REFERENCES

1. Bigardi, G., and M.G. Grottoli, Development of a New Simulation Model for Real Trays Distillation Column., Computer & Chemical Engineering, Vol. 13, no. 4-5 (April-May 1989): 441-449.
2. Cairns. Brett P. and Furzer, lan A., Multicomponent Three-Phase Azeotropic Distillation 1. Extensive Experimental Data and Simulation Results, Ind. Eng. Chem. Res., Vol. 29, no. 7 (July 1990): 1349-1363.
3. Cairns. Brett P. and Furzer, lan A., Multicomponent Three-Phase Azeotropic Distillation 2. Phase-stability and phase-splitting algorithms., Ind. Eng. Chem. Res., Vol. 29, no. 7 (July 1990): 1364-1382.
4. Cairns. Brett P. and Furzer, lan A., Multicomponent Three-Phase Azeotropic Distillation 3. Modern Thermodynamic Models and Multiple Solutions, Ind. Eng. Chem. Res., Vol. 29, no. 7 (July 1990): 1383-1395.
5. Chapra, S.C. and R. P. Canale, Numerical Methods for Engineers, 2 ed., McGraw-Hill Book Co., Singapore, 1990.
6. Charles H. Fisher, Calculation of Critical Temperatures from the Number of Carbons in Organic Molecules, JAOCS, Vol. 67, no. 2 (Febuary 1990): 101-102.
7. Constantinides, A., Applied Numerical Methods with Personal Computers, McGraw-Hill Book Co., Singapore, 1987.
8. de Vlaminck, Prediction of Thermodynamic Properties of Organic Fluids with the Help of Computer Software, International Chemical Engineering, Vol. 30, no.30 (July 1990): 479-486.
9. Edmister, W. C., and B.I. Lee, Applied Hydrocarbon Thermodynamics, Vol.1, second edition, Gulf Publishing Company, Texas, 1984.

10. Edmister, W. C., and B.I. Lee, Applied Hydrocarbon Thermodynamics, Vol.2, second edition, Gulf Publishing Company, Texas, 1988.

11. Ernest J. Henley and J.D. Seader, Equilibrium-Stage Seperation Operations in Chemical Engineering, John Wiley & Sons Inc., New York ,1981.

12. Encyclopedia of Chemical Technology, 3rd ed., 17(1980): 186-187.

13. Gani R. and Cameron I.T., Extension of Dynamic Models of Distillation Columns to Steady-State Simulation., Computer & Chemical Engineering, Vol. 13, no. 3 (March 1989): 271-280.

14. Glinos R. and Malone M. F., Net Work Consumption in Distillation. Short-Cut Evaluation and Applications to Synthesis., Computer & Chemical Engineering, Vol. 13, no. 3 (March 1989): 295-305.

15. Gorak A., Kraslaki A. and Vogelpohl A., The Simulation and Optimization of Multicomponent Distillation, International Chemical Engineering, Vol.30, no.1 (January 1990): 1-15.

16. Hasting, C. Jr., Approximations for Digital Computers, Princeton Univ. Press, Princeton, N.J., 1955.

17. Holland, C.D., Fundamentals of Multicomponent Distillation, McGraw-Hill Book Co., USA., 1981.

18. ________, Unsteady Stage Processes with Applications in Multicomponent Distillation, Pritice-Hall, Englewood Cliffs, New Jersey, 1966.

19. ________, Multicomponent Distillation, Pritice-Hall, Englewood Cliffs, New Jersey, 1963.

20. Karen, S.P., Perthomassen, and A. Fredenslund, Thermodynamics of Petroleum Mixtures Containing Heavy Hydrocarbons. 1. Phase Envelope Calculations by Use of the Soave-Redlich-Kwong Equation of State., Ind. Eng. Chem. Process Des. Dev., Vol. 23, no.1 (1984): 163-170.

21. Karen, S.P., Perthomassen, and A. Fredenslund, Thermodynamics of Petroleum Mixtures Containing Heavy Hydrocarbons. 2. Flash and PVT Calculation with the Soave-Redlich-Kwong Equation of State., Ind. Eng. Chem. Process Des. Dev., Vol. 23, no. 3 (1984): 566-573.

22. ________, Perthomassen, and A. Fredenslund, Thermodynamics of Petroleum Mixtures Containing Heavy Hydrocarbons. 3. Efficient Flash Calculation Procedures Using the Soave-Redlich-Kwong Equation of State., Ind. Eng. Chem. Process Des. Dev., Vol. 24, no.4 (1985): 948-954.

23. Luyben, W.L., Practical Distillation Control, Van Nostrand Reinhold, New York, 1992.

24. Maxwell, J.B., Data Book on Hydrocarbons, D. Van Nostrand Co., Princeton, N.J., 1950.

25. Maxwell, J.B. and L.S. Bonnell, "Derivation and Precision of a New Vapor Pressure Correlation for Petroleum Hydrocarbon", Ind. Eng. Chem., 49(7), (1957): 1187-1196.

26. Mazaev, V.N., and V.S. Nikitin, Matrix Calculation for Mass Exchange During Distillation of a Multicomponent Mixture on a Contact Stage with a Cross Flow., Theoretical Foundations of Chemical Engineering (English Translation of Teoreticheskie Osnovy Khimicheskoi Tedhnologii), Vol. 22, no. 5 (May 1989): 409-412.

27. Nelson, W.L., Petroleum Refinery Engineering, 4 ed., McGraw-Hill Book Co., New York, 1958.

28. ________, "Does crude boil at 1,400 ^{o}F?", Oil and Gas J., 66(12), (1968) :125-130.

29. Onna A. and Mbala, Hikolo A., SOR Method for Multistaged Separation Columns Computations, Computers & Chemical Engineering, Vol. 17, no.8 (1993): 799-805.

30. Perry, R.H. and D. Green, Perry's Chemical Engineers' Handbook, sixth edition, McGraw-Hill Book Co., New York, 1984.

31. Ratzsch, M.T., H. Kehlen, and J. Schumann, Computer Simulation of Complex Multicomponent Hydrocarbon Distillation by Continuous Thermodynamics., Fluid Phase Equilibria, Vol. 51 (November 1989): 133-146.

32. Schwartzentrubber Jacquues, Renon Henri, and Watanasiri Suphat, Development of a New Cubic Equation of State for Phase Equilibrium Calculations., Fluid Phase Equilibria., Vol. 52, n pt 1 (December 1989): 127-134.

33. Seader J.D., The Rate-based Approach for Modeling Staged Separations, Chemical Engineering Progress (October 1989): 41-49.

34. Torres-Marchal Carlos, Cantalino Adalberto L.,and De Brito Rita Maria, Prediction of Vapor-Liquid Equilibria (VLE) from Dilute System Data Using the SRK Equation of State., Industrial Applications., Fluid Phase Equilibria., Vol. 52, n pt 1 (December 1989): 111-117.

35. Tremblay, J-P. and J.M. Dedourek, Programming in Pascal, 2 ed., McGraw-Hill Book Co., Singapore, 1989.

36. Van Winkle, M., Distillation, McGraw-Hill Book Co., USA., 1969.

37. Wang, J. C., and G.E. Henke, Tridiagonal Matrix for Distillation, Hydrocarbon Processing, Vol. 45, no. 8 (August 1966): 155-163.

