE (Y/N):");
            strng(880.0, 340.0, epsilon, 1, 3);

            switch (epsilon[0])
            {
            case 'y':
            case 'Y':
                good = 1;
                cont = 1;
                break;
            case 'n':
            case 'N':
                good = 0;
                cont = 1;
                break;
            default:
                cont = 0;
                break;
            }
        } while (!cont);
        fclose(fp);
    }
} while (!good);
```

```
    if ((fp = fopen(datfile, "w")) == NULL)
    {
        fprintf(stderr, "\n\n\tFailure to open file");
        fprintf(stderr, "\n\n\tPROGRAM TERMINATED\n\n");
        exit(0);
    }

    fprintf(fp, "%d\n",probtype);
    fprintf(fp, "%g\n",x_bott);
    fprintf(fp, "%g\n",x_feed);
    fprintf(fp, "%g\n",q);
    fprintf(fp, "%g\n",x_dist);
    fprintf(fp, "%g\n",opslope);
    fprintf(fp, "%g\n",rratio);
    fprintf(fp, "%d\n",nostages);

    fprintf(fp, "%d\n",eqltype);
    fprintf(fp, "%g\n",alpha);

    fprintf(fp, "%d\n",points);
    for (j = 0; j (= points; j++)
    {
        fprintf(fp, "%g\n",eql[1][j]);
        fprintf(fp, "%g\n",eql[2][j]);
    }

    fprintf(fp, "%d\n",operpts);
    for (j = 0; j (= operpts; j++)
    {
        fprintf(fp, "%g\n",opl[1][j]);
        fprintf(fp, "%g\n",opl[2][j]);
    }

    fprintf(fp, "%d\n", feedtray);
    fprintf(fp, "%d\n", optimal);
    fprintf(fp, "%d\n", top_to_bott);
    fprintf(fp, "%d\n", bott_to_top);

    fclose(fp);

    _setviewport(0.0, 0.0, 1.0, 1.0, -1, BLACK);
    _setworld(0.0, 0.0, 1000.0, 1000.0);
    draw_dia();
    draw_eql();
    draw_answer();

    gwrite(picfile);

}

int retrieve()
{
    char     line[15], lambda[15], picfile[15], datfile[15];
    int  ldigit, j, i, good, newdrive;
    FILE *fopen(), *fp;
```

```
    _setviewport(NORMENX1, NORMENY1, NORMENX2, NORMENY2, GREEN, GREY);
    _setworld(0.0, 0.0, 1000.0, 1000.0);

    do
    {
        ldigit = 0;
        _setcolor(GREY);
        clr();
        _setext(2,1,0,1);
        _settextclr(GREEN, GREY);
        attext(240.0, 700.0, "FILENAME:");
        _settextclr(L_GREEN, GREY);
        strng(280.0, 400.0, lambda, 1, 12);

        if (isdigit(lambda[0]))
        {
            _setcolor(GREY);
            clr();
            _settextclr(WHITE, RED);
            attext(240.0, 600.0, "INVALID ENTRY");
            ldigit = 1;
        }
    } while (ldigit);

/*  Unless a drive has been specified, truncate the
    filename to eight characters.
*/
    if (lambda[1] != ':')
        lambda[8] = '\0';

/*  Copy the filename into the arrays 'picfile' and 'datfile' and
    give the filenames the extensions '.pic' and '.dat' respectively.
*/
    i = 0;
    while (((picfile[i] = datfile[i] = lambda[i]) != '.') &&
        ((picfile[i] = datfile[i] = lambda[i]) != '\0'))
    {
        i++;
    }
    picfile[i] = '.';
    datfile[i] = '.';
    i++;
    picfile[i] = 'p';
    datfile[i] = 'd';
    i++;
    picfile[i] = 'i';
    datfile[i] = 'a';
    i++;
    picfile[i] = 'c';
    datfile[i] = 't';
    i++;
    picfile[i] = datfile[i] = '\0';


    while ((fp = fopen(datfile, "r")) == NULL)
```

```
{
     char eps[5];
     int  cont;

     do
     {
          _setcolor(GREY);
          clr();
          _setext(2,1,0,1);
          _settextclr(GREEN, GREY);
          attext(160.0, 800.0, "THE SPECIFIED FILE");
          attext(160.0, 700.0, "  DOES NOT EXIST.");
          _setext(1,1,0,1);
          attext(40.0,500.0,"DO YOU WISH TO SPECIFY A");
          attext(40.0,450.0,"DIFFERENT DRIVE (Y/N):");
          strng(920.0, 440.0, eps, 1, 3);

          switch (eps[0])
          {
          case 'y':
          case 'Y':
               newdrive = 1;
               cont = 1;
               break;
          case 'n':
          case 'N':
               newdrive = 0;
               cont = 1;
               break;
          default:
               cont = 0;
               break;
          }
     } while (!cont);

     if (newdrive)
     {
          char      pictmp[20], dattmp[20];

          do
          {
               _setcolor(GREY);
               clr();
               _setext(2,1,0,1);
               attext(120.0, 700.0, "ALTERNATE DISK DRIVE");
               _setext(1,1,0,1);
               attext(80.0,400.0,"WHICH DRIVE SHOULD BE");
               attext(80.0,350.0,"SEARCHED:");
               strng(520.0, 340.0, eps, 1, 2);

               if (isupper(eps[0]))
               {
                    char tmp = eps[0];
                    eps[0] = tolower(tmp);
               }
```

```
            } while ((eps[0] != 'a') && (eps[0] != 'c'));


            i = 0;
            while ((pictmp[i] = picfile[i]) != '\0')
                i++;
            i = 0;
            while ((dattmp[i] = datfile[i]) != '\0')
                i++;

            picfile[0] = datfile[0] = eps[0];
            picfile[1] = datfile[1] = ':';
            i = 0;
            while ((picfile[i+2] = pictmp[i]) != '\0')
                i++;

            i = 0;
            while ((datfile[i+2] = dattmp[i]) != '\0')
                i++;

        }
        else
            return(0);

    }

    fgets(line, 12, fp);
    probtype = atoi(line);
    fgets(line, 12, fp);
    x_bott = (float)atof(line);
    fgets(line, 12, fp);
    x_feed = (float)atof(line);
    fgets(line, 12, fp);
    q = (float)atof(line);
    fgets(line, 12, fp);
    x_dist = (float)atof(line);
    fgets(line, 12, fp);
    opslope = (float)atof(line);
    fgets(line, 12, fp);
    rratio = (float)atof(line);
    fgets(line, 12, fp);
    nostages = atoi(line);

    fgets(line, 12, fp);
    eqltype = atoi(line);
    fgets(line, 12, fp);
    alpha = (float)atof(line);
    fgets(line, 12, fp);
    points = atoi(line);

    for (j = 0; j <= points; j++)
    {
        fgets(line, 12, fp);
        eql[1][j] = (float)atof(line);
        fgets(line, 12, fp);
        eql[2][j] = (float)atof(line);
```

```
    }

    fgets(line, 12, fp);
    operpts = atoi(line);
    for (j = 0; j <= operpts; j++)
    {
        fgets(line, 12, fp);
        opl[1][j] = (float)atof(line);
        fgets(line, 12, fp);
        opl[2][j] = (float)atof(line);
    }

    fgets(line, 12, fp);
    feedtray = atoi(line);
    fgets(line, 12, fp);
    optimal = atoi(line);
    fgets(line, 12, fp);
    top_to_bott = atoi(line);
    fgets(line, 12, fp);
    bott_to_top = atoi(line);

    fclose(fp);

    _setviewport(0.0, 0.0, 1.0, 1.0, -1, BLACK);
    _setworld(0.0, 0.0, 1000.0, 1000.0);

    gread(picfile);

}


ermess(num, txt1, txt2, txt3, txt4, txt5, txt6, txt7, txt8)
int  num;
char *txt1, *txt2, *txt3, *txt4, *txt5, *txt6, *txt7, *txt8;
{
    float mx, my;
    int bt;

    _setviewport(norcomx1, norcomy1, norcomx2, norcomy2, GREEN, GREY);
    _setworld(0.0, 0.0, 1000.0, 1000.0);

    _setext(2,1,0,1);
    _settextclr(WHITE, RED);
    attext(40.0, 850.0, "   WARNING   ");
    _setext(1,1,0,1);
    _settextclr(YELLOW, GREY);
    if (num >= 1)
        attext(10.0, 700.0, txt1);
    if (num >= 2)
        attext(10.0, 650.0, txt2);
    if (num >= 3)
        attext(10.0, 600.0, txt3);
    if (num >= 4)
        attext(10.0, 550.0, txt4);
    if (num >= 5)
        attext(10.0, 500.0, txt5);
```

```
    if (num )= 6)
        attext(10.0, 450.0, txt6);
    if (num )= 7)
        attext(10.0, 400.0, txt7);
    if (num )= 8)
        attext(10.0, 350.0, txt8);

    _settextclr(L_GREEN, GREY);

    if (mouse_in_use)
    {
        attext(160.0, 250.0, "CLICK ANY");
        attext(10.0, 200.0, "BUTTON ON THE");
        attext(160.0, 150.0, "MOUSE TO");
        attext(160.0, 100.0, "CONTINUE.");
    }
    else
    {
        attext(10.0, 250.0, "   HIT ANY   ");
        attext(10.0, 200.0, " KEY ON THE  ");
        attext(10.0, 150.0, " KEYBOARD TO ");
        attext(10.0, 100.0, "  CONTINUE.  ");
    }

    deltcur();

    sleep();
}


/*
    This function returns an 'x' value of the intersection of two lines
    given a point and the slope of each.
*/

float psinterx(x1, y1, slope1, x2, y2, slope2)
double    x1, y1, slope1, x2, y2, slope2;
{
    float tmp1, tmp2, tmp3;

    tmp1 = (float)((slope2 * x2) - (slope1 * x1));
    tmp2 = (float)(y2 - y1);
    tmp3 = (float)(slope2 - slope1);

    return((float)((tmp1-tmp2)/tmp3));
}

float feedcond()
{
    int  option;

    _setviewport(NORMENX1, NORMENY1, NORMENX2, NORMENY2, GREEN, GREY);
    _setworld(0.0, 0.0, 1000.0, 1000.0);

    _setext(1,1,0,1);
    _settextclr(WHITE, RED);
```

```
    attext(TITLEX, STITLEY, "  FEED CONDITION  ");
    _settextclr(YELLOW, GREY);
    attext(TXTX, TXT1Y, "SATURATED VAPOR");
    attext(TXTX, TXT2Y, "SATURATED LIQUID");
    attext(TXTX, TXT3Y, "TWO PHASES");
    attext(TXTX, TXT4Y, "RETURN");
    deltcur();
    option = menubox(4, BLACK, MAGENTA);

    switch(option)
    {
    case 1:                 /* Saturated Vapor  */
        return(0.0);
    case 2:                 /* Saturated Liquid */
        return(1.0);
    case 3:                 /* Two Phases       */
        return(key_x("ENTER THE VALUE OF 'Q'",
                 "FOR THE FEED STREAM"));
    case 4:                 /* Return      */
        break;
    }
}