38. Watkins, R.N., Petroleum Refinery Distillation, 2nd ed., Gulf Publishing Co., Houston, Texas, 1981.

APPENDIX

LIST OF PROGRAM

```
{ FILE NAME : SIMULATE.PAS }

PROGRAM Simulation;
USES Crt,Dos,Newton;

CONST
   T0           = 500;          { deg. F }
   Tref         = 492;          { deg. R }
   R            = 10.73;        { (psia.ft^3)/(lbmol. deg.R) }
   MaxStage     = 20;           { Maximum of Stage }
   MaxComp      = 20;           { Maximum of Component }
   MaxRound     = 30;           { Maximum of Iteration }
   MaxK         = 10;
   LengthComp   = 15;
   Enter        = #13;
   Esc          = #27;
   Up           = #72;
   Down         = #80;
   Left         = #75;
   Right        = #77;

TYPE
   CompRec = Record
                name  : string[15];
                pos   : integer;
             End;
   Pointer   = ^MtxRec;
   MtxRec    = Record
                 Mtx  : ARRAY[1..Maxcomp,1..Maxstage] of real;
                 Ptr  : Pointer;
               End;
   Matrix1   = ARRAY[1..MaxRound,1..MaxStage] of real;
   Vector    = ARRAY[1..MaxStage] of real;
   Vector1   = ARRAY[1..MaxComp] of CompRec;
   Vector2   = ARRAY[1..MaxComp] of real;
   Vector3   = ARRAY[1..MaxK] of real;
VAR
   F,W,U,Q,L,P          : Vector;
   T,V,TT               : Matrix1;
   K_val,X,X_pre        : Pointer;
   Pi,Co,Y,z            : Pointer;
   Fugal,Fugav          : Pointer;
   A_prime, B_prime     : Pointer;
   Hv,Hl                : Vector;
   T_old                : Vector3;
   Bottom,Tf,Pb0,Hf     : real;
   Comp,Anto            : Vector1;
   ai,alpha             : Pointer;
   N,m,i,Fj,It,k        : integer;
   ch,Funckey           : char;
   name                 : ARRAY[1..MaxComp] of string[15];
   A,B,bb,aa,Zl,Zv                  : Vector;
   Pc,Tc,ww,bi,ac,mm,SG,Tb          : Vector2;
   A1,B1,C1,T50,API,Pb              : Vector2;
   min,max,eTc,ePc,eW               : real;
   isconverged,check                : boolean;
```

```
   ii,jj,iv,j                : integer;
   outfile                   : text;

{$M 65520,0,655360}
{$I COMPUTEX.PAS}
{$I FEED.PAS}
{$I PHY-PROP.PAS}
{$I THER-PROP.PAS}
{$I INITY.PAS}
{$I FUGACITY.PAS}
{$I SRK-CONS.PAS}
{$I DISPLAY.PAS}
{$I COMPUTEH.PAS}

PROCEDURE GetX;
VAR
   i,j    : integer;
   infile : text;
Begin
   Assign(infile,'GetX20.txt');
   Reset(infile);
   j := 1;
   While not(eof(infile)) Do
      Begin
         For i := 1 to m Do
            Begin
              read(infile,X^.mtx[i,j]);
              Y^.mtx[i,j]:=X^.mtx[i,j];
              write(X^.mtx[i,j]:5:4);
           End;
        readln(infile);
        writeln;
        j := j + 1;
      End;
   Close(infile);
End;

PROCEDURE Initials;
VAR
   i,j : integer;
Begin
   New(z);          New(K_val);
   New(X);          New(X_pre);
   New(Pi);         New(Y);
   New(Fugal);      New(Fugav);
   New(A_prime);    New(B_prime);
   New(ai);         New(alpha);
   New(Co);
   z^.ptr         := Nil;      K_val^.ptr     := Nil;
   X^.ptr         := Nil;      X_pre^.ptr     := Nil;
   Pi^.ptr        := Nil;      Y^.ptr         := Nil;
   ai^.ptr        := Nil;      alpha^.ptr     := Nil;
   Fugav^.ptr     := Nil;      Fugal^.ptr     := Nil;
   A_prime^.ptr   := Nil;      B_prime^.ptr   := Nil;
   Co^.ptr        := Nil;
   For i := 1 to MaxComp Do
```

```
      For j := 1 to MaxStage Do
         Begin
            z^.mtx[i,j] := 0;
            W[j] := 0;  U[j] := 0;
            Q[j] := 0;  F[j] := 0;
            L[j] := 0;  P[j] := 0;
         End;
   For i := 0 to MaxRound Do
      For j := 1 to MaxStage Do
         Begin
            T[i,j] := 0;
            V[i,j] := 0;
         End;
End;

FUNCTION Power(x,n : real) : real;
Begin
   Power := Exp(ln(x) * n);
End;

FUNCTION Power1(x : real;
                n : integer) : real;
VAR
   i   : integer;
   sum : real;
Begin
   If n = 0 Then
      Power1 := 1
   Else
      Begin
         sum := 1;
         For i := 1 to n Do
            sum := sum * x;
         Power1 := sum;
      End;
End;

PROCEDURE InitialV;
VAR
   i,j,line : integer;
Begin
   clrscr;
   gotoxy(22,3);write('     Initialize tear variables');
   gotoxy(22,4);write('====================================');
   gotoxy(22,5);write(' Stage  j                Vj,lbmole/hr');
   gotoxy(22,6);write('====================================');
      { Print To file}
   writeln(outfile);
   writeln(outfile,'      Initialize tear variables');
   writeln(outfile,'====================================');
   writeln(outfile,' Stage  j                Vj,lbmole/hr');
   writeln(outfile,'====================================');
   line := 7;
   For j := 1 to N Do
      Begin
         gotoxy(28,line);  write(j);
```

```
         Inc(line);
      End;
   line := 7;
   For j := 1 to N Do
      Begin
         gotoxy(47,line); Read(V[It,j]);
         writeln(outfile,j:6,V[It,j]:24:2);
         Inc(line);
      End;
   Close(outfile);
End;

PROCEDURE InitialT;
VAR
   j     : integer;
   sigma : real;
Begin
   For j := 1 to N Do
      Begin
         T[1,j]  := T0+460;
         TT[1,j] := T[1,j]+460;
      End;
End;

PROCEDURE InitialX;
VAR
   i,j,line,col : integer;
Begin
   clrscr;
   write('========================================');
   writeln('==================================');
   write('stage j');
   For i := 1 to m do
      write(comp[i].name : length(comp[i].name)+5);
   writeln;
   write('========================================');
   writeln('==================================');
   line := 4;
   col  := 10;
   For j := 1 to N do
      Begin
         gotoxy(4,line);  write(j);
         If j = Fj Then
           Begin
             For i := 1 to m Do
               Begin
                  gotoxy(col,line); write(X^.mtx[i,j]:3:4);
                  col := col + 6;
               End;
           End
         Else
           Begin
               For i := 1 to m Do
                  Begin
                     gotoxy(col,line);  read(X^.mtx[i,j]);
                     col := col + 6;
```

```
                    End;
                End;
              col := 10;
              line := line + 1;
          End;
End;

PROCEDURE ComputeNewV;
   PROCEDURE AdjustV;
   VAR
      j,m : integer;
      C,A,B : real;
   Begin
      For j := 3 to N Do
        Begin
          V[It,j] := 1/(Hv[j]-Hl[j-1])*((Hv[j-1]-Hl[j-1])*
                     (V[It,j-1]+W[j-1])+(Hl[j-1]-Hl[j-2])*
                     L[j-2] - F[j-1]*(Hf-Hl[j-1])+Q[j-1]);
        End;
   End;
Begin
   ComputeEnthal;
   AdjustV;
End;

Begin      { Main Program }
   Initials;
   It := 1;
   Feed;
   InitialT;              { Set Initial T from deg.F to deg.R }
   InitialV;              { Initial V[j] }
   GetX;                  { Set x[i,j] = y[i,j] }
   It := 2;
   For i := 1 to m Do
      For j := 1 to N do
         X_pre^.mtx[i,j] := X^.mtx[i,j];
   ComputeY;              { Call from InitY subprog.    }
   ComputeK;              { Call from Phy-prop subprog.}
   ComputeX;              { Call from Compute X subprog. }
   For j := 1 to N Do
      Begin
        For iv := 1 to m Do
          While (ABS(X^.mtx[iv,j]-X_pre^.mtx[iv,j]) >= 0.0001) Do
             Begin
                For ii := 1 to m Do
                   For jj := 1 to N do
                      X_pre^.mtx[ii,jj] := X^.mtx[ii,jj];
                ComputeNewV;
                It := It+1;
                ComputeY;   { Call from InitY subprog.     }
                ComputeK;   { Call from Phy-prop subprog. }
                ComputeX;   { Call from Compute X subprog. }
             End;
      End;
   Display;
End.       { End Main Program }
```

```
{ THER-PRO.PAS }

PROCEDURE Thermodynamic(X : Pointer;
                        VAR Z : Vector;
                        name : string);

VAR
   j : integer;
{$I SRK-CONST.PAS}

Begin
    SRK_const(X);
    For j := 1 to N Do
       Begin
          NewtonOfPolynomial(j,min,max,A[j],B[j]);
          If name = 'X'Then
             Zl[j] := min
          Else
             If name = 'Y' Then
                Zv[j] := max;
       End;
End;
```

```
{ SRK-CONS.PAS }

PROCEDURE SRK_const(X : Pointer);
VAR
   i,j,jj : integer;
   ab     : real;
   flag   : boolean;

   FUNCTION Power(x,n : real) : real;
   Begin
      Power := Exp(ln(x) * n);
   End;

Begin
   For i := 1 to m Do
      Begin
         bi[i] := (0.08644*R*Tc[comp[i].pos])/Pc[comp[i].pos];
         mm[i] := 0.48 + 1.574*ww[comp[i].pos]
                  - 0.176*SQR(ww[comp[i].pos]);
         ac[i] := 0.42748*SQR(R)*SQR(Tc[comp[i].pos])
                  / Pc[comp[i].pos];
      End;
   For j := 1 to N Do
      For i := 1 to m Do
         Begin
            alpha^.mtx[i,j] := 1 + mm[i]*(1-Power((T[k,j]
                               / Tc[comp[i].pos]),0.5));
            alpha^.mtx[i,j] := alpha^.mtx[i,j]*alpha^.mtx[i,j];
            ai^.mtx[i,j] := ac[i] * alpha^.mtx[i,j];
         End;
   For j := 1 to N Do
      Begin
         bb[j] := 0;
         aa[j] := 0;
         For i := 1 to m Do
            Begin
               bb[j] := bb[j] + X^.mtx[i,j]*bi[i];
               ab := 0;
               For jj := 1 to m Do
                  ab := ab + X^.mtx[i,j]* X^.mtx[jj,j]
                        * Power(ai^.mtx[i,j]*ai^.mtx[jj,j],0.5);
               aa[j] := aa[j] + ab;
            End;
      End;
   For j := 1 to N Do
      Begin
         A[j] := (aa[j]*P[j]) / SQR(R*T[k,j]);
         B[j] := bb[j]*P[j] / (R*T[k,j]);
      End;
End;
```

```
{ FILE NAME : PHY-PROP.PAS }

PROCEDURE ComputeK;

   FUNCTION Power(x,n : real) : real;
   Begin
      Power := Exp(ln(x) * n);
   End;

   PROCEDURE Init;
   VAR
      i,j : integer;
   Begin
      For j := 1 to MaxComp Do
         For i := 1 to MaxStage Do
            Begin
               K_val^.mtx[j,i] := 0;
            End;
   End;

  PROCEDURE K_value;
  VAR
      i, j : integer;
  Begin
     For i := 1 to m Do
        For j := 1 to N Do
           K_val^.mtx[i,j] := Fugal^.mtx[i,j] /
                              Fugav^.mtx[i,j];
  End;

  Procedure w;
  var i ,j : integer;
  Begin
    writeln('K[i,j]');
    For i := 1 to m Do
       write(comp[i].name:8);
    writeln;
    For j := 1 to N Do
      begin
       For i := 1 to m Do
          Begin
             write(K_val^.mtx[i,j]:10:4);
           End;
       writeln;
       end;
   End;

 Begin
    Init;
    K_value;
    w;
 End;
```

```
{ FILE NAME : NEWTON.PAS }

UNIT Newton;

INTERFACE
PROCEDURE NewtonOfPolynomial(j              : integer;
                             VAR min,max : real;
                             A,B            : real);

IMPLEMENTATION

PROCEDURE NewtonOfPolynomial;
CONST
   M = 3;                    { Degree of Polynomial }
VAR
   xCof,Cof     : ARRAY[1..4] of real;
   RootI,RootR : ARRAY[1..3] of real;
   N,i          : integer;
   Ifit         : integer;
   flag         : boolean;

PROCEDURE Newton;
VAR
   nx,nxx,n2,kj1,l,mt,i,fi,ict          : integer;
   flag1,flag2,flag3,flag4,flag5        : boolean;
   u,ux,uy,v,xt,yt,temp,xt2,yt2,alpha: real;
   sumsq,dx,dy,x,y,xo,yo,inn,xpr,ypr : real;
   PROCEDURE Swap1(VAR cof1 : real;
                   VAR cof2 : real);
   VAR
      tmp : real;
   Begin
      tmp := Cof1;
      Cof1 := Cof2;
      Cof2 := tmp;
   End;

   PROCEDURE Swap2(VAR cof1 : integer;
                   VAR cof2 : integer);
   VAR
      tmp : integer;
   Begin
      tmp := Cof1;
      Cof1 := Cof2;
      Cof2 := tmp;
   End;

   PROCEDURE Proc1;
   Begin
      x   := xo;
      xo  := -10 * yo;
      yo  := -10 * x;
      x   := xo;
      y   := yo;
      inn := inn + 1;
   End;
```

```
PROCEDURE Proc2;
Begin
   ict := 0;
End;

PROCEDURE Proc3;
VAR
   i,l : integer;
Begin
   ux := 0;
   uy := 0;
   v := 0;
   yt := 0;
   xt := 1;
   u := Cof[N+1];
   flag1 := u = 0;
   If not (flag1) Then
      Begin
         For i := 1 to N Do
            Begin
               l := N - i + 1;
               temp := Cof[l];
               xt2 := x * xt - y * yt;
               yt2 := x * yt + y * xt;
               u := u + temp * xt2;
               v := v + temp * yt2;
               fi := i;
               ux := ux + fi * xt * temp;
               uy := uy - fi * yt * temp;
               xt := xt2;
               yt := yt2;
            End;
         sumsq := ux * ux + uy * uy;
         flag2 := sumsq = 0;
         If not(flag2) Then
            Begin
               dx := (v * uy - u * ux) / sumsq;
               x := x + dx;
               dy := -(u * uy + v * ux) /sumsq;
               y := y + dy;
               If (ABS(dy) + ABS(dx) - 0.00001) < 0 then
                  Begin
                     For l := 1 to nxx Do
                        Begin
                           mt := kj1 - l + 1;
                           Swap1(xCof[mt],Cof[l]);
                        End;
                     Swap2(N,nx);
                     flag3 := Ifit = 0;
                     If flag3 Then
                        Begin
                           Ifit := 1;
                           xpr := x;
                           ypr := y;
                           Proc2;
```

```
                    Proc3;
                 End;
             End
          Else
             Begin
                ict := ict + 1;
                If ict < 500 Then
                   Begin
                      Proc3;
                   End
                Else
                   If Ifit = 0 Then
                      If inn < 5 then
                         Begin
                            Proc1;
                            Proc2;
                            Proc3;
                         End
                      Else
                         Begin
                            write('Unable to ');
                            writeln('determine Root);
                            Exit;
                         End
                   Else
                      Begin
                         For l := 1 to nxx Do
                            Begin
                               mt := kj1 - l + 1;
                               Swap1(xCof[mt],Cof[l]);
                            End;
                            Swap2(N,nx);
                            flag3 := Ifit = 0;
                            If flag3 Then
                               Begin
                                  Ifit := 1;
                                  xpr  := x;
                                  ypr  := y;
                                  Proc2;
                                  Proc3;
                               End;
                      End;
             End;
       End;
    End;
End;

PROCEDURE Proc4;
VAR
   l : integer;
Begin
   Cof[2] := cof[2] + alpha * Cof[1];
   For l := 2 to N Do
      Cof[l+1] := Cof[l+1] + alpha * Cof[l] - sumsq * Cof[l-1];
   Repeat
      RootI[N2] := y;
```

```
      RootR[N2] := x;
      N2 := N2 + 1;
      flag4 := sumsq = 0;
      If not(flag4) Then
         Begin
            y := -y;
            sumsq := 0;
         End;
   Until flag4;
   flag5 := N > 0;
End;

PROCEDURE IfitNE0;
Begin
   Ifit := 0;
   If (ABS(y) - 0.0001 * ABS(x)) < 0 Then
      Begin
         y := 0;
         sumsq := 0;
         alpha := x;
         N := N - 1;
         Proc4;
      End
   Else
      Begin
         alpha := x + x;
         sumsq := x * x + y * y;
         N := N - 2;
         Proc4;
      End;
End;

PROCEDURE SumsqEq0;
Begin
   If Ifit = 0 Then
      Begin
         Proc1;
         Proc2;
         Proc3;
      End
   Else
      Begin
         x := xpr;
         y := ypr;
         IfitNE0;
      End;
End;

PROCEDURE UEQ0;
Begin
   x   := 0;
   nx  := nx-1;
   nxx := nxx-1;
   y   := 0;
   sumsq := 0;
   alpha := x;
```

```
      N := N - 1;
      Proc4;
   End;

Begin                   { Of Newton }
   u := 0;
   ux := 0;
   temp := 0;
   nx := N;
   nxx := N + 1;
   n2 := 1;
   kj1 := N + 1;
   For i := 1 to kj1 Do
      Begin
         mt := kj1 - i + 1;
         Cof[mt] := xCof[i];
      End;
   Repeat
      xo := 0.005;
      yo := 0.01;
      inn := 0;
      flag2 := false;
      flag3 := true;
      Repeat
         Proc1;
         Proc2;
         Proc3;
      Until (not(flag3)) or (flag1) or (flag2);
      If  (not(flag3)) Then
          IfitNE0
      Else
          If (flag1) and (flag2) Then
             SumsqEq0
          Else
             If flag1 Then
                UEq0;
   Until not(flag5);
End;                      { Of Newton }

PROCEDURE PrintRoot;
VAR
   i     : integer;
   yy    : real;