/*
    This function allows the choice of the Mouse or the Keyboard
    for input of numerical data.  A '1' is returned if the Mouse
    is chosen and a '0' if the keyboard is chosen.
*/

int mork()
{
    float    mx, my;
    int  bt;

   if (mouse_in_use)
   {
    _setviewport(NORMENX1, NORMENY1, NORMENX2, NORMENY2, GREEN, GREY);
    _setworld(0.0, 0.0, 1000.0, 1000.0);
    _setext(1,1,0,1);
    _settextclr(WHITE, RED);
    attext(50.0, 850.0, "   USE WHICH DEVICE FOR  ");
    attext(50.0, 800.0, " INPUT OF NUMERICAL DATA?");
    _setcolor(GREEN);
    _bar(10.0, 450.0, 350.0, 625.0);
    _bar(10.0, 150.0, 350.0, 325.0);
    _settextclr(GREY, GREEN);
    attext(50.0, 550.0, "RIGHT ");
    attext(50.0, 500.0, "BUTTON");
    attext(50.0, 250.0, " LEFT ");
    attext(50.0, 200.0, "BUTTON");
    _settextclr(GREEN, GREY);
    attext(450.0, 525.0, "MOUSE");
    attext(450.0, 225.0, "KEYBOARD");
```

```
     deltcur();
     for(;;)
     {
          readlocator(&mx, &my, &bt);
          if (RIGHT_BT)
               return(1);          /* MOUSE is chosen  */
          if (LEFT_BT)
               return(0);          /* keyboard is chosen    */
     }
    }
    else
     return(0);
}


int rorl(ttxt, sttxt, rtxt, ltxt)
char *ttxt, *sttxt, *rtxt, *ltxt;
{
     float     mx, my;
     int  bt;

     _setviewport(NORMENX1, NORMENY1, NORMENX2, NORMENY2, GREEN, GREY);
     _setworld(0.0, 0.0, 1000.0, 1000.0);
     _setext(1,1,0,1);
     _settextclr(WHITE, RED);
     attext(50.0, 850.0, ttxt);
     attext(50.0, 800.0, sttxt);
     _setcolor(GREEN);
     _bar(10.0, 450.0, 350.0, 625.0);
     _bar(10.0, 150.0, 350.0, 325.0);
     _settextclr(GREY, GREEN);
     if (mouse_in_use)
     {
          attext(50.0, 550.0, "RIGHT");
          attext(50.0, 500.0, "BUTTON");
          attext(50.0, 250.0, " LEFT");
          attext(50.0, 200.0, "BUTTON");
     }
     else
     {
          attext(50.0, 550.0, "UP");
          attext(50.0, 500.0, "ARROW");
          attext(50.0, 250.0, "DOWN");
          attext(50.0, 200.0, "ARROW");
     }
     _settextclr(GREEN, GREY);
     attext(450.0, 525.0, rtxt);
     attext(450.0, 225.0, ltxt);
     deltcur();
     if (mouse_in_use)
     {
          for(;;)
          {
               readlocator(&mx, &my, &bt);
               if (RIGHT_BT)
                    return(1);
```

```
                if (LEFT_BT)
                    return(0);
            }
        }
        else
        {
            int chr;
            for (;;)
            {
                chr = _getch();
                if (chr == 328)
                    return(1);
                if (chr == 336)
                    return(0);
            }
        }

}

float key_x(txt, stxt)
char *txt, *stxt;
{
    char lambda[12];
    double  omega;

    _setviewport(NORMENX1, NORMENY1, NORMENX2, NORMENY2, GREEN, BLUE);
    _setworld(0.0, 0.0, 1000.0, 1000.0);
    _setext(1,1,0,1);
    _settextclr(YELLOW, BLUE);
    attext(50.0, 650.0, txt);
    attext(50.0, 600.0, stxt);
    _settextclr(WHITE, BLUE);
    strng(200.0, 300.0, lambda, 1, 10);

    omega = atof(lambda);

    return((float)omega);
}

normcoor()
{
    norcomx1 = COMX1/1000.0;
    norcomy1 = COMY1/1000.0;

    norcomx2 = COMX2/1000.0;
    norcomy2 = COMY2/1000.0;

    nordiax1 = DIAX1/1000.0;
    nordiay1 = DIAY1/1000.0;

    nordiax2 = DIAX2/1000.0;
    nordiay2 = DIAY2/1000.0;
}


/*
```

```
     This is an "intelligent" getch() command that adds 256 to the ASCII
     codes of control characters
*/

ctl_chr()
{
     int chr = cget();

     if (chr != 0)
          return(chr);

     chr = cget();
     return(256+chr);
}


/*
     The next two routines allow keyboard input to be displayed on
     the color screen.  Original code by A. Skjellum.
*/


strng(tcurx,tcury,string,imin,imax)     /* sets text cursor position */
double tcurx,tcury;           /* then goes to _strng()     */
char *string;
int imin,imax;
{
     _movtcurabs(tcurx,tcury);
     return(_strng(string,imin,imax));
}

_strng(string,imin,imax) /* displays text on color screen and  */
char *string;            /* allows backspacing during input    */
int imin,imax;
{
     float o_curx,o_cury;
     float curx,cury;
     float delta;
     int dflag = 0;
     int y,x;
     int i = 0;
     int done = 0;

     if(imax ( imin)
     {
          int temp;
          temp = imax;
          imax = imin;
          imin = temp;   /* swap for robust behavior */
     }

     inqtcur(&o_curx,&o_cury);
     curx = o_curx;
     cury = o_cury;
     delta = 0.0;
```

```
    while(!done)
    {
        int chr;
        chr = cget();  /* get a character */
        switch(chr)
        {
            case '\b':     /* backspace     */
            case 0x07f:
                if(i)  /* decrement count */
                {
                    i--;
                    o_curx = curx;
                    curx -= delta;
                    _movtcurabs(curx,cury);
                    text(" ");
                    _movtcurabs(curx,cury);
                }
                break;

            case '\n':     /* RETURN or carriage feed    */
            case '\r':
                if(i )= imin) done = 1;
                break;

            default:  /* text to be displayed       */
                if(i ( imax)
                {
                    char str[2];
                    if(chr ( 32) break;
                    string[i++] = chr;
                    str[0] = chr;
                    str[1] = '\0';
                    text(str);
                    o_curx = curx;
                    o_cury = cury;
                    inqtcur(&curx,&cury);
                    if(dflag == 0)
                    {
                        delta = curx - o_curx;
                        dflag = 1;
                    }
                }
                break;
        }

    } /* end while(!done) */

    string[i] = '\0';   /* eos */

    return(i);
}
```

```
/*
     Last Revision
     5/2/86

     MC3A.C

     Contains the following procedures:

          Curr_val()
          Trial()
          Draw_answer()
          Copline().
*/

#include (math.h)
#include (color.h)
#include (stdio.h)
#include (mcext.h)


int copline(mode)
int mode;
{
     if (probtype == STRIP)
     {
          opl[1][1] = x_bott;
          opl[2][1] = x_bott;
          if (q == 1.0)
          {
               opl[1][2] = x_feed;
               opl[2][2] = opl[2][1] + opslope * (x_feed - x_bott);
          }
          if (q == 0.0)
          {
               opl[2][2] = x_feed;
               opl[1][2] = opl[1][1] + (opl[2][2]-opl[2][1])/opslope;
          }
          if ((q != 0.0) && (q != 1.0))
          {
               opl[1][2] = psinterx(opl[1][1], opl[2][1],opslope,
                              x_feed, x_feed, q/(q-1));
               opl[2][2] = opl[2][1] + opslope*(opl[1][2]-opl[1][1]);
          }
          if (mode == 1)
          {
               _setcolor(GREEN);
               _movabs(diax(x_bott), diay(x_bott));
               _lnabs(diax(opl[1][2]), diay(opl[2][2]));
               _setcolor(BROWN);
               _lnabs(diax(x_feed), diay(x_feed));
          }
          if ((opl[1][2] ( 0.0) !! (opl[1][2] ) 1.0))
               return(0);
          if (opl[2][2] )= equil_y(opl[1][2]))
               return(0);
```

```
        return(1);
    }
    if (probtype == RECTIFY)
    {
        opl[1][2] = x_dist;
        opl[2][2] = x_dist;
        if (q == 1.0)
        {
            opl[1][1] = x_feed;
            opl[2][1] = opl[2][2] + opslope * (x_feed - x_dist);
        }
        if (q == 0.0)
        {
            opl[2][1] = x_feed;
            opl[1][1] = opl[1][2] + (opl[2][1]-opl[2][2])/opslope;
        }
        if ((q != 0.0) && (q != 1.0))
        {
            opl[1][1] = psinterx(opl[1][2], opl[2][2],opslope,
                        x_feed, x_feed, q/(q-1));
            opl[2][1] = opl[2][2] + opslope*(opl[1][1]-opl[1][2]);
        }

        if (mode == 1)
        {
            _setcolor(GREEN);
            _movabs(diax(x_dist), diay(x_dist));
            _lnabs(diax(opl[1][1]), diay(opl[2][1]));
            _setcolor(BROWN);
            _lnabs(diax(x_feed), diay(x_feed));
        }
        if ((opl[1][1] < 0.0) || (opl[1][1] > 1.0))
            return(0);
        if (opl[2][1] >= equil_y(opl[1][1]))
            return(0);

        return(1);
    }
    if (probtype == SINGLE)
    {
        float    topslope;

        topslope = 1.0/(1.0 + rratio);

        opl[1][1] = x_bott;
        opl[2][1] = x_bott;
        opl[1][3] = x_dist;
        opl[2][3] = x_dist;
        if (q == 1.0)
        {
            opl[1][2] = x_feed;
            opl[2][2] = opl[2][3] - topslope * (x_dist - x_feed);
        }
        if (q == 0.0)
        {
            opl[2][2] = x_feed;
```

```
        opl[1][2] = opl[1][3]-(opl[2][3]-opl[2][2])/topslope;
    }
    if ((q != 0.0) && (q != 1.0))
    {
        opl[1][2] = psinterx(opl[1][3], opl[2][3],topslope,
                    x_feed, x_feed, q/(q-1));
        opl[2][2] = opl[2][3] -topslope*(opl[1][3]-opl[1][2]);
    }
    if (mode == 1)
    {
        _setcolor(GREEN);
        _movabs(diax(opl[1][1]), diay(opl[2][1]));
        _lnabs(diax(opl[1][2]), diay(opl[2][2]));
        _lnabs(diax(opl[1][3]), diay(opl[2][3]));
        _setcolor(BROWN);
        _movabs(diax(opl[1][2]), diay(opl[2][2]));
        _lnabs(diax(x_feed), diay(x_feed));
    }
    if ((opl[1][2] ( 0.0) || (opl[1][2] ) 1.0))
        return(0);
    if (opl[2][2] )= equil_y(opl[1][2]))
        return(0);

    if (!optimal)
    {
        if (top_to_bott)
        {
            int  i;
            float    hi_limit, lo_limit, mid;
            float    op_y, diff;

            lo_limit = 0.0;
            hi_limit = x_dist;
            diff = 0.0;
            do
            {
                if (diff ( 0.0)
                    lo_limit = mid;
                if (diff ) 0.0)
                    hi_limit = mid;

                mid = (hi_limit + lo_limit)/2.0;
                op_y = x_dist-topslope*(x_dist-mid);

                diff = equil_y(mid) - op_y;