   flag1,flag2,flag3,flag4 : boolean;

   PROCEDURE Roots;
   Begin
      If RootI[i] = 0 then
         writeln('Z(',i,') = ',RootR[i]:8:3)
      Else
         If RootR[i] = 0 Then
            writeln('Z(',i,') = ',RootI[i]:8:3)
         Else
            If RootI[i] > 0 then
writeln('Z(',i,') = ',RootR[i]:8:3,'  +',RootI[i]:8:3,'i')
            Else
```

```
writeln('Z(',i,') = ',RootR[i]:8:3,'  -',ABS(RootI[i]):8:3,'i');
   End;

   PROCEDURE Proc5;
   Begin
      yy := 1000;
      Repeat
         flag2 := ABS(RootI[i] * yy) > 32000;
         If flag2 Then
            yy := yy / 10;
      Until not(flag2);
      If RootI[i] = 0 Then
         Roots
      Else
         Begin
            Repeat
               flag3 := ABS(RootI[i] * yy) < 1000;
               If flag3 Then
                  yy := yy * 10;
            Until not(flag3);
            RootI[i] := (RootI[i] * yy) / yy;
            Roots;
         End;
   End;

Begin
   For i:= 1 to M Do
      Begin
         yy := 1000;
         Repeat
            flag1 := ABS(RootR[i] * yy) > 32000;
            If flag1 Then
               yy := yy / 10;
         Until not(flag1);
         If RootR[i] = 0 Then
            Proc5
         Else
            Begin
               Repeat
                  flag4 := ABS(RootR[i] * yy) < 1000;
                  If flag4 Then
                     yy := yy * 10;
               Until not flag4;
               RootR[i] := (RootR[i] * yy) / yy;
               Proc5;
            End;
      End;
End;

PROCEDURE ZValues;
VAR
   i : integer;
Begin
   min := 1.7E38;
   max := 2.9E-39;
   For i := 1 to M Do
```

```
      Begin
         If RootR[i] < min Then
            min := RootR[i];
         If RootR[i] > max Then
            max := RootR[i];
         If RootI[i] <> 0 then
            Begin
               writeln('Z is not real');
               flag := false;
               Exit;
            End;
      End;
End;

Begin         { Of NewtonOfPolynomial }
   xCof[1] := -A * B;            { Coefficient of order 0 }
   xCof[2] := A - B - SQR(B);  { Coefficient of order 1 }
   xCof[3] := -1;                { Coefficient of order 2 }
   xCof[4] := 1;                 { Coefficient of order 3 }
   Ifit := 0;
   N := M;
   If xCof[N+1] = 0  then
      Begin
         writeln('High order coefficient can''t be zero');
         Exit;
      End;
   Newton;
   ZValues;
End;         { Of NewtonOfPolynomial }
End.         { Of Unit Newton2 }
```

```
{ FILE NAME : NEWTON.PAS }

UNIT Newton;

INTERFACE
PROCEDURE NewtonOfPolynomial(j                   : integer;
                             VAR min,max : real;
                             A,B         : real);

IMPLEMENTATION

PROCEDURE NewtonOfPolynomial;
CONST
   M = 3;                      { Degree of Polynomial }
VAR
   xCof,Cof      : ARRAY[1..4] of real;
   RootI,RootR   : ARRAY[1..3] of real;
   N,i           : integer;
   Ifit          : integer;
   flag          : boolean;

PROCEDURE Newton;
VAR
   nx,nxx,n2,kj1,l,mt,i,fi,ict             : integer;
   flag1,flag2,flag3,flag4,flag5           : boolean;
   u,ux,uy,v,xt,yt,temp,xt2,yt2,alpha: real;
   sumsq,dx,dy,x,y,xo,yo,inn,xpr,ypr : real;
   PROCEDURE Swap1(VAR cof1 : real;
                   VAR cof2 : real);
   VAR
      tmp : real;
   Begin
      tmp := Cof1;
      Cof1 := Cof2;
      Cof2 := tmp;
   End;

   PROCEDURE Swap2(VAR cof1 : integer;
                   VAR cof2 : integer);
   VAR
      tmp : integer;
   Begin
      tmp := Cof1;
      Cof1 := Cof2;
      Cof2 := tmp;
   End;

   PROCEDURE Proc1;
   Begin
      x   := xo;
      xo  := -10 * yo;
      yo  := -10 * x;
      x   := xo;
      y   := yo;
      inn := inn + 1;
   End;
```

```
PROCEDURE Proc2;
Begin
   ict := 0;
End;

PROCEDURE Proc3;
VAR
   i,l : integer;
Begin
   ux := 0;
   uy := 0;
   v := 0;
   yt := 0;
   xt := 1;
   u := Cof[N+1];
   flag1 := u = 0;
   If not (flag1) Then
      Begin
         For i := 1 to N Do
            Begin
               l := N - i + 1;
               temp := Cof[l];
               xt2 := x * xt - y * yt;
               yt2 := x * yt + y * xt;
               u := u + temp * xt2;
               v := v + temp * yt2;
               fi := i;
               ux := ux + fi * xt * temp;
               uy := uy - fi * yt * temp;
               xt := xt2;
               yt := yt2;
            End;
         sumsq := ux * ux + uy * uy;
         flag2 := sumsq = 0;
         If not(flag2) Then
            Begin
               dx := (v * uy - u * ux) / sumsq;
               x := x + dx;
               dy := -(u * uy + v * ux) /sumsq;
               y := y + dy;
               If (ABS(dy) + ABS(dx) - 0.00001) < 0 then
                  Begin
                     For l := 1 to nxx Do
                        Begin
                           mt := kj1 - l + 1;
                           Swap1(xCof[mt],Cof[l]);
                        End;
                     Swap2(N,nx);
                     flag3 := Ifit = 0;
                     If flag3 Then
                        Begin
                           Ifit := 1;
                           xpr := x;
                           ypr := y;
                           Proc2;
```

```
                  Proc3;
               End;
            End
         Else
            Begin
               ict := ict + 1;
               If ict < 500 Then
                  Begin
                     Proc3;
                  End
               Else
                  If Ifit = 0 Then
                     If inn < 5 then
                        Begin
                           Proc1;
                           Proc2;
                           Proc3;
                        End
                     Else
                        Begin
                           writeln('Unable to determine Rc
                           Exit;
                        End
                  Else


                           Begin
                              mt := kj1 - l + 1;
                              Swap1(xCof[mt],Cof[l]);
                           End;
                           Swap2(N,nx);
                           flag3 := Ifit = 0;
                           If flag3 Then
                              Begin
                                 Ifit := 1;
                                 xpr  := x;
                                 ypr  := y;
                                 Proc2;
                                 Proc3;
                              End;
                     End;
               End;
            End;
      End;
End;

PROCEDURE Proc4;
VAR
   l : integer;
Begin
   Cof[2] := cof[2] + alpha * Cof[1];
   For l := 2 to N Do
      Cof[l+1] := Cof[l+1] + alpha * Cof[l] - sumsq * Cof[l-1];
   Repeat
      RootI[N2] := y;
      RootR[N2] := x;
```

```
         N2 := N2 + 1;
         flag4 := sumsq = 0;
         If not(flag4) Then
            Begin
               y := -y;
               sumsq := 0;
            End;
      Until flag4;
      flag5 := N > 0;
   End;

   PROCEDURE IfitNE0;
   Begin
      Ifit := 0;
      If (ABS(y) - 0.0001 * ABS(x)) < 0 Then
         Begin
             y := 0;
             sumsq := 0;
             alpha := x;
             N := N - 1;
             Proc4;
         End
      Else
         Begin
            alpha := x + x;
            sumsq := x * x + y * y;
            N := N - 2;
            Proc4;
         End;
   End;

   PROCEDURE SumsqEq0;
   Begin
      If Ifit = 0 Then
         Begin
            Proc1;
            Proc2;
            Proc3;
         End
      Else
         Begin
            x := xpr;
            y := ypr;
            IfitNE0;
         End;
   End;

   PROCEDURE UEQ0;
   Begin
      x    := 0;
      nx   := nx-1;
      nxx  := nxx-1;
      y    := 0;
      sumsq := 0;
      alpha := x;
      N := N - 1;
```

```
      Proc4;
   End;

Begin                    { Of Newton }
   u := 0;
   ux := 0;
   temp := 0;
   nx := N;
   nxx := N + 1;
   n2 := 1;
   kj1 := N + 1;
   For i := 1 to kj1 Do
      Begin
         mt := kj1 - i + 1;
         Cof[mt] := xCof[i];
      End;
   Repeat
      xo := 0.005;
      yo := 0.01;
      inn := 0;
      flag2 := false;
      flag3 := true;
      Repeat
         Proc1;
         Proc2;