            } while (fabs(diff) )= 0.001);

            ofx = opl[1][2];
            ofy = opl[2][2];
            opl[1][2] = mid;
            opl[2][2] = equil_y(mid);
        }
        if (bott_to_top)
        {
            int  i;
```

```
                    float     hi_limit, lo_limit, mid, bottslope;
                    float     op_y, diff;

                    bottslope = (opl[2][2]-x_bott)
                              /(opl[1][2]-x_bott);

                    lo_limit = x_bott;
                    hi_limit = 1.0;
                    diff = 0.0;
                    do
                    {
                         if (diff < 0.0)
                              hi_limit = mid;
                         if (diff > 0.0)
                              lo_limit = mid;

                         mid = (hi_limit + lo_limit)/2.0;
                         op_y = x_bott-bottslope*(x_bott-mid);

                         diff = equil_y(mid) - op_y;

                    } while (fabs(diff) >= 0.001);

                    ofx = opl[1][2];
                    ofy = opl[2][2];
                    opl[1][2] = mid;
                    opl[2][2] = equil_y(mid);
               }
          }
          return(1);
     }
}

draw_answer()
{
     int  i;
     float     cur_x, cur_y;
     float     tmp_x, tmp_y;

     if (probtype == STRIP)
     {
          if ( (x_bott != -1.0) &&
               (x_feed != -1.0) &&
               (q != -1.0) &&
               (opslope != -1.0) &&
               (nostages != 0))
          {
               _setcolor(GREEN);
               _movabs(diax(x_bott), diay(x_bott));
               _lnabs(diax(opl[1][2]), diay(opl[2][2]));
               _setcolor(BROWN);
               _lnabs(diax(x_feed), diay(x_feed));
               cur_x = opl[1][2];
               cur_y = opl[2][2];
               _setcolor(WHITE);
               _movabs(diax(cur_x), diay(cur_y));
```

```
                        /* Feed Location */
                for (i = 1; i (= nostages; i++)
                {
                    cur_x = equil_x(cur_y);
                    _lnabs(diax(cur_x), diay(cur_y));
                    cur_y = oper_y(cur_x);
                    _lnabs(diax(cur_x), diay(cur_y));
                }
            }
        }
        if (probtype == RECTIFY)
        {
            if ( (x_dist != -1.0) &&
                (x_feed != -1.0) &&
                (q != -1.0) &&
                (opslope != -1.0) &&
                (nostages != 0))
            {
                _setcolor(GREEN);
                _movabs(diax(x_dist), diay(x_dist));
                _lnabs(diax(opl[1][1]), diay(opl[2][1]));
                _setcolor(BROWN);
                _lnabs(diax(x_feed), diay(x_feed));
                cur_x = opl[1][1];
                cur_y = opl[2][1];
                _setcolor(WHITE);
                _movabs(diax(cur_x), diay(cur_y));
                        /* Feed Location */
                for (i = 1; i (= nostages; i++)
                {
                    cur_y = equil_y(cur_x);
                    _lnabs(diax(cur_x), diay(cur_y));
                    cur_x = oper_x(cur_y);
                    _lnabs(diax(cur_x), diay(cur_y));
                }
            }
        }
        if ((probtype == SINGLE) && (optimal))
        {
            if ( (x_bott != -1.0) &&
                (x_feed != -1.0) &&
                (q != -1.0) &&
                (x_dist != -1.0) &&
                (rratio != -1.0) &&
                (nostages != 0))
            {
                _setcolor(GREEN);
                _movabs(diax(opl[1][1]), diay(opl[2][1]));
                _lnabs(diax(opl[1][2]), diay(opl[2][2]));
                _lnabs(diax(opl[1][3]), diay(opl[2][3]));
                _setcolor(BROWN);
                _movabs(diax(opl[1][2]), diay(opl[2][2]));
                _lnabs(diax(x_feed), diay(x_feed));
                cur_x = opl[1][1];
                cur_y = opl[2][1];
                _setcolor(WHITE);
```

```
            _movabs(diax(cur_x), diay(cur_y));
                        /* Feed Location */
            for (i = 1; i (= nostages; i++)
            {
                cur_y = equil_y(cur_x);
                _lnabs(diax(cur_x), diay(cur_y));
                cur_x = oper_x(cur_y);
                _lnabs(diax(cur_x), diay(cur_y));
            }
        }
    }
    if ((probtype == SINGLE) && (!optimal))
    {
        if ( (x_bott != -1.0) &&
            (x_feed != -1.0) &&
            (q != -1.0) &&
            (x_dist != -1.0) &&
            (rratio != -1.0) &&
            (nostages != 0))
        {
            tmp_x = opl[1][2];
            tmp_y = opl[2][2];

            if (top_to_bott)
            {
                _setcolor(GREEN);
                _movabs(diax(opl[1][3]), diay(opl[2][3]));
                _lnabs(diax(opl[1][2]), diay(opl[2][2]));
                _movabs(diax(ofx), diay(ofy));
                _lnabs(diax(opl[1][1]), diay(opl[2][1]));
                _setcolor(BROWN);
                _movabs(diax(ofx), diay(ofy));
                _lnabs(diax(x_feed), diay(x_feed));

                cur_x = opl[1][3];
                cur_y = opl[2][3];
                _setcolor(WHITE);
                _movabs(diax(cur_x), diay(cur_y));

                for (i = 1; i (= nostages; i++)
                {
                    cur_x = equil_x(cur_y);
                    _lnabs(diax(cur_x), diay(cur_y));

                    if (i == feedtray)
                    {
                        opl[1][2] = ofx;
                        opl[2][2] = ofy;
                    }
                    cur_y = oper_y(cur_x);
                    _lnabs(diax(cur_x),,diay(cur_y));
                }
            }
            if (bott_to_top)
            {
                _setcolor(GREEN);
```

```
                    _movabs(diax(opl[1][1]), diay(opl[2][1]));
                    _lnabs(diax(opl[1][2]), diay(opl[2][2]));
                    _movabs(diax(ofx), diay(ofy));
                    _lnabs(diax(opl[1][3]), diay(opl[2][3]));
                    _setcolor(BROWN);
                    _movabs(diax(ofx), diay(ofy));
                    _lnabs(diax(x_feed), diay(x_feed));

                    cur_x = opl[1][1];
                    cur_y = opl[2][1];
                    _setcolor(WHITE);
                    _movabs(diax(cur_x), diay(cur_y));

                    for (i = 1; i <= nostages; i++)
                    {
                         cur_y = equil_y(cur_x);
                         _lnabs(diax(cur_x), diay(cur_y));
                         if (i == feedtray)
                         {
                              opl[1][2] = ofx;
                              opl[2][2] = ofy;
                         }
                         cur_x = oper_x(cur_y);
                         _lnabs(diax(cur_x), diay(cur_y));
                    }
               }
               opl[1][2] = tmp_x;
               opl[2][2] = tmp_y;
          }
     }

}

float trial()
{
     int  i;
     float     cur_x, cur_y, overshot;
     float     tmp_x, tmp_y;

     _setxor(1);

     if (probtype == STRIP)
     {
          _setcolor(GREEN);
          _movabs(diax(x_bott), diay(x_bott));
          _lnabs(diax(opl[1][2]), diay(opl[2][2]));
          _setcolor(BROWN);
          _lnabs(diax(x_feed), diay(x_feed));

          cur_x = opl[1][2];
          cur_y = opl[2][2];
          _setcolor(WHITE);
          _movabs(diax(cur_x), diay(cur_y)); /* Feed Location */

          for (i = 1; i <= nostages; i++)
          {
```

```
            cur_x = equil_x(cur_y);
            _rlnabs(diax(cur_x), diay(cur_y));
            _movabs(diax(cur_x), diay(cur_y));
            cur_y = oper_y(cur_x);
            _rlnabs(diax(cur_x), diay(cur_y));
            _movabs(diax(cur_x), diay(cur_y));
        }
        delln();
        overshot = x_bott - cur_x;

        _setcolor(GREEN);
        _movabs(diax(x_bott), diay(x_bott));
        _lnabs(diax(opl[1][2]), diay(opl[2][2]));
        _setcolor(BROWN);
        _lnabs(diax(x_feed), diay(x_feed));
    }
    if (probtype == RECTIFY)
    {
        _setcolor(GREEN);
        _movabs(diax(x_dist), diay(x_dist));
        _lnabs(diax(opl[1][1]), diay(opl[2][1]));
        _setcolor(BROWN);
        _lnabs(diax(x_feed), diay(x_feed));

        cur_x = opl[1][1];
        cur_y = opl[2][1];
        _setcolor(WHITE);
        _movabs(diax(cur_x), diay(cur_y)); /* Feed Location */

        for (i = 1; i (= nostages; i++)
        {
            cur_y = equil_y(cur_x);
            _rlnabs(diax(cur_x), diay(cur_y));
            _movabs(diax(cur_x), diay(cur_y));
            cur_x = oper_x(cur_y);
            _rlnabs(diax(cur_x), diay(cur_y));
            _movabs(diax(cur_x), diay(cur_y));
        }
        delln();
        overshot = cur_x - x_dist;

        _setcolor(GREEN);
        _movabs(diax(x_dist), diay(x_dist));
        _lnabs(diax(opl[1][1]), diay(opl[2][1]));
        _setcolor(BROWN);
        _lnabs(diax(x_feed), diay(x_feed));
    }
    if ((probtype == SINGLE) && (optimal))
    {
        _setcolor(GREEN);
        _movabs(diax(opl[1][1]), diay(opl[2][1]));
        _lnabs(diax(opl[1][2]), diay(opl[2][2]));
        _lnabs(diax(opl[1][3]), diay(opl[2][3]));
        _setcolor(BROWN);
        _movabs(diax(opl[1][2]), diay(opl[2][2]));
        _lnabs(diax(x_feed), diay(x_feed));
```

```
        cur_x = opl[1][1];
        cur_y = opl[2][1];
        _setcolor(WHITE);
        _movabs(diax(cur_x), diay(cur_y)); /* Feed Location */

        for (i = 1; i (= nostages; i++)
        {
             cur_y = equil_y(cur_x);
             _rlnabs(diax(cur_x), diay(cur_y));
             _movabs(diax(cur_x), diay(cur_y));
             cur_x = oper_x(cur_y);
             _rlnabs(diax(cur_x), diay(cur_y));
             _movabs(diax(cur_x), diay(cur_y));
        }
        delln();
        overshot = cur_x - x_dist;