         Proc3;
      Until (not(flag3)) or (flag1) or (flag2);
      If  (not(flag3)) Then
          IfitNE0
      Else
          If (flag1) and (flag2) Then
             SumsqEq0
          Else
             If flag1 Then
                UEq0;
   Until not(flag5);
End;                     { Of Newton }

PROCEDURE PrintRoot;
VAR
   i     : integer;
   yy    : real;
   flag1,flag2,flag3,flag4 : boolean;

   PROCEDURE Roots;
   Begin
      If RootI[i] = 0 then
         writeln('Z(',i,') = ',RootR[i]:8:3)
      Else
         If RootR[i] = 0 Then
            writeln('Z(',i,') = ',RootI[i]:8:3)
         Else
            If RootI[i] > 0 then
writeln('Z(',i,') = ',RootR[i]:8:3,'  +',RootI[i]:8:3,'i')
            Else
writeln('Z(',i,') = ',RootR[i]:8:3,'  -',ABS(RootI[i]):8:3,'i');
```

```
   End;

   PROCEDURE Proc5;
   Begin
      yy := 1000;
      Repeat
         flag2 := ABS(RootI[i] * yy) > 32000;
         If flag2 Then
            yy := yy / 10;
      Until not(flag2);
      If RootI[i] = 0 Then
         Roots
      Else
         Begin
            Repeat
               flag3 := ABS(RootI[i] * yy) < 1000;
               If flag3 Then
                  yy := yy * 10;
            Until not(flag3);
            RootI[i] := (RootI[i] * yy) / yy;
            Roots;
         End;
   End;

Begin
   For i:= 1 to M Do
      Begin
         yy := 1000;
         Repeat
            flag1 := ABS(RootR[i] * yy) > 32000;
            If flag1 Then
               yy := yy / 10;
         Until not(flag1);
         If RootR[i] = 0 Then
            Proc5
         Else
            Begin
               Repeat
                  flag4 := ABS(RootR[i] * yy) < 1000;
                  If flag4 Then
                     yy := yy * 10;
               Until not flag4;
               RootR[i] := (RootR[i] * yy) / yy;
               Proc5;
            End;
      End;
End;

PROCEDURE ZValues;
VAR
   i : integer;
Begin
   min := 1.7E38;
   max := 2.9E-39;
   For i := 1 to M Do
      Begin
```

```
            If RootR[i] < min Then
               min := RootR[i];
            If RootR[i] > max Then
               max := RootR[i];
            If RootI[i] <> 0 then
               Begin
                  writeln('Z is not real');
                  flag := false;
                  Exit;
               End;
        End;
End;

Begin           { Of NewtonOfPolynomial }
   xCof[1] := -A * B;             { Coefficient of order 0 }
   xCof[2] := A - B - SQR(B);     { Coefficient of order 1 }
   xCof[3] := -1;                 { Coefficient of order 2 }
   xCof[4] := 1;                  { Coefficient of order 3 }
   Ifit := 0;
   N := M;
   If xCof[N+1] = 0  then
      Begin
         writeln('High order coefficient can''t be zero');
         Exit;
      End;
   Newton;
   ZValues;
End;            { Of NewtonOfPolynomial }
End.            { Of Unit Newton2 }
```

```
{ FILE NAME : INITY.PAS }

PROCEDURE NormalizeY;
    VAR
       ii,jj,jv   : integer;
       sigma_y : real;
    Begin
       writeln('Y[i,j]');
       For jv := 1 to m Do
          write(comp[jv].name:10);
          writeln;
       For jj := 1 to N Do
          Begin
             write(jj:3);
             sigma_y := 0;
             For ii := 1 to m Do
                sigma_y := sigma_y + Y^.mtx[ii,jj];
             IF sigma_y <> 1 Then  { Normalized Y[ij] values }
                For ii := 1 to m Do
                   Begin
                      Y^.mtx[ii,jj]  := Y^.mtx[ii,jj]/sigma_y;
                      write(Y^.mtx[ii,jj]:10:4);
                   End;
             writeln;
          End;
    End;

PROCEDURE ComputeY;
{$I FUGACITY.PAS}
{$I CALTEMP.PAS}
VAR
   ii,i,j : integer;
   Y_pre  : Pointer;

   FUNCTION Power(x,n : real) : real;
   Begin
      Power := Exp(ln(x) * n);
   End;

   PROCEDURE CalYi;
   VAR
      i,j : integer;
   Begin
      For i := 1 to m Do
         For j := 1 to N Do
            Y^.mtx[i,j] := (Fugal^.mtx[i,j]*X^.mtx[i,j])
                            / Fugav^.mtx[i,j];
      NormalizeY;   { Normalized Y[ij] for each stage }
   End;

   PROCEDURE AdjustY;
   VAR
      ii,j,iv,jj : integer;
   Begin
      For j := 1 to N Do
         Begin
```

```
            For iv := 1 to m Do
              While (ABS(Y^.mtx[iv,j] - Y_pre^.mtx[iv,j])
                    >= 0.0001) Do
                 Begin
                    For ii := 1 to m do
                       For jj := 1 to N Do
                         Y_pre^.mtx[ii,jj] := Y^.mtx[ii,jj];
                    ComputeNewT(T_old);
                    CalYi;
                 End;
         End;
   End;

Begin
   clrscr;
   For i := 1 to m Do
      For j := 1 to N Do
         Y_pre^.mtx[i,j] := Y^.mtx[i,j];
   For j := 1 to N Do
      V[It,j] := V[It-1,j];
   ComputeNewT(T_old);
   CalYi;
   AdjustY;
End;
```

```
{ FILE NAME : HDEPT1.PAS }

PROCEDURE Ther_Hdept1;

   PROCEDURE SRK_const;
   VAR
   i,j,jj : integer;
   ab     : real;
   flag   : boolean;

   FUNCTION Power(x,n : real) : real;
   Begin
      Power := Exp(ln(x) * n);
   End;

   Begin
      For i := 1 to m Do
         Begin
            bi[i]  := (0.08644*R*Tc[comp[i].pos]) /
                      Pc[comp[i].pos];
            mm[i]  := 0.48 + 1.574*ww[comp[i].pos] -
                      0.176*SQR(ww[comp[i].pos]);
            ac[i]  := 0.42748*SQR(R)*SQR(Tc[comp[i].pos])/
                      Pc[comp[i].pos];
         End;
      For i := 1 to m Do
          Begin
            alpha^.mtx[i,1]  := 1+mm[i]*(1-Power
                                  ((Tf/Tc[comp[i].pos]),0.5));
            alpha^.mtx[i,1]  := alpha^.mtx[i,1]*alpha^.mtx[i,1];
            ai^.mtx[i,1]  := ac[i] * alpha^.mtx[i,1];
          End;
      bb[1]  := 0;
      aa[1]  := 0;
      For i := 1 to m Do
          Begin
             bb[1]  := bb[1]  + z^.mtx[i,Fj]*bi[i];
             ab := 0;
             For jj  := 1 to m Do
                 ab  := ab + z^.mtx[i,Fj]* z^.mtx[jj,Fj]*
                        Power(ai^.mtx[i,1]*ai^.mtx[jj,1],0.5);
             aa[1]  := aa[1]  + ab;
          End;
      A[1]  := (aa[1]*P[Fj]) / SQR(R*Tf);
      B[1]  := bb[1]*P[Fj]  / (R*Tf);
   End;

Begin
    Srk_const;
    NewtonOfPolynomial(1,min,max,A[1],B[1]);
    Zl[1]  := min;
End;
```

```
{ FILE NAME : GAUSS.PAS }

PROCEDURE Gauss(i   : integer);
PROCEDURE GaussElim;
VAR
   flag : boolean;
   ch   : char;

   PROCEDURE Pivoting(N   : integer;
                      i   : integer );
   VAR
      j,jc : integer;
      PV   : integer;      { Pivoting Index }
      Temp : real;
   Begin                   { of Pivoting }
      flag := true;
      PV := i;
      For j := i+1 to N Do
         If (ABS(Co^.mtx[PV,i]) < ABS(Co^.mtx[j,i])) Then
            PV := j;
      If (PV <> i) Then
         For jc := 1 to N+1 Do      { Swap the coefficient }
            Begin
               Temp := Co^.mtx[i,jc];
               Co^.mtx[i,jc] := Co^.mtx[PV,jc];
               Co^.mtx[PV,jc] := Temp;
            End
       Else
          If (Co^.mtx[i,i] = 0) Then
             Begin
               writeln('Matrix is inconsistent !');
               flag := false;
               Exit;
             End;
   End;                    { of Pivoting }

   PROCEDURE Eliminate(N   : integer);
   VAR
      i,jr,k : integer;
      factor : real;
   Begin                   { of Eliminate }
      For i := 1 to N-1 Do
         Begin
            Pivoting(N,i);
            { Elimination of below-diagonal }
            For jr := i+1 to N Do
               Begin
                  If (Co^.mtx[jr,i] <> 0) Then
                     Begin
                        factor := Co^.mtx[jr,i]/Co^.mtx[i,i];
                        For k := i+1 to N+1 Do
                           Co^.mtx[jr,k] :=
                           Co^.mtx[jr,k]-factor*Co^.mtx[i,k];
                     End;
               End;
         End;
```

```
   End;                          { of Eliminate }

   PROCEDURE Substitute( N   : integer;
                         i   : integer);
   VAR
      j,k : integer;
      Sum : real;
   Begin
      flag := true;
      If (Co^.mtx[N,N] = 0) Then
         Begin
            writeln('Set of equation is inconsistent !');
            flag := false;
            Exit;