        _setcolor(GREEN);
        _movabs(diax(opl[1][1]), diay(opl[2][1]));
        _lnabs(diax(opl[1][2]), diay(opl[2][2]));
        _lnabs(diax(opl[1][3]), diay(opl[2][3]));
        _setcolor(BROWN);
        _movabs(diax(opl[1][2]), diay(opl[2][2]));
        _lnabs(diax(x_feed), diay(x_feed));
    }
    if ((probtype == SINGLE) && (!optimal))
    {
        tmp_x = opl[1][2];
        tmp_y = opl[2][2];

        if (top_to_bott)
        {
             _setcolor(GREEN);
             _movabs(diax(opl[1][3]), diay(opl[2][3]));
             _lnabs(diax(opl[1][2]), diay(opl[2][2]));
             _movabs(diax(ofx), diay(ofy));
             _lnabs(diax(opl[1][1]), diay(opl[2][1]));
             _setcolor(BROWN);
             _movabs(diax(ofx), diay(ofy));
             _lnabs(diax(x_feed), diay(x_feed));

             cur_x = opl[1][3];
             cur_y = opl[2][3];
             _setcolor(WHITE);
             _movabs(diax(cur_x), diay(cur_y));

             for (i = 1; i (= nostages; i++)
             {
                  cur_x = equil_x(cur_y);
                  _rlnabs(diax(cur_x), diay(cur_y));
                  _movabs(diax(cur_x), diay(cur_y));
                  if (i == feedtray)
                  {
                       if (cur_x ) ofx)
                       {
```

```
                    overshot = -1.0;
                    break;
               }
               opl[1][2] = ofx;
               opl[2][2] = ofy;
          }
          cur_y = oper_y(cur_x);
          _rlnabs(diax(cur_x), diay(cur_y));
          _movabs(diax(cur_x), diay(cur_y));

          overshot = cur_x - x_bott;
     }
     delln();

     _setcolor(GREEN);
     _movabs(diax(opl[1][3]), diay(opl[2][3]));
     _lnabs(diax(opl[1][2]), diay(opl[2][2]));
     _movabs(diax(ofx), diay(ofy));
     _lnabs(diax(opl[1][1]), diay(opl[2][1]));
     _setcolor(BROWN);
     _movabs(diax(ofx), diay(ofy));
     _lnabs(diax(x_feed), diay(x_feed));
}
if (bott_to_top)
{
     _setcolor(GREEN);
     _movabs(diax(opl[1][1]), diay(opl[2][1]));
     _lnabs(diax(opl[1][2]), diay(opl[2][2]));
     _movabs(diax(ofx), diay(ofy));
     _lnabs(diax(opl[1][3]), diay(opl[2][3]));
     _setcolor(BROWN);
     _movabs(diax(ofx), diay(ofy));
     _lnabs(diax(x_feed), diay(x_feed));

     cur_x = opl[1][1];
     cur_y = opl[2][1];
     _setcolor(WHITE);
     _movabs(diax(cur_x), diay(cur_y));

     for (i = 1; i <= nostages; i++)
     {
          cur_y = equil_y(cur_x);
          _rlnabs(diax(cur_x), diay(cur_y));
          _movabs(diax(cur_x), diay(cur_y));
          if (i == feedtray)
          {
               if (cur_y < ofy)
               {
                    overshot = -1.0;
                    break;
               }
               opl[1][2] = ofx;
               opl[2][2] = ofy;
          }
          cur_x = oper_x(cur_y);
          _rlnabs(diax(cur_x), diay(cur_y));
```

```
                    _movabs(diax(cur_x), diay(cur_y));

                    overshot = cur_x - x_dist;
                }
                delln();

                _setcolor(GREEN);
                _movabs(diax(opl[1][1]), diay(opl[2][1]));
                _lnabs(diax(opl[1][2]), diay(opl[2][2]));
                _movabs(diax(ofx), diay(ofy));
                _lnabs(diax(opl[1][3]), diay(opl[2][3]));
                _setcolor(BROWN);
                _movabs(diax(ofx), diay(ofy));
                _lnabs(diax(x_feed), diay(x_feed));
            }

            opl[1][2] = tmp_x;
            opl[2][2] = tmp_y;
        }
        _setxor(0);

        return(overshot);
}

int curr_val(txt)
char *txt;
{
        int tmp;

        _setviewport(NORMENX1, NORMENY1, NORMENX2, NORMENY2, GREEN, GREY);
        _setworld(0.0, 0.0, 1000.0, 1000.0);
        _setext(1,1,0,1);
        _settextclr(WHITE, RED);
        attext(TITLEX,TITLEY," THE CURRENT VALUE IS ");
        _settextclr(YELLOW, GREY);
        attext(200.0, STITLEY, txt);
        attext(TXTX, TXT1Y, "CHANGE VALUE");
        attext(TXTX,TXT2Y, "LEAVE UNSPECIFIED");
        attext(TXTX, TXT3Y, "NO CHANGE");
        deltcur();
        tmp = menubox(3, BLACK, MAGENTA);

        return(tmp);
}
```

```
/*
     Last Revision
     5/2/86

     MC3B.C

     Contains the following procedures:

          Singlecalc()
          Singlecol()
*/

#include (math.h)
#include (color.h)
#include (stdio.h)
#include (mcext.h)


singlecol()
{
     char lambda[35];
     int  option,
          cond_option,
          values;

     probtype = SINGLE;

     do
     {
     _setviewport(norcomx1, norcomy1, norcomx2, norcomy2, GREEN, GREY);
     _setworld(0.0, 0.0, 1000.0, 1000.0);

          _setext(1,1,0,1);
          _settextclr(WHITE, BLUE);
          attext(TITLEX, TITLEY, "SINGLE FEED");
          attext(TITLEX, STITLEY,"   COLUMN  ");
          _settextclr(YELLOW, GREY);
          attext(TXTX, TRI1Y, "DISTILLATE");
          attext(TXTX, STRI1Y, "COMPOSIT'N");
          attext(TXTX, TRI2Y, "FEED");
          attext(TXTX, STRI2Y, "COMPOSIT'N");
          attext(TXTX, TRI3Y, "BOTTOMS");
          attext(TXTX, STRI3Y, "COMPOSIT'N");
          attext(TXTX, TRI4Y, "REFLUX");
          attext(TXTX, STRI4Y, "RATIO");
          attext(TXTX, TRI5Y, "NUMBER OF");
          attext(TXTX, STRI5Y, "STAGES");
          _settextclr(GREEN, GREY);
          attext(TXTX, TXT6Y, "BEGIN");
          attext(TXTX, STXT6Y, "CALCULAT'N");
          attext(TXTX, TXT7Y, "CHANGE");
          attext(TXTX, STXT7Y, "CONDITIONS");
          attext(TXTX, TXT8Y, "RETURN");

          if (x_dist != -1.0)
          {
```

```
        _settextclr(WHITE, GREY);
        sprintf(lambda, "  %.3g", x_dist);
        attext(TXTX, VTRI1Y, lambda);
        _settextclr(L_BLUE, GREY);
        attext(0.0, TXT1Y, "*");
    }
    else
    {
        _settextclr(WHITE, GREY);
        attext(TXTX, VTRI1Y, " _____");
    }
    if (x_feed != -1.0)
    {
        _settextclr(WHITE, GREY);
        sprintf(lambda, "  %.3g", x_feed);
        attext(TXTX, VTRI2Y, lambda);
        _settextclr(L_BLUE, GREY);
        attext(0.0, TXT2Y, "*");
    }
    else
    {
        _settextclr(WHITE, GREY);
        attext(TXTX, VTRI2Y, " _____");
    }
    if (x_bott != -1.0)
    {
        _settextclr(WHITE, GREY);
        sprintf(lambda, "  %.3g", x_bott);
        attext(TXTX, VTRI3Y, lambda);
        _settextclr(L_BLUE, GREY);
        attext(0.0, TXT3Y, "*");
    }
    else
    {
        _settextclr(WHITE, GREY);
        attext(TXTX, VTRI3Y, " _____");
    }
    if (rratio != -1.0)
    {
        _settextclr(WHITE, GREY);
        sprintf(lambda, "  %.3g", rratio);
        attext(TXTX, VTRI4Y, lambda);
        _settextclr(L_BLUE, GREY);
        attext(0.0, TXT4Y, "*");
    }
    else
    {
        _settextclr(WHITE, GREY);
        attext(TXTX, VTRI4Y, " _____");
    }
    if (nostages != 0)
    {
        _settextclr(WHITE, GREY);
        sprintf(lambda, "  %d", nostages);
        attext(TXTX, VTRI5Y, lambda);
        _settextclr(L_BLUE, GREY);
```

```
        attext(0.0, TXT5Y, "*");
    }
    else
    {
        _settextclr(WHITE, GREY);
        attext(TXTX, VTRI5Y, " _____");
    }
    deltcur();

    option = menubox(8, BLACK, MAGENTA);

    switch(option)
    {
    case 1:                 /* Distillate Comp. */
        if (x_dist != -1.0)
        {
            sprintf(lambda, "%.3g", x_dist);
            option = curr_val(lambda);

            if (option == 2)
            {
                x_dist = -1.0;
                break;
            }
            if (option == 3)
                break;
        }
        if (mark())
            x_dist = mouse_x();
        else
            x_dist = key_x("MOLE FRACTION OF LIGHT",
                    "COMPONENT IN DISTILLATE");
        break;
    case 2:                 /* Feed Comp.       */
        if (x_feed != -1.0)
        {
            sprintf(lambda, "%.3g", x_feed);
            option = curr_val(lambda);

            if (option == 2)
            {
                x_feed = -1.0;
                break;
            }
            if (option == 3)
                break;
        }
        if (mark())
            x_feed = mouse_x();
        else
            x_feed = key_x("MOLE FRACTION OF LIGHT",
                    "COMPONENT IN FEED");
        break;
    case 3:                 /* Bottoms Composition   */
        if (x_bott != -1.0)
        {
```

```
                sprintf(lambda, "%.3g", x_bott);
                option = curr_val(lambda);

                if (option == 2)
                {
                    x_bott = -1.0;
                    break;
                }
                if (option == 3)
                    break;
            }
            if (mork())
                x_bott = mouse_x();
            else
                x_bott = key_x("MOLE FRACTION OF LIGHT",
                        "COMPONENT IN BOTTOMS");
            break;
        case 4:                 /* Reflux Ratio          */
            if (rratio != -1.0)
            {
                sprintf(lambda,"%.3g [DIST/REFLUX]",rratio);
                option = curr_val(lambda);

                if (option == 2)
                {
                    rratio = -1.0;
                    break;
                }
                if (option == 3)
                    break;
            }
    _setviewport(norcomx1, norcomy1, norcomx2, norcomy2, GREEN, GREY);
    _setworld(0.0, 0.0, 1000.0, 1000.0);
            _setext(1,1,0,1);
            _settextclr(WHITE, BLUE);
            attext(TITLEX, TITLEY,  "   REFLUX   ");
            attext(TITLEX, STITLEY, "   RATIO    ");
            _settextclr(YELLOW, GREY);
            attext(TXTX, TXT1Y,  "EXTERNAL");
            attext(TXTX, STXT1Y, " (D/L)");
            attext(TXTX, TXT2Y,  "INTERNAL");
            attext(TXTX, STXT2Y, " (L/V)");
            deltcur();
            option = menubox(2, BLACK, MAGENTA);

            if (option == 1)
                rratio = key_x("EXTERNAL REFLUX RATIO",
                    "(DISTILL / REFLUX  FLOWS)");
            else
            {
                float    tmp1;

                tmp1 = key_x("INTERNAL REFLUX RATIO",
                    "(LIQUID / VAPOR  FLOWS)");
                rratio = (1.0/tmp1) - 1.0;
            }
```

```
            break;
        case 5:                 /* Number of Stages      */
            if (nostages != 0)
            {
                sprintf(lambda, "%d STAGES", nostages);
                option = curr_val(lambda);

                if (option == 2)
                {
                    nostages = 0;
                    break;
                }
                if (option == 3)
                    break;
            }
            nostages = key_x("NUMBER OF THEORETICAL",
                    "EQUILIBRIUM STAGES");
            break;
        case 6:                 /* Begin Calculations    */
            singlecalc();
            break;
        case 7:
            draw_cond(1);  /* draw in "menu" color scheme     */
            cond_option = alter_cond();
            if (cond_option == 1)    /* optimal feed tray     */
            {
                feed_loc();
            }
            if (cond_option == 2)    /* feed condition   */
            {
                q = feedcond();
            }
            if (cond_option == 3)    /* equilibrium      */
            {
                eqlmenu();
            }
            draw_cond(0);  /* draw in display color scheme    */
            break;
        case 8:                 /* No Change        */
            option = 0;
            break;
        }
    } while (option);

}

int singlecalc()
{
    float     cur_x, cur_y, mx, my;
    int  specified, solvfor, bt, i;
    float     tmp_x, tmp_y;

    specified = 0;

    if (x_dist == -1.0)
        solvfor = 1;
```

```
    else
        specified++;

    if (x_feed == -1.0)
        solvfor = 2;
    else
        specified++;

    if (x_bott == -1.0)
        solvfor = 3;
    else
        specified++;

    if (rratio == -1.0)
        solvfor = 4;
    else
        specified++;

    if (nostages == 0)
        solvfor = 5;
    else
        specified++;

    if (!eqltype)
    {
        ermess(5,
            "     AN",
            " EQUILIBRIUM",
            "  LINE MUST",
            "   FIRST BE",
            "  SPECIFIED.");
        return(0);
    }

    if (specified (= 3)      /* under specified problem    */
    {
        ermess(5,
            " ALTEAST FOUR",
            "   OF THE",
            "  VARIABLES",
            "   MUST BE",
            "  SPECIFIED.");
        return(0);
    }

    if (specified )= 5)      /* over specified problem     */
    {
        ermess(5,
            " ATLEAST ONE",
            "   OF THE",
            "  VARIABLES",
            "   MUST BE",
            " UNSPECIFIED.");
        return(0);
    }
```

```
    if ((x_dist (= x_feed) && (x_dist != -1.0))
    {
         ermess(7,
              " COMPOSITION",
              "OF THE LIGHT",
              "COMPONENT IN",
              "FEED MUST BE",
              "  LESS THAN",
              " THAT OF THE",
              " DISTILLATE.");
         return(0);
    }
    if ((x_feed (= x_bott) && (x_feed != -1.0))
    {
         ermess(7,
              " COMPOSITION",
              "OF THE LIGHT",
              "COMPONENT IN",
              "FEED MUST BE",
              "GREATER THAN",
              " THAT OF THE",
              "   BOTTOMS.");
         return(0);
    }
    if ((x_dist (= x_bott) && (x_dist != -1.0))
    {
         ermess(7,
              " COMPOSITION",
              "OF THE LIGHT",
              "COMPONENT IN",
              "BOTTOMS MUST",
              "BE LESS THAN",
              " THAT OF THE",
              " DISTILLATE.");
         return(0);
    }

    if (nostages ( 0)
    {
         ermess(5,
              " A NEGATIVE",
              "  NUMBER OF",
              " STAGES HAS",
              "    BEEN",
              "  SPECIFIED.");
         return(0);
    }
    if ((!optimal) && (nostages ( feedtray) && (nostages != 0))
    {
         ermess(6,
              "  THE FEED ",
              "TRAY NUMBER",
              " IS GREATER",
              "  THAN THE",
              "  NUMBER OF",
              "   STAGES ");
```

```
        return(0);
    }

    if ((rratio < 0.0) && (rratio != -1.0))
    {
        ermess(3,
            "  THE REFLUX",
            "RATIO MUST BE",
            "NON-NEGATIVE.");
        return(0);
    }

    _setviewport(0.0, 0.0, 0.66, 1.0, -1, BLACK);
    _setworld(0.0, 0.0, 660.0, 1000.0);
    draw_dia();
    draw_eql();

    operpts = 3;

/* Solving for the Number of Trays in a Single Feed Column  */
    if (solvfor == 5)
    {
        char lambda[20];
        int  stagetrk;

        if (!copline(1))
        {
            ermess(   4,
                "THE SPECIFIED",
                "  SYSTEM IS",
                " PHYSICALLY",
                " IMPOSSIBLE");
            return(0);
        }
        if (optimal)
        {
            cur_x = opl[1][1];
            cur_y = opl[2][1];
            _setcolor(WHITE);
            _movabs(diax(cur_x), diay(cur_y));

            stagetrk = 0;
            while (cur_x < x_dist)
            {
                cur_y = equil_y(cur_x);
                _lnabs(diax(cur_x), diay(cur_y));
                cur_x = oper_x(cur_y);
                _lnabs(diax(cur_x), diay(cur_y));
                stagetrk++;
            }
            nostages = stagetrk;
        }
        else
        {
            tmp_x = opl[1][2];
            tmp_y = opl[2][2];
```

```
if (top_to_bott)
{
     _setcolor(GREEN);
     _movabs(diax(opl[1][3]), diay(opl[2][3]));
     _lnabs(diax(opl[1][2]), diay(opl[2][2]));
     _movabs(diax(ofx), diay(ofy));
     _lnabs(diax(opl[1][1]), diay(opl[2][1]));
     _setcolor(BROWN);
     _movabs(diax(ofx), diay(ofy));
     _lnabs(diax(x_feed), diay(x_feed));

     cur_x = opl[1][3];
     cur_y = opl[2][3];
     _setcolor(WHITE);
     _movabs(diax(cur_x), diay(cur_y));

     i = 0;
     while ((cur_x ) x_bott) || (i ( feedtray))
     {
          i++;
          cur_x = equil_x(cur_y);
          _lnabs(diax(cur_x), diay(cur_y));

          if (i == feedtray)
          {
               if (cur_x ) ofx)
               {
                    ermess(   4,
                    "THE SPECIFIED",
                    "  SYSTEM IS",
                    " PHYSICALLY",
                    " IMPOSSIBLE");
                    return(0);
               }
               opl[1][2] = ofx;
               opl[2][2] = ofy;
          }
          cur_y = oper_y(cur_x);
          _lnabs(diax(cur_x), diay(cur_y));
     }
     nostages = i;
}
if (bott_to_top)
{
     _setcolor(GREEN);
     _movabs(diax(opl[1][1]), diay(opl[2][1]));
     _lnabs(diax(opl[1][2]), diay(opl[2][2]));
     _movabs(diax(ofx), diay(ofy));
     _lnabs(diax(opl[1][3]), diay(opl[2][3]));
     _setcolor(BROWN);
     _movabs(diax(ofx), diay(ofy));
     _lnabs(diax(x_feed), diay(x_feed));

     cur_x = opl[1][1];
     cur_y = opl[2][1];
     _setcolor(WHITE);
```

```
                _movabs(diax(cur_x), diay(cur_y));

                i = 0;
                while ((cur_x ( x_dist) || (i ( feedtray))
                {
                     i++;
                     cur_y = equil_y(cur_x);
                     _lnabs(diax(cur_x), diay(cur_y));
                     if (i == feedtray)
                     {
                          if (cur_y ( ofy)
                          {
                               ermess(  4,
                               "THE SPECIFIED",
                               "  SYSTEM IS",
                               " PHYSICALLY",
                               " IMPOSSIBLE");
                               return(0);
                          }
                          opl[1][2] = ofx;
                          opl[2][2] = ofy;
                     }
                     cur_x = oper_x(cur_y);
                     _lnabs(diax(cur_x), diay(cur_y));
                }
                nostages = i;
           }
           opl[1][2] = tmp_x;
           opl[2][2] = tmp_y;
      }

      sprintf(lambda, "%d STAGES", nostages);

_setviewport(norcomx1, norcomy1, norcomx2, norcomy2, GREEN, GREY);
_setworld(0.0, 0.0, 1000.0, 1000.0);

      _setext(2,1,0,1);
      _settextclr(WHITE, RED);
      attext(40.0, 850.0, " THE ANSWER ");
      _setext(1,1,0,1);
      _settextclr(YELLOW, GREY);
      attext(210.0, 700.0, "FOR THE");
      attext(210.0, 650.0, "DESIRED");
      attext(70.0, 600.0, "SEPARATION,");
      attext(140.0, 500.0, lambda);
      attext(70.0, 450.0, "ARE NEEDED.");

      _settextclr(L_GREEN, GREY);
      if (mouse_in_use)
      {
           attext(160.0, 250.0, "CLICK ANY");
           attext(10.0, 200.0, "BUTTON ON THE");
           attext(160.0, 150.0, "MOUSE TO");
           attext(160.0, 100.0, "CONTINUE.");
      }
      else
```

```
        {
             attext(10.0, 250.0, "HIT ANY KEY");
             attext(10.0, 200.0, "ON KEYBOARD");
             attext(10.0, 150.0, "TO CONTINUE.");
        }

        deltcur();

        sleep();

        return(1);
    }
/* Solve for the Distillate Composition      */
    if (solvfor == 1)
    {
        int  i;
        float     cur_x, cur_y, overshot, upper, lower;

        upper = 1.0;
        lower = x_feed;

        do
        {
             x_dist = ((upper - lower)/2) + lower;

             if (copline(2))
             {
                  overshot = trial();
                  if (overshot ) 0.0)
                       lower = x_dist;
                  if (overshot ( 0.0)
                       upper = x_dist;
                  if (overshot == 0.0)
                       break;
             }
             else
             {
                  upper = x_dist;
                  overshot = 1.0;
             }

             i++;
             if (i == 25)
             {
                  ermess(   4,
                       "THE SPECIFIED",
                       "  SYSTEM IS",
                       " PHYSICALLY",
                       " IMPOSSIBLE");
                  x_dist = -1.0;
                  return(0);
             }