         End;
      Co^.mtx[N,N+1] := Co^.mtx[N,N+1]/Co^.mtx[N,N];
      { Backward Substitution }
      For j := N-1 downto 1 Do
          Begin
             Sum := Co^.mtx[j,N+1];
             For k := j+1 to N Do
                 Sum := Sum-Co^.mtx[j,k]*Co^.mtx[k,N+1];
             Co^.mtx[j,N+1] := Sum/Co^.mtx[j,j];
          End;
      For j := 1 to N Do
         Begin
            X^.mtx[i,j] := Co^.mtx[j,N+1];
         End;
   End;

   PROCEDURE GaussPrint(VAR flag : boolean);
   VAR
      j,line : integer;
   Begin
      If flag Then
         Begin
            For j := 1 to N Do
               Begin
                  write(X^.mtx[i,j]:10:4,' ');
               End;
             writeln;
         End;
   End;

Begin                            { of GaussElim }
   Eliminate(N);
   If flag Then
      Substitute(N,i);
End;                             { of GaussElim }

Begin  { Main Program }
   GaussElim;
End;   { End Main Program }
```

```
{ FEED.PAS }

PROCEDURE Feed;
{$I CORELATE.PAS}
VAR
   i,j,kk,line,col : integer;
   b               : string[10];

   PROCEDURE Readchr;
   VAR
      regs : registers;
   Begin
      regs.AH := $0;
      intr($16,regs);
      ch := chr(regs.AX);
      Funckey := #0;
      if ch = #0 then
         Funckey := chr(regs.AH);
   End;

   PROCEDURE PositComp(j : integer;
                       i : integer;
                       VAR col   : integer;
                       VAR line  : integer;
                       VAR found : boolean);
   VAR
      jj      : integer;
   Begin
      found := false;
      jj := 1;
      While (jj <= MaxComp) and (not found) Do
         Begin
            If comp[i].name = name[jj] Then
               Begin
                  comp[i].pos := jj;
                  found   := true;
               End;
            Inc(jj);
         End;
      If (i mod 2) = 0 Then
         col := 41
      Else
         col := 7;
         If (jj > MaxComp) and (found = false) then
            Begin
               comp[i].name := '';
               gotoxy(col,line);  clreol;
            End;
   End;

   PROCEDURE ReadComp(VAR j    : integer;
                      VAR i    : integer;
                          m    : integer;
                      VAR col  : integer;
                      VAR line : integer);
   VAR
```

```
   k,l,a,b : integer;
   name    : ARRAY[1..LengthComp] of char;
   found   : boolean;
Begin
   Repeat
      b := 0;
      comp[i].name := '';
      Repeat
         readchr;
         Case funckey of
            Up,Down : gotoxy(col,line);
            Left    : if (col > 15) then
                      Begin
                         col := col - 1;
                         gotoxy(col,line);
                      End;
            Right   : if (col < 60) then
                      Begin
                         col := col + 1;
                         gotoxy(col,line);
                      End;
         Else
            Begin
               gotoxy(col,line);  write(ch);
               If (i mod 2) = 0 Then
                  k := (col-41)+1
               Else
                  k := (col-7)+1;
                  b := b+1;
                  name[k] := ch;
                  col := col+1;
               End;
            End;
      Until ch in[#13,#27];
      For a := 1 to b-1 Do
         Comp[i].name := Comp[i].name + name[a];
      PositComp(j,i,col,line,found);
   Until found = true;
End;

PROCEDURE Getproperties;
CONST
   blank = ' ';
VAR
   i      : integer;
   infile : text;
   str    : string[15];
Begin
   Assign(infile,'Crit20.txt');
   Reset(infile);
   For i := 1 to MaxComp Do
      name[i] := '';
   i := 0;
   While not(eof(infile)) Do
      Begin
         i := i + 1;
```

```
         str := '';
         Repeat
            read(infile,ch);
            if ch <> blank Then
               str := str + ch;
         Until ch = blank;
         name[i] := str;
         readln(infile,T50[i],API[i]);     { deg.F ,deg.API }
         SG[i] := 141.5/(API[i] + 131.5);
         GetCorelate(i);                   {  Tc,Pc,ww }
         Tc[i] := Tc[i]+460;               { deg.R}
      End;
   Close(infile);
End;

PROCEDURE GetVar;
VAR
   no, line : integer;
Begin
   gotoxy(4,6);  write('Stage');
   gotoxy(11,6); write('P(psia)   W(lbmole/h)   U(lbmole/h)');
   gotoxy(48,6); write('Q(Btu/h)   V(lbmole/h)');
   gotoxy(4,7);  write('==========================================');
                 write('==========================================');
   gotoxy(27,24);write('Press Stage=0 to Exit');
   {print to file}
   write(outfile,'Stage  P(psia)   W(lbmole/h)   U(lbmole/h)')
   writeln(outfile,'   Q(Btu/h)   V(lbmole/h)');
   write(outfile,'==========================================');
   writeln(outfile,'==========================================');
   line :=8;
   Repeat
      gotoxy(5,line);
      read(no);              { no. of Stage }
      IF no=0 Then
         Exit;
      gotoxy(11,line);  read(P[no]);
      gotoxy(23,line);  read(W[no]);
      gotoxy(37,line);  read(U[no]);
      gotoxy(50,line);  read(Q[no]);
      gotoxy(60,line);  read(V[It,no]);
      write(outfile,no:3,P[no]:10:2,W[no]:10:2,U[no]:14:2);
      writeln(outfile,Q[no]:13:2,V[1,no]:11:2);
      Inc(line);
   Until no=0;
   writeln(outfile);
   writeln(outfile,'Press Stage=0 to Exit');
End;

PROCEDURE DisplayZ;
VAR
   i,line,col : integer;
Begin
   line := 12;
   For i := 1 to m Do
      Begin
```

```
            gotoxy(55,line);  write(z^.mtx[i,Fj]:7:4);
            line := line + 1;
         End;
   End;

   PROCEDURE GetComp;
   VAR
      i : integer;
   Begin
      If m <> 0 Then
         Begin
           line := 5;
           i:=0;
           clrscr;
           gotoxy(10,2);
           write('Input Feed Composition');
           gotoxy(10,3);
           write('======================');
           gotoxy(5,4);
           write('Component           Lbmole/hr          ');
           write('Component           Lbmole/hr     ');
           Repeat
             If line = 21 Then
                Begin
                  clrscr;
                  gotoxy(10,2);
                  write('Input Feed Composition');
                  gotoxy(10,3);
                  write('======================');
                  gotoxy(5,4);
                  write('Component           Lbmole/hr          ');
                  write('Component           Lbmole/hr     ');
                  line :=5;
                End;
            col := 7;
            ch  := ' ';
            gotoxy(col,line);
            i:=i+1;
            ReadComp(Fj,i,m,col,line);
            gotoxy(23,line);  read(z^.mtx[i,Fj]);
            IF i = m Then
               Exit;
            col := 41;
            ch := ' ';
            gotoxy(col,line);
            i:=i+1;
            ReadComp(Fj,i,m,col,line);
            gotoxy(57,line);  read(z^.mtx[i,Fj]);
            line := line + 1;
           Until i = m
         End;
   End;

Begin          { of Feed }
   clrscr;
   Assign(outfile,'Input.txt');
```

```
Rewrite(outfile);
gotoxy(5,2);   write('N (No. of stage)         = ');
gotoxy(40,2);  write('B (kg-mole/h)            = ');
gotoxy(5,3);   write('Feed - ','Stage            = ');
gotoxy(40,3);  write('m (No. of comp.)         = ');
gotoxy(5,4);   write('Temp. of Feed (deg.F)    = ');
gotoxy(40,4);  write('Percentage Error of Tc = ');
gotoxy(5,5);   write('Percentage Error of Pc = ');
gotoxy(40,5);  write('Percentage Error of  w = ');
gotoxy(30,2);  read(N);
gotoxy(66,2);  read(Bottom);        { Read B }
Repeat
   gotoxy(30,3);  read(Fj);
Until Fj <= N;
Repeat
   gotoxy(66,3);  read(m);          { Read m (no. of comp.) }
Until m <= MaxComp;
gotoxy(30,4); read(Tf);             { deg.F }
Tf := Tf+460;
gotoxy(66,4);  read(eTc);
gotoxy(30,5);  read(ePc);
gotoxy(66,5);  read(eW);
{ Print to Input File }
write(outfile,'N (No. of stage)         = ');
write(outfile,n);
write(outfile,'            B (kg-mole/h)            = ');
writeln(outfile,bottom:4:2);
write(outfile,'Feed - ','Stage              = ',Fj);
writeln(outfile,'            m (No. of comp.)         = ',m);
write(outfile,'Temp. of Feed (deg.F)    = ',Tf:4:2);
writeln(outfile,'         Percentage Error of Tc = ',eTc:4:2)
write(outfile,'Percentage Error of Pc = ',ePc:4:2);
writeln(outfile,'          Percentage Error of  w = ',eW:4:2)
writeln(outfile);
GetVar;
For j := 1 to N do
   P[j] := P[1];
For i := 1 to MaxComp do
   Pb[i] := P[1];
Getproperties;
{print to file}
writeln(outfile);
writeln(outfile,'Input Feed Composition');
writeln(outfile,'======================');
writeln(outfile);
writeln(outfile,'Component            Lbmole/hr     z[ij]');
GetComp;
For i := 1 to m Do
    F[Fj] := z^.mtx[i,Fj] + F[Fj];
gotoxy(5,line+2);
write('Total Feed = ',F[Fj]:7:2,' Lbmole/hr');
ch := readkey;
For i := 1 to m Do
   Begin
      write(outfile,comp[i].name:5,z^.mtx[i,Fj]:19:2);
      z^.mtx[i,Fj] := z^.mtx[i,Fj] / F[Fj];
```

```
        writeln(outfile,z^.mtx[i,Fj]:12:4);
        X^.mtx[i,Fj]  := z^.mtx[i,Fj];
    End;
End;                    { of Feed  }
```

```
{ FILE NAME : FUGACITY.PAS }

PROCEDURE Fugacity(X   : Pointer;
                   ZZ  : Vector;
                   VAR Fuga : Pointer);
VAR
   i,j,jj    : integer;
   sigma,Ao,Bo : real;

   FUNCTION Power(x,n : real) : real;
   Begin
        Power := Exp(ln(x) * n);
   End;

Begin
   For i := 1 to m Do
      For j := 1 to N Do
         Begin
            B_prime^.mtx[i,j] := bi[i] / bb[j];
            sigma := 0;
            For jj := 1 to m Do
               sigma := sigma + (X^.mtx[jj,j] *
                             Power(ai^.mtx[jj,j],0.5));
            A_prime^.mtx[i,j] := 1/aa[j]
                              *(2*Power(ai^.mtx[i,j],0.5)
                              * sigma);
            Ao := aa[j]*P[j] / SQR(R*T[k,j]);
            Bo := bb[j]*P[j] / (R*T[k,j]);
            Fuga^.mtx[i,j] := -ln(ZZ[j] - Bo) + (ZZ[j]-1)
                           * B_prime^.mtx[i,j]-(Ao/Bo)
                           * (A_prime^.mtx[i,j] -
                           B_prime^.mtx[i,j])*ln(1+Bo/ZZ[j]);
            Fuga^.mtx[i,j] := EXP(Fuga^.mtx[i,j]);
         End;
End;
```

```
{ FILE NAME : CORELATE.PAS }

PROCEDURE GetCorelate(i : integer);
VAR
   Tbr,Pbr,Kw : Vector2;

   FUNCTION Power(x,n : real) : real;
   Begin
      Power := Exp(ln(x) * n);
   End;

   PROCEDURE Cavett(i : integer);
   VAR
      Exp2,Tc1,Tc2 : real;
   Begin
      Exp2 := T50[i]*T50[i] * API[i]*API[i];
      Tc1  := 768.071+1.7134*T50[i]-0.10834/100*T50[i]*T50[i]
              +0.3889/1000000*Power(T50[i],3)-0.89213/100
              *T50[i]*API[i];
      Tc2  := 0.53095/1000000*T50[i]*T50[i]*API[i]
              +0.32712/10000000*Exp2;
      Tc[i]  := Tc1+Tc2;      {deg.F}
      Tc[i]  := (1+eTc/100)*Tc[i];
      Pc[i]  := 2.829+0.9412/1000*T50[i]
                -0.30475/100000*T50[i]*T50[i]
                +0.15141/100000000*Power(T50[i],3)
                -0.20876/10000*T50[i]*API[i]+0.11048/10000000
                *T50[i]*T50[i]*API[i]+0.1395/1000000000*Exp2
                -0.4827/10000000*T50[i]*API[i]*API[i];
      Pc[i]  := Power(10,Pc[i]);
      Pc[i]  := (1+ePc/100)*Pc[i];
   End;

   PROCEDURE Lee_Kesler(i : integer);
   VAR
         w1,w2 : real;
   Begin
      Tb[i]  := T50[i]+460;       { deg.R }
      Tbr[i]  := Tb[i]/(Tc[i]+460);
      Pbr[i]  := Pb[i]/Pc[i];
      Kw[i]  := Power(Tb[i],1/3) / SG[i];
      If Tbr[i] < 0.8 Then
         Begin
            w1  := ln(Pbr[i])-5.92714+6.09648/Tbr[i]
                   +1.28862*ln(Tbr[i])-0.169347*Power(Tbr[i],6);
            w2  := 15.2518-15.6875/Tbr[i]-13.4721*ln(Tbr[i])
                   +0.43577*Power(Tbr[i],6);
            ww[i]  := w1/w2;
         End
      Else
         ww[i]  := -7.904+0.1352*Kw[i]-0.0007465*Kw[i]*Kw[i]
                   +8.359*Tbr[i]+(1.408-0.01063*Kw[i])/Tbr[i];
      ww[i]  := (1+eW/100)*ww[i];
   End;
Begin
   Cavett(i);
```

```
   Lee_Kesler(i);
End;
```

```
{ DISPLAY.PAS }

PROCEDURE ShowEnthal;
VAR
   j : integer;
Begin
   clrscr;
   writeln('Stage         H-Liquid              H-Vapor');
   writeln(outfile,'Stage         H-Liquid              H-Vapor');
   For j := 1 to N Do
      Begin
           writeln(j:3,Hl[j]:18:4,Hv[j]:18:4);
           writeln(outfile,j:3,Hl[j]:20:4,Hv[j]:20:4);
      End;
End;

PROCEDURE Show(Out  : Pointer;
               head : string);
VAR
   i,j : integer;
Begin
   clrscr;
   writeln(head);
   writeln(outfile,head);
   write('Stage');
   write(outfile,'Stage');
   For i := 1 to m Do
      Begin
           write(Comp[i].name:5);
           write(outfile,Comp[i].name:5);
      End;
   writeln;
   writeln(outfile);
   For j := 1 to N Do
      Begin
         write(j:3,'   ');
         write(outfile,j:3,'   ');
         For i := 1 to m Do
            Begin
               write(Out^.mtx[i,j]:7:4);
               write(outfile,Out^.mtx[i,j]:7:4);
            End;
         writeln;
         writeln(outfile);
      End;
   writeln(outfile);
   ch := readkey;
End;

PROCEDURE Display;
VAR
   i,j : integer;
Begin
    Assign(outfile,'result.txt');
    Rewrite(outfile);
    Repeat
```

```
    clrscr;
    writeln;
    write('Stage          T[0]         T[',It-1,']              V[0]')
    write(outfile,'Stage          T[0]         T[',It-1,']
    writeln('        V[',It-1,']');
    writeln(outfile,'        V[',It-1,']');
    write('               deg.F          deg.F       (lbmole)/h');
    write(outfile,'               deg.F          deg.F       (lbmole)/
    writeln('    (lbmole)/h');
    writeln(outfile,'    (lbmole)/h');
    For j := 1 to N Do
       Begin
          write(j:3,TT[1,j]-460:14:2,TT[It-1,j]-460:14:2);
          write(V[1,j]:14:2);
          writeln(V[It-1,j]:14:2);
          write(outfile,j:3,TT[1,j]-460:14:2,TT[It-1,j]-460:14:2
          writeln(outfile,V[1,j]:14:2,V[It-1,j]:14:2);
       End;
    writeln(outfile);
    ch := readkey;
    Show(X,'X[i,j]');
    Show(Y,'Y[i,j]');
    Show(K_val,'K[i,j]');
    ShowEnthal;
    gotoxy(10,22);
    writeln('Press Esc to exit...');
    gotoxy(55,22);   ch := readkey;
    Until Upcase(ch) in[Esc];
    Close(outfile);
End;
```

```
{ COMPUTEX.PAS }

PROCEDURE Tridg_Matrix;
VAR
   A,B,C,D     : Vector;
   DD          : Real;
   j           : integer;

   PROCEDURE ClearABCD;
   VAR
      j : integer;
   Begin
      For j := 1 to N Do
         Begin
            A[j] := 0;  B[j] := 0;
            C[j] := 0;  D[j] := 0;
            L[j] := 0;
         End;
   End;

   PROCEDURE A_Value;
   VAR
      j,i : integer;
      sigma,DD : real;
   Begin
      DD := V[It-1,1] + U[1];      { D = V1+U1 }
      For j := 2 to N-1 Do         { A[j], 2<=j<=N-1 }
         Begin
            sigma := 0;
            For i:= 2 to j-1 Do
               sigma := sigma + (F[i] - W[i] - U[i]);
            A[j] := V[It-1,j] + sigma - DD;
            L[j-1] := A[j];
         End;                      { A[N] }
      A[N] := V[It-1,N] + Bottom;
   End;

   PROCEDURE BCD_Value;
   VAR
      j : integer;
   Begin
      For j := 1 to N-1 Do         { B[j],C[j]  , 1<=j<=N-1 }
         Begin
            B[j] := -((V[It-1,j] + W[j]) * K_val^.mtx[i,j]
                     + A[j+1] + U[j]);
            C[j] := V[It-1,j+1] * K_val^.mtx[i,j+1];
            D[j] := -F[j] * z^.mtx[i,j];
         End;
      B[N] := -(V[It-1,N]* K_val^.mtx[i,N] + Bottom); { B[N] }
   End;

   PROCEDURE D_Value;
   VAR
      j : integer;
   Begin
      D[1] := 0;                           { D[1] }
```

```
      For j := 2 to N-1 Do                       { D[j] , 2<=j<=N-1 }
         D[j] := -F[j] * z^.mtx[i,j];
      D[N] := 0;                                 { D[N] }
   End;

   PROCEDURE InitMatrix;
   VAR
      i,j : integer;
   Begin
      For i := 1 to N Do
         For j := 1 to N+1 Do
            Co^.mtx[i,j] := 0;
   End;

   PROCEDURE GetValToMatrix;
   VAR
      i,j : integer;
   Begin
        For j := 1 to N Do
           Begin
              Co^.mtx[j,j] := B[j];
              If j <= N-1 Then
                 Begin
                      Co^.mtx[j+1,j] := A[j+1];
                      Co^.mtx[j,j+1] := C[j];
                 End;
              Co^.mtx[j,N+1] := D[j];
           End;
   End;

   PROCEDURE PrnMatrix;
   VAR
      i,j : integer;
   Begin
       clrscr;
       For i := 1 to N Do
           Begin
               For j := 1 to N+1 Do
                  write(Co^.mtx[i,j]:7:2);