        } while (fabs(overshot) ) 0.001);

        draw_answer();
```

```
        return(1);
    }
/* Solve for Feed Composition      */
    if (solvfor == 2)
    {
        int  i;
        float     overshot, upper, lower;

        upper = x_dist;
        lower = x_bott;
        i = 0;
        do
        {
            x_feed = ((upper - lower)/2) + lower;

            if (copline(2))
            {
                overshot = trial();
                if (overshot ) 0.0)
                    upper = x_feed;
                if (overshot ( 0.0)
                    lower = x_feed;
                if (overshot == 0.0)
                    break;
            }
            else
            {
                lower = x_feed;
                overshot = 1.0;
            }
            if (++i == 18)
            {
                ermess(   4,
                    "THE SPECIFIED",
                    "  SYSTEM IS",
                    " PHYSICALLY",
                    " IMPOSSIBLE");
                x_feed = -1.0;
                return(0);
            }

        } while (fabs(overshot) ) 0.001);

        draw_answer();
        return(1);
    }
/* Solve for the Bottoms Composition        */
    if (solvfor == 3)
    {
        int  i;
        float     cur_x, cur_y, overshot, upper, lower;

        upper = x_feed;
        lower = 0.0;

        do
```

```
        {
            x_bott = ((upper - lower)/2) + lower;

            if (copline(2))
            {
                overshot = trial();
                if (overshot ) 0.0)
                    upper = x_bott;
                if (overshot ( 0.0)
                    lower = x_bott;
                if (overshot == 0.0)
                    break;
            }
            else
            {
                lower = x_bott;
                overshot = 1.0;
            }
            i++;
            if (i == 20)
            {
                ermess(   4,
                    "THE SPECIFIED",
                    "  SYSTEM IS",
                    " PHYSICALLY",
                    " IMPOSSIBLE");
                x_bott = -1.0;
                return(0);
            }


        } while (fabs(overshot) ) 0.001);

        draw_answer();
        return(1);
    }

/* Solve for Reflux Ratio          */
    if (solvfor == 4)
    {
        int  i;
        float    overshot, upper, lower;

        upper = 100.0;
        lower = 0.0;

        i = 0;
        do
        {
            rratio = ((upper - lower)/2) + lower;

            if (copline(2))
            {
                overshot = trial();
                if (overshot ) 0.0)
                    lower = rratio;
```

```
                    if (overshot < 0.0)
                        upper = rratio;
                    if (overshot == 0.0)
                        break;
                }
                else
                {
                    upper = rratio;
                    overshot = 1.0;
                }
                i++;
                if (i == 30)
                {
                    ermess(  4,
                        "THE SPECIFIED",
                        "  SYSTEM IS",
                        " PHYSICALLY",
                        " IMPOSSIBLE");
                    rratio = -1.0;
                    return(0);
                }

            } while (fabs(overshot) > 0.001);

            draw_answer();
            return(1);
        }
    }
```

```
/*
     Last Revision
     5/8/86

     MC4.C

     This file contains the following procedures:

          diax()         dotmatrix()
          diay()         menubox()
          invdiax()      _getch()
          invdiay()
          draw_dia()
          status().

*/


#include (stdio.h)
#include (color.h)
#include (math.h)
#include (mcext.h)

float     diax(), diay(), invdiax(), invdiay();

float diax(x)
float     x;
{
     float     xint;
     float     answer;

     xint = fabs(DIAX2 - DIAX1);
     answer = (float)(DIAX1 + xint * x);
     return(answer);
}

float invdiax(x)
float     x;
{
     float     xint;
     float     answer;

     xint = fabs(DIAX2 - DIAX1);
     answer = (float)((x - DIAX1)/xint);
     return(answer);
}

float diay(y)
float     y;
{
     float     yint;
     float     answer;

     yint = fabs(DIAY2 - DIAY1);
     answer = (float)(DIAY1 + yint * y);
     return(answer);
```

```
}

float invdiay(y)
float     y;
{
     float     yint;
     float     answer;

     yint = fabs(DIAY2 - DIAY1);
     answer = (float)((y - DIAY1)/yint);
     return(answer);
}



draw_dia()
{
     int  i;

     _setcolor(CYAN);
     _movabs((diax(0.0)), (diay(0.0)));
     _lnabs((diax(1.0)), (diay(0.0)));
     _lnabs((diax(1.0)), (diay(1.0)));
     _lnabs(diax(0.0), diay(1.0));
     _lnabs(diax(0.0), diay(0.0));
     _lnabs(diax(1.0), diay(1.0));


     /* y-axis tick marks     */

     for (i = 0; i (= 10; i++)
     {
          _movabs(diax(0.0), diay(1.0 - i/10.0));
          _lnabs(diax(-0.02), diay(1.0 - i/10.0));
     }

     /* x-axis tick marks     */

     for (i = 0; i (= 10; i++)
     {
          _movabs(diax(1.0 - i/10.0), diay(0.0));
          _lnabs(diax(1.0 - i/10.0), diay(-0.02));
     }

     /* y-axis title          */
     _setext(1,1,1,1);
     _settextclr(YELLOW, BLACK);
     attext(10.0, 280.0, "Y, LIGHTER COMP. IN VAPOR");

     /* x-axis title          */
     _setext(1,1,0,1);
     _settextclr(YELLOW, BLACK);
     attext(0.0, DIAY1-60.0, "X, LIGHTER COMP. IN LIQUID");


     _setext(1, 1, 0, 1);
```

```
    _settextclr(CYAN, BLACK);
    attext(0.0, diay(1.0), "1.0");
    attext(0.0, diay(0.0), "0.0");
    attext(diax(-0.075), DIAY1-25.0, "0.0");
    attext(diax(0.925), DIAY1-25.0, "1.0");

    _setcolor(GREEN);
    _box(660.0, 0.0, 999.0, 999.0);

    _setext(1,1,0,1);
    _settextclr(GREEN, BLACK);
    attext(675.0, 975.0, "McCABE-THIELE");
    attext(750.0, 950.0, "DIAGRAM");

    if (mouse_in_use)
    {
        _settextclr(WHITE, RED);
        attext(675.0, 50.0, " CLICK MOUSE ");
        attext(675.0, 25.0, "FOR MAIN MENU");
    }
    else
    {
        _settextclr(WHITE, RED);
        attext(675.0, 50.0, " HIT ANY KEY ");
        attext(675.0, 25.0, "FOR MAIN MENU");
    }

    status();
    draw_cond(0);

    deltcur();
}

status()
{
    float    offset;
    char lambda[30];

    switch (eqltype)
    {
    case RVOLATILE:
        _settextclr(L_GREEN, BLACK);
        attext(750.0, 900.0, "RELATIVE");
        attext(725.0, 875.0, "VOLATILITY");
        _settextclr(WHITE, BLACK);
        sprintf(lambda, "%g", alpha);
        attext(750.0, 825.0, lambda);
        break;
    case USERSPEC:
        _settextclr(L_GREEN, BLACK);
        attext(670.0, 900.0, "EQUILBRM LINE");
        attext(670.0, 875.0, "USER DEFINED");
        break;
    default:
        break;
    }
```

```
    _settextclr(GREEN, BLACK);
    switch (probtype)
    {
    case RECTIFY:
         attext(670.0, 775.0, "  RECTIFYING");
         attext(670.0, 750.0, "   SECTION");
         break;
    case STRIP:
         attext(670.0, 775.0, "  STRIPPING");
         attext(670.0, 750.0, "   SECTION");
         break;
    case SINGLE:
         attext(670.0, 775.0, " SINGLE FEED");
         attext(670.0, 750.0, "   COLUMN");
         break;
    default:
         break;
    }

    offset = 75.0;
    _settextclr(L_GREEN, BLACK);
    if (((probtype == RECTIFY) || (probtype == SINGLE))
                          && (x_dist != -1.0))
    {
         _settextclr(L_GREEN, BLACK);
         attext(670.0, 700.0, " DISTILL COMP");
         sprintf(lambda, "%.3g", x_dist);
         _settextclr(WHITE, BLACK);
         attext(770.0, 675.0, lambda);
         offset = 0.0;
    }
    if (x_feed != -1.0)
    {
         _settextclr(L_GREEN, BLACK);
         attext(670.0, (625.0 + offset), "  FEED COMP");
         sprintf(lambda, "%.3g", x_feed);
         _settextclr(WHITE, BLACK);
         attext(770.0, (600.0 + offset), lambda);
    }
    if (((probtype == STRIP) || (probtype == SINGLE))
                               && (x_bott != -1.0))
    {
         _settextclr(L_GREEN, BLACK);
         attext(670.0, (550.0 + offset), " BOTTOMS COMP");
         sprintf(lambda, "%.3g", x_bott);
         _settextclr(WHITE, BLACK);
         attext(770.0, (525.0 + offset), lambda);
    }

    if (((probtype == RECTIFY)||(probtype == STRIP)) && (opslope != -1.0))
    {
         _settextclr(L_GREEN, BLACK);
         attext(670.0, (475.0 + offset), "OPERATN SLOPE");
         sprintf(lambda, "%.3g", opslope);
         _settextclr(WHITE, BLACK);
```

```
        attext(770.0, (450.0 + offset), lambda);
    }

    if ((probtype == SINGLE) && (rratio != -1.0))
    {
        _settextclr(L_GREEN, BLACK);
        attext(670.0, (475.0 + offset), "REFLUX RATIO");
        sprintf(lambda, "%.3g", rratio);
        _settextclr(WHITE, BLACK);
        attext(770.0, (450.0 + offset), lambda);
    }
    if (nostages != 0)
    {
        _settextclr(L_GREEN, BLACK);
        attext(670.0, (400.0 + offset), "NUM OF STAGES");
        sprintf(lambda, "%d", nostages);
        _settextclr(WHITE, BLACK);
        attext(800.0, (375.0 + offset), lambda);
    }

    deltcur();
}

dotmatrix()
{
    int  i;

    _setviewport(0.0, 0.0, 1.0, 1.0, -1, BLACK);
    _setworld(0.0, 0.0, 1000.0, 1000.0);
    draw_dia();

    if (grid)
    {
        _setcolor(L_BLUE);
        for (i = 1; i <= 9; i++)
        {
            _movabs(diax(0.1*i), diay(0.0));
            _lnabs(diax(0.1*i), diay(1.0));
            _movabs(diax(0.0), diay(0.1*i));
            _lnabs(diax(1.0), diay(0.1*i));
        }
    }

    /*
        Draws Equilibrium line in BROWN so that
        it will be shown on the Dot Matrix Printer.
    */
    if (eqltype)
    {
        _setcolor(BROWN);
        _movabs(diax(eql[1][0]), diay(eql[2][0]));

        for (i = 1; i <= points; i++)
        {
            _lnabs(diax(eql[1][i]), diay(eql[2][i]));
        }
```

```
    }
    draw_answer();

    _setgprint(1);
    gprint();
}


/*
    The following routine was updated on Nov 16, 1985 to include
    the possibility of not have the mouse as a pointing device.
    With these additions, the keyboard can be used as an input
    device.  The global flag 'mouse_in_use' determines which method
    to use.
*/
int menubox(num, before, after)
int  num, before, after;
{

    float     boxrad = 40.0,      /* radius of the menu boxes   */
         xcen = 100.0,       /* x value of the box center  */
         ycen;               /* y value of the box center  */
    int  i,             /* a counting variable        */
         chosen;             /* a box choice flag          */


    for (i=1; i<=num; i++)        /* draws icons on screen */
    {
         ycen = 900.0 - (i*100.0);
         _setcolor(before);
         _movabs(xcen - boxrad, ycen + boxrad);
         _lnabs(xcen + boxrad, ycen + boxrad);
         _lnabs(xcen + boxrad, ycen - boxrad);
         _lnabs(xcen - boxrad, ycen - boxrad);
         _lnabs(xcen - boxrad, ycen + boxrad);

    }

   if (mouse_in_use)
   {
    _setlocator(3,1);   /* preparation of crosshair cursor */
    _orglocator(xcen, 900.0);
    _inithcur(10.0,10.0,7);


    chosen = 0;
    while (!chosen)               /* while a box is not chosen  */
    {
         float     mx, my;
         int  bt, inside, old_inside;

         readlocator(&mx, &my, &bt);

         _movhcurabs(mx, my);

         old_inside = inside;
```

```
        inside = 0;

        if ((mx )= (xcen - boxrad) && (mx (= (xcen + boxrad))))
        {
             for (i=1; i(=num; i++)
             {
                  if ((my (= ((900.0+boxrad)-(i*100.0))) &&
                      (my )= ((900.0-boxrad)-(i*100.0))))
                  {
                       inside = i;

                       delhcur();
                       ycen = 900.0 - (i*100.0);
                       _setcolor(after);
                       _movabs(xcen - boxrad, ycen + boxrad);
                       _lnabs(xcen + boxrad, ycen + boxrad);
                       _lnabs(xcen + boxrad, ycen - boxrad);
                       _lnabs(xcen - boxrad, ycen - boxrad);
                       _lnabs(xcen - boxrad, ycen + boxrad);
                       _inithcur(10.0, 10.0, 7);

                       if (CLICK)
                       {
                            chosen = i;
                            return(i);
                       }

                  }
             }
        }
        if ((old_inside) && (inside - old_inside))
        {
             delhcur();
             ycen = 900.0 - (old_inside*100.0);
             _setcolor(before);
             _movabs(xcen - boxrad, ycen + boxrad);
             _lnabs(xcen + boxrad, ycen + boxrad);
             _lnabs(xcen + boxrad, ycen - boxrad);
             _lnabs(xcen - boxrad, ycen - boxrad);
             _lnabs(xcen - boxrad, ycen + boxrad);
             _inithcur(10.0, 10.0, 7);
        }
    }
   }
   else  /*i.e. the mouse is not in use          */
   {
    int  new_box,  old_box;
    int  chr;           /* control varible */

    old_box = new_box = 1;

    ycen = 900.0 - (new_box*100.0);
    _setcolor(after);
    _box(xcen - boxrad, ycen - boxrad, xcen + boxrad, ycen + boxrad);

    chosen = 0;
```

```
    while (!chosen)
    {
        old_box = new_box;

        switch (chr = _getch())
        {
        case 13:  /* RETURN key       */
        case 27:  /* ESC key          */
            chosen = 1;
            break;    /* a selection has been made  */

        case 8:        /* back space key   */
        case 43:  /* + key       */
        case 328: /* up cursor arrow  */
            if (old_box == 1)
                new_box = num;
            else
                new_box = old_box - 1;
            break;

        case 32:  /* SPACE bar        */
        case 45:  /* - key       */
        case 336: /* down cursor arrow     */
            if (old_box == num)
                new_box = 1;
            else
                new_box = old_box + 1;
            break;

        case 327: /* HOME key         */
            new_box = 1;
            break;

        case 335: /* END key          */
            new_box = num;
            break;

        default:
            break;
        }

        /* If there has been a change      */
        if (new_box != old_box)
        {
            ycen = 900.0 - (old_box*100.0);
            _setcolor(before);
            _movabs(xcen - boxrad, ycen + boxrad);
            _lnabs(xcen + boxrad, ycen + boxrad);
            _lnabs(xcen + boxrad, ycen - boxrad);
            _lnabs(xcen - boxrad, ycen - boxrad);
            _lnabs(xcen - boxrad, ycen + boxrad);

            ycen = 900.0 - (new_box*100.0);
            _setcolor(after);
            _movabs(xcen - boxrad, ycen + boxrad);
            _lnabs(xcen + boxrad, ycen + boxrad);
```

```
                _lnabs(xcen + boxrad, ycen - boxrad);
                _lnabs(xcen - boxrad, ycen - boxrad);
                _lnabs(xcen - boxrad, ycen + boxrad);
            }
        }
        return(new_box);
       }
}


/*
    This is an "intelligent" getch() command that adds 256 to the ASCII
    codes of control characters
*/

int _getch()
{
    int chr = getch();

    if (chr != 0)
        return(chr);

    chr = getch();
    return(256+chr);
}
```

```
/*
     Last Revision
     5/2/86

     MC6.C

     This file contains the following procedures:

          ALTER_COND()
          DRAW_COND()

*/

#include (mcext.h)
#include (color.h)
#include (stdio.h)
#include (math.h)

draw_cond(mode)
int mode;
{
     float     line1y = 100.0,
          line2y = 75.0,
          line3y = 50.0,
          line4y = 25.0;

     float     box1x = 0.0,
          box2x = 220.0,
          box3x = 440.0;

     char lambda[10];

     int  tforeclr,
          tbackclr,
          foreclr,
          backclr;

     switch(mode)
     {
     case 0:
          _setviewport(0.0, 0.850, 0.66, 1.0, GREEN, BLACK);
          tforeclr = GREEN;
          tbackclr = BLACK;
          foreclr = WHITE;
          backclr = BLACK;
          break;
     case 1:
          _setviewport(0.0, 0.850, 0.66, 1.0, GREEN, GREY);
          tforeclr = GREEN;
          tbackclr = GREY;
          foreclr = YELLOW;
          backclr = GREY;
          break;
     case 2:
     default:
          _setviewport(0.0, 0.850, 0.66, 1.0, GREEN, BLACK);
```

```
        tforeclr = GREEN;
        tbackclr = BLACK;
        foreclr = WHITE;
        backclr = BLACK;
        break;
    }


    _setworld(0.0, 0.0, 660.0, 150.0);
    _setcolor(GREEN);
    _movabs(box2x, 0.0);
    _lnabs(box2x, 150.0);
    _movabs(box3x, 0.0);
    _lnabs(box3x, 150.0);
    _setext(1,1,0,1);
    _settextclr(tforeclr, tbackclr);
    attext(box1x+10.0, line1y, "  FEED  ");
    attext(box1x+10.0, line2y, "TRAY LOC");
    attext(box2x+10.0, line1y, "  FEED  ");
    attext(box2x+10.0, line2y, "CONDIT'N");
    if (eqltype == RVOLATILE)
    {
        attext(box3x+10.0, line1y, "RELATIVE");
        attext(box3x+10.0, line2y, "VOLATIL ");
        _settextclr(foreclr, backclr);
        sprintf(lambda, " %.3g", alpha);
        attext(box3x+10.0, line4y, lambda);
    }
    if (eqltype == USERSPEC)
    {
        attext(box3x+10.0, line2y, "  USER  ");
        attext(box3x+10.0, line3y, "SPECIFYD");
    }
    if (!eqltype)
    {
        attext(box3x+10.0, line1y, "NO EQUIL");
        attext(box3x+10.0, line2y, "  LINE  ");
        attext(box3x+10.0, line3y, "SPECIFYD");
    }

    _settextclr(foreclr, backclr);

    if (optimal)
    {
        attext(box1x+10.0, line4y, "OPTIMAL");
    }
    else
    {
        sprintf(lambda, "TRAY %d", feedtray);
        attext(box1x+10.0, line4y, lambda);
    }

    if (q != -1.0)
    {
        sprintf(lambda, "Q=%.3g", q);
        attext(box2x+10.0, line4y, lambda);
```

```
    }
    deltcur();

    /* return nondestructively to the full veiwport        */

    _setviewport(0.0, 0.0, 1.0, 1.0, -1, -1);
    _setworld(0.0, 0.0, 1000.0, 1000.0);


}

int alter_cond()
{
    float     xcenb1 = 110.0,
         xcenb2 = 330.0,
         xcenb3 = 550.0,
         xrad   = 105.0,
         ycen   = 75.0,
         yrad   = 65.0,
         xcen;

    int  chosen, i;
    int  num = 3;


    _setviewport(0.0, 0.850, 0.66, 1.0, -1, -1);
    _setworld(0.0, 0.0, 660.0, 150.0);

    /* Draw menu boxes  */

    _setcolor(BLACK);
    _box(xcenb1-xrad, ycen-yrad, xcenb1+xrad, ycen+yrad);
    _box(xcenb2-xrad, ycen-yrad, xcenb2+xrad, ycen+yrad);
    _box(xcenb3-xrad, ycen-yrad, xcenb3+xrad, ycen+yrad);


    if (mouse_in_use)
    {
         _setlocator(3,1);
         _orglocator(xcenb1, ycen+(yrad*0.75));
         _inithcur(10.0, 10.0, WHITE);

         chosen = 0;
         while (!chosen)
         {
              float     mx, my;
              int  bt, in, old_in;

              readlocator(&mx, &my, &bt);
              _movhcurabs(mx, my);
              old_in = in;
              in = 0;

              if ((my)=(ycen-yrad)) && (my(=(ycen+yrad)))
              {
                   for (i=1; i(=3; i++)
```

```
                {
                    xcen = -110.0 + (i * 220.0);

                    if ((mx (= (xcen + xrad)) &&
                        (mx )= (xcen - xrad)))
                    {
                        in = i;
                        delhcur();
                        _setcolor(MAG);
                        _box(xcen-xrad, ycen-yrad,
                            xcen+xrad, ycen+yrad);
                        _inithcur(10.0, 10.0, WHITE);
                        if (CLICK)
                        {
                            chosen = i;
                            return(i);
                        }
                    }
                }
            }
            if ((old_in) && (in - old_in))
            {
                xcen = -110.0 + (old_in * 220.0);
                delhcur();
                _setcolor(BLACK);
                _box(xcen-xrad, ycen-yrad,
                    xcen+xrad, ycen+yrad);
                _inithcur(10.0, 10.0, WHITE);
            }
        }
    }
    else /* i.e. the mouse is not in use        */
    {
        int  new_box,  old_box;
        int  chr;          /* control varible */

        old_box = new_box = 1;

        xcen = -110.0 + (old_box * 220.0);
        _setcolor(MAG);
        _box(xcen-xrad, ycen-yrad, xcen+xrad, ycen+yrad);

        chosen = 0;
        while (!chosen)
        {
            old_box = new_box;

            switch (chr = _getch())
            {
            case 13:  /* RETURN key       */
            case 27:  /* ESC key          */
                chosen = 1;
                break;    /* a selection has been made  */

            case 8:       /* back space key   */
            case 43:  /* + key       */
```

```
            case 331: /* left cursor arrow     */
                 if (old_box == 1)
                      new_box = num;
                 else
                      new_box = old_box - 1;
                 break;

            case 32:  /* SPACE bar        */
            case 45:  /* - key       */
            case 333: /* right cursor arrow    */
                 if (old_box == num)
                      new_box = 1;
                 else
                      new_box = old_box + 1;
                 break;

            case 327: /* HOME key         */
                 new_box = 1;
                 break;

            case 335: /* END key          */
                 new_box = num;
                 break;

            default:
                 break;
            }

            /* If there has been a change      */
            if (new_box != old_box)
            {
                 xcen = -110.0 + (old_box * 220.0);
                 _setcolor(BLACK);
                 _movabs(xcen - xrad, ycen + yrad);
                 _lnabs(xcen + xrad, ycen + yrad);
                 _lnabs(xcen + xrad, ycen - yrad);
                 _lnabs(xcen - xrad, ycen - yrad);
                 _lnabs(xcen - xrad, ycen + yrad);

                 xcen = -110.0 + (new_box * 220.0);
                 _setcolor(MAG);
                 _movabs(xcen - xrad, ycen + yrad);
                 _lnabs(xcen + xrad, ycen + yrad);
                 _lnabs(xcen + xrad, ycen - yrad);
                 _lnabs(xcen - xrad, ycen - yrad);
                 _lnabs(xcen - xrad, ycen + yrad);
            }
        }
        return(new_box);
    }
}

/*
    Last Revision
    5/2/86
```

```
     MCX.C

     This file contains the following procedures:

          mouse_xy(&x, &y)
          click_buffer()
          feed_loc()
*/


int feed_loc()
{
     if (rorl("USE OPTIMAL FEED LOCATION",
          "  OR SPECIFY FEED TRAY.  ",
               "OPTIMAL TRAY", "SPECIFY TRAY") )
     {
          optimal = 1;
          return(1);
     }

     optimal = 0;

     if (rorl("   IN WHICH ORDER SHOULD  ",
          "   THE TRAYS BE NUMBERED. ",
               "TOP TO BOTTOM", "BOTTOM TO TOP") )
     {
          top_to_bott = 1;
          bott_to_top = 0;
     }
     else
     {
          top_to_bott = 0;
          bott_to_top = 1;
     }

     feedtray = (int)key_x(" NUMBER OF SPECIFIED ",
                    "      FEED TRAY.    ");
     return(1);
}


click_buffer()
{
     float     mx, my;
     int  bt;

     do
     {
          readlocator(&mx, &my, &bt);
     } while ((bt != 128) && (bt != 0));
}

float mouse_xy(ptrx, ptry)
float *ptrx, *ptry;
{
     char xlambda[8], ylambda[8];
```

```
float     chosen_x, mx, my;
int  bt;

_setviewport(nordiax1, nordiay1, nordiax2, nordiay2, CYAN, -1);
_setworld(0.0, 0.0, 1.0, 1.0);
_setcolor(CYAN);
_movabs(0.0, 0.0);
_lnabs(1.0, 1.0);

_orglocator(0.5, 0.5);
click_buffer();

ftinit();
_ftsize(1,10);
_setcolor(WHITE);
_settextclr(WHITE, BLACK);
_ftcolor(WHITE, BLACK);
_ftlocate(1,1);
ftext("X =      ");
_ftlocate(2,1);
ftext("Y =      ");

do
{
     readlocator(&mx, &my, &bt);
     if (bt )= 128)
     {
          if (mx )= 1.0)
               mx = 1.0;
          if (mx (= 0.0)
               mx = 0.0;

          _movabs(mx, 0.0);
          _rlnabs(mx, 1.0);
          sprintf(xlambda, "%.2f", mx);
          _ftlocate(1, 5);
          ftext(xlambda);
     }
} while (!CLICK);

click_buffer();

chosen_x = mx;
_movabs(0.0, 0.5);
_rlnabs(1.0, 0.5);
_setxor(1);
_movabs(mx, 0.0);
_lnabs(mx,1.0);
_orglocator(mx, 0.5);
do
{
     readlocator(&mx, &my, &bt);
     if (bt )= 128)
     {
          if (my )= 1.0)
               my = 1.0;
```

```
            if (my (= 0.0)
                my = 0.0;
            _movabs(0.0, my);
            _rlnabs(1.0, my);
            sprintf(ylambda, "%.2f", my);
            _ftlocate(2, 5);
            ftext(ylambda);
        }
    } while (!CLICK);

    click_buffer();

    _movabs(chosen_x, 0.0);
    _lnabs(chosen_x,1.0);
    delln();
    _setxor(0);
    _setcolor(WHITE);
    _box(chosen_x-0.01, my-0.01, chosen_x+0.01, my+0.01);
    _setcolor(RED);
    _box(chosen_x-0.005, my-0.005, chosen_x+0.005, my+0.005);

    *ptrx = chosen_x;
    *ptry = my;
}
```

```
/*
     THIS VERSION OF _HALO IS MADE ESPECIALLY FOR THE MC1 PROGRAM.
     11/16/85 {RLN}

     Useful Halo definitions to avoid messy pointer arithmetic.
     Initial version prepared by R. Natter

     Revision 6/28/85    {RLN}
     Includes fast text
     and Mouse locator commands.

     Revision 7/11/85.  {BMA}
     Rubberband Shapes
     Inqclr

     Revision 7/21/85    {RLN}
     gprint commands, moveto and movefrom

     Revision 10/22/85   {RLN}
     Updated to latest HALO version.
*/


_movabs(x,y)
float x, y;
{
     float v, w;
     v = x;
     w = y;
     movabs(&v,&w);
}

_lnabs(x,y)
float x,y;
{
     float w, v;
     v = x;
     w = y;
     lnabs(&v,&w);
}

_setcolor(i)
int i;
{
     int j;
     j=i;
     setcolor(&j);
}


/* This routine initializes halo graphics   */

_initgraphics(m)
int m;
{
     int one = 1;
```

```
     initgraphics(&m);
     setieee(&one);          /*  working with correct real # format */
}

/* The world coordinates are set in this routine   */

_setworld(llx,lly,urx,ury)
double llx,lly,urx,ury;
{
     float     xl = (float)llx,
          yl = (float)lly,
          xh = (float)urx,
          yh = (float)ury;

     setworld(&xl, &yl, &xh, &yh);
}

/* viewport: */

_setviewport(x1,y1,x2,y2,border,backgr)
double x1,y1,x2,y2;
int border,backgr;
{
     float _x1 = (float)x1,
           _y1 = (float)y1,
           _x2 = (float)x2,
           _y2 = (float)y2;

     setviewport(&_x1,&_y1,&_x2,&_y2,&border,&backgr);
}

_setxor(onoff)
int onoff;
{
     int nof = onoff;
     setxor(&nof);
}

_inithcur(height,width,color)
double height,width;
int color;
{
     int clr = color;
     float h = height,
           w = width;

     inithcur(&h,&w,&clr);
}

_inqhcur(x,y,color)
double *x,*y;
int *color;
{
     int clr;
     float xx,yy;
```

```
     inqhcur(&xx,&yy,&clr);
     *x = xx;
     *y = yy;
     *color = clr;
}

_movhcurabs(x,y)
double x,y;
{
     float xx,yy;
     xx = (float)x;
     yy = (float)y;
     movhcurabs(&xx,&yy);
}

_movtcurabs(x,y)
double x,y;
{
     float xx,yy;
     xx = (float)x;
     yy = (float)y;
     movtcurabs(&xx,&yy);
}

_settextclr(color,back)
int color,back;
{
     int  clr = color,
          bak = back;
     settextclr(&clr,&bak);
}

_inittcur(h,w,c)
int h,w,c;
{
     int  hh = h,
          ww = w,
          cc = c;
     inittcur(&hh,&ww,&cc);
}


attext(x,y,txt)
double x,y;
char *txt;
{
     float     xx = (float)x,
          yy = (float)y;
     _movtcurabs(xx,yy);
     text(txt);
}

_ftcolor(fore, back)
int fore, back;
{
     int  f = fore,
```

```
          b = back;
     ftcolor(&f, &b);
}

_ftlocate(row ,col)
int row, col;
{
     int  r = row,
          c = col;
     ftlocate(&r, &c);
}

_ftsize(height, boxh)
int height, boxh;
{
     int  h = height,
          b = boxh;
     ftsize(&h, &b);
}

_bar(x1, y1, x2, y2)
double x1, y1, x2, y2;
{
     float xx1, yy1, xx2, yy2;
     xx1 = x1;
     yy1 = y1;
     xx2 = x2;
     yy2 = y2;
     bar(&xx1, &yy1, &xx2, &yy2);
}

_box(x1, y1, x2, y2)
double x1, y1, x2, y2;
{
     float xx1, yy1, xx2, yy2;
     xx1 = x1;
     yy1 = y1;
     xx2 = x2;
     yy2 = y2;
     box(&xx1, &yy1, &xx2, &yy2);
}

_rbox(x1, y1, x2, y2)
double x1, y1, x2, y2;
{
     float xx1, yy1, xx2, yy2;
     xx1 = x1;
     yy1 = y1;
     xx2 = x2;
     yy2 = y2;
     rbox(&xx1, &yy1, &xx2, &yy2);
}

_orglocator(x, y)
double x, y;
{
```

```
     float xx, yy;
     xx = x;
     yy = y;
     orglocator(&xx, &yy);
}

_setlocator(dev, port)
int dev, port;
{
     int  d = dev,
          p = port;
     setlocator(&d, &p);
}

_setext(h, w, p, m)
int h, w, p, m;
{
     int  hh = h,
          ww = w,
          pp = p,
          mm = m;
     settext(&hh, &ww, &pp, &mm);
}

_rlnabs(x, y)
double x, y;
{
     float xx, yy;
     xx = (float)x;
     yy = (float)y;
     rlnabs(&xx, &yy);
}

_setgprint(dev)
int dev;
{
     int ddev;
     ddev = dev;
     setgprint(&ddev);
}
```