               writeln;
           End;
   End;
Begin
   ClearABCD;
   A_Value;
   BCD_Value;
   InitMatrix;
   GetValToMatrix;
End;

PROCEDURE NormalizeX;
VAR
   ii,jj,jv : integer;
   sigma_x  : real;
Begin
   writeln('X[i,j]');
```

```
   For jv := 1 to m Do
      write(comp[jv].name:10);
   writeln;
   For jj := 1 to N Do
      Begin
         write(jj:3);
         sigma_x := 0;
         For ii := 1 to m Do
            sigma_X := sigma_X + X^.mtx[ii,jj];
         If sigma_X <> 1 Then    { Normalized X[ij] values }
            For ii := 1 to m Do
               Begin
                  X^.mtx[ii,jj] := X^.mtx[ii,jj]/sigma_X;
                  write(X^.mtx[ii,jj]:5:4);
               End;
            writeln;
      End;
End;

PROCEDURE ComputeX;
{$I GAUSS.PAS}
VAR
   ii,j : integer;
Begin
  clrscr;
  For ii := 1 to m Do
     For j := 1 to N do
        X_pre^.mtx[ii,j] := X^.mtx[ii,j];
  For i := 1 to m Do      { Compute X[ij] }
     Begin
        Tridg_Matrix;
        Gauss(i);         { Call from Gauss subprog. }
     End;
  NormalizeX;             { Normalized X[ij] for each stage }
End;
```

```
{ COMPUTEH.PAS }

PROCEDURE ComputeEnthal;
{$I THER-PRO.PAS}

VAR
   H         : Pointer;
   HDF,TdaF : real;
   Tda, TotalH,HDL,HDV       : Vector;
   HIdealF,HIdealL,HIdealV   : Vector;

   FUNCTION Power(x,n : real) : real;
   Begin
      Power := Exp(ln(x) * n);
   End;

   PROCEDURE WriteEnthal;
   VAR
      i,j : integer;
   Begin
      clrscr;
      writeln('Stage         H-Liquid      H-Vapor');
      For j := 1 to N Do
         writeln(j:3,Hl[j]:18:4,Hv[j]:12:4);
   End;

   PROCEDURE InitH2;
   VAR
      j : integer;
   Begin
      New(H);
      H^.ptr := Nil;
      For j := 1 To N Do
         Begin
            Hl[j] := 0;
            Hv[j] := 0;
         End;
      Hf:=0;
   End;

   PROCEDURE Thermo_Feed;

       PROCEDURE SRK_const;
       VAR
       i,j,jj : integer;
       ab     : real;
       flag   : boolean;

       FUNCTION Power(x,n : real) : real;
       Begin
          Power := Exp(ln(x) * n);
       End;

       Begin
          For i := 1 to m Do
             Begin
```

```
            bi[i] := (0.08644*R*Tc[comp[i].pos]) /
                        Pc[comp[i].pos];
            mm[i] := 0.48 + 1.574*ww[comp[i].pos] -
                        0.176*SQR(ww[comp[i].pos]);
            ac[i] := 0.42748*SQR(R)*
                        SQR(Tc[comp[i].pos])/Pc[comp[i].pos];
         End;
      For i := 1 to m Do
         Begin
            alpha^.mtx[i,1] := 1 + mm[i]*(1-Power(
                               (Tf/Tc[comp[i].pos]),0.5));
            alpha^.mtx[i,1] := alpha^.mtx[i,1]*
                               alpha^.mtx[i,1];
            ai^.mtx[i,1] := ac[i] * alpha^.mtx[i,1];
         End;
      bb[1] := 0;
      aa[1] := 0;
      For i := 1 to m Do
         Begin
            bb[1] := bb[1] + z^.mtx[i,Fj]*bi[i];
            ab := 0;
            For jj := 1 to m Do
               ab := ab + z^.mtx[i,Fj]* z^.mtx[jj,Fj]*
                     Power(ai^.mtx[i,1]*ai^.mtx[jj,1],0.5);
            aa[1] := aa[1] + ab;
         End;
      A[1] := (aa[1]*P[Fj]) / SQR(R*Tf);
      B[1] := bb[1]*P[Fj] / (R*Tf);
   End;

Begin
   SRK_const;
   NewtonOfPolynomial(1,min,max,A[1],B[1]);
   Zl[1] := min;
End;

PROCEDURE WriteY;
VAR
   i,j : integer;
Begin
   clrscr;
   writeln('Y^.mtx[i,j]');
   write('Stage');
   For i := 1 to m Do
      write(comp[i].name:8);
   writeln;
   For j := 1 to N Do
      Begin
         write(j:3);
         For i := 1 to m Do
            write(Y^.mtx[i,j]:10:4);
         writeln;
      End;
End;

PROCEDURE ComputeH(j      : integer;
```

```
                        Ttran : real);
VAR
    A,B,C,A_prime,B_prime,C_prime : real;
    CF,deltaT1,deltaT2,deltaT3    : real;
    KK : real;
    i  : integer;
Begin
   For i := 1 to m do
      Begin
         KK := Power(Tb[comp[i].pos],1/3)/SG[comp[i].pos];
         A  := -0.32646+0.02678*KK;
         B  := -(1.3892-1.2122*KK+0.03803*KK*KK)/10000;
         C  := -1.5393/10000000;
         CF := SQR((12.8/KK-1)*(10/KK-1)*100);
         A_prime := -0.0084773+0.080809*SG[comp[i].pos];
         B_prime := (2.1773-2.0826*SG[comp[i].pos])/10000;
         C_prime := -(0.78649-0.70423*SG[comp[i].pos])
                    /10000000;
         deltaT1 := Ttran-Tref;
         deltaT2 := SQR(Ttran)-SQR(Tref);
         deltaT3 := SQR(Ttran)*Ttran-SQR(Tref)*Tref;
         H^.mtx[i,j] := A*deltaT1+B/2*deltaT2+C/3*deltaT3
                        +CF*(A_prime*deltaT1
                        +B_prime/2*deltaT2
                        +C_prime/3*deltaT3);
      End;
End;

PROCEDURE ComputeHIdeal(j   : integer;
                        X   : pointer;
                        VAR  HIdeal : vector);
VAR
   i     : integer;
   sigma : real;
Begin
   sigma := 0;
   For i:= 1 to m Do
      HIdeal[j] := sigma + X^.mtx[i,j]*H^.mtx[i,j];
End;

PROCEDURE TdabydT(X : Pointer);
VAR
   i,j,jj    : integer;
   Tr,sigma : real;
Begin
   For j := 1 to N Do
      Begin
         Tda[j] := 0;
         For i := 1 to m Do
            Begin
               sigma := 0;
               For jj := 1 to m Do
                  Begin
                     Tr := TT[It-1,j]/Tc[comp[jj].pos];
                     sigma := sigma + X^.mtx[i,j]*
                     X^.mtx[jj,j]* mm[jj]*
```

```
                        Power(ai^.mtx[i,j]*ac[jj]*Tr,0.5);
                    End;
                Tda[j] := Tda[j] + sigma;
            End;
      End;
End;

PROCEDURE TdabydTFeed;
VAR
     i,jj   : integer;
     Tr,sigma : real;
Begin
     TdaF := 0;
     For i := 1 to m Do
        Begin
            sigma := 0;
            For jj := 1 to m Do
               Begin
                   Tr := Tf/Tc[comp[jj].pos];
                   sigma := sigma+Z^.mtx[i,Fj]*Z^.mtx[jj,Fj]*
                   mm[jj]*Power(ai^.mtx[i,1]*ac[jj]*Tr,0.5);
               End;
            TdaF := TdaF + sigma;
        End;
End;

PROCEDURE HDeptFeed;
VAR
    val1,val2 : real;
Begin
    val1 := (A[1]/B[1])*(1+TdaF/aa[1]);
    val2 := ln(1+(B[1]/Zl[1]));
    HDF := (R*Tf)*((Zl[1]-1) - val1*val2);
End;

PROCEDURE HDepart(Ztran : Vector;
              VAR Htran : Vector);
VAR
    j          : integer;
    val1,val2 : real;
Begin
    For j := 1 to N Do
        Begin
            val1 := (A[j]/B[j])*(1+Tda[j]/aa[j]);
            val2 := ln(1+(B[j]/Ztran[j]));
            Htran[j]:=(R*TT[It-1,j])*((Ztran[j]-1)-val1*val2);
        End;
End;

PROCEDURE Enthalpy;
VAR
    j : integer;
Begin
    InitH2;
    Thermo_Feed;
    TdabydTFeed;
```

```
        HDeptFeed;
        Thermodynamic(X,Zl,'X');
        ComputeH(1,Tf);
        ComputeHIdeal(1,z,HIdealF);
        Hf := HDF+HIdealF[1];
        TdabydT(X);
        HDepart(Zl,HDL);
        For j := 1 to N do
           Begin
              ComputeH(j,TT[It-1,j]);
              ComputeHIdeal(j,X,HIdealL);
              Hl[j]  := HDL[j]+HIdealL[j];
           End;
        Thermodynamic(Y,Zv,'Y');
        TdabydT(Y);
        HDepart(Zv,HDV);
        For j := 1 to N do
           Begin
              ComputeH(j,TT[It-1,j]);
              ComputeHIdeal(j,Y,HIdealV);
              Hv[j]  := HDV[j]+HIdealV[j];
           End;
        WriteEnthal;
        Dispose(H);
     End;

  Begin      { of ComputeEnthal }
     Enthalpy;
  End;       { end ComputeEnthal }
```

```
{ FILE NAME : CALTEMP.PAS }

PROCEDURE ComputeNewT(VAR T_old : Vector3);
VAR
   S,SD,KK               : real;
   j,i                   : integer;
   flag                  : boolean;
   convergedT            : boolean;
   Ul,Uv,Sl,Sv,ABl,ABv   : real;

PROCEDURE Func;
   VAR
      flag               : boolean;
      i, jj              : integer;
      c,d,sigma,Ao,Bo    : real;
   Begin
      Thermodynamic(X,Zl,'X');
      Fugacity(X,Zl,Fugal);
      For i := 1 to m Do
         Begin
            Ul  := (P[j] * bb[j]) / (R*T[k,j]*Zl[j]);
            ABl := (aa[j]/(R*T[k,j]*bb[j]))
                   * (A_prime^.mtx[i,j]-B_prime^.mtx[i,j]);
            Sl  := Ul/(1+Ul) + ln(1+Ul);
        End;
      Thermodynamic(Y,Zv,'Y');
      Fugacity(Y,Zv,Fugav);
      For i := 1 TO m Do
         Begin
            Uv  := (P[j] * bb[j]) / (R*T[k,j]*Zv[j]);
            ABv := (aa[j]/(R*T[k,j]*bb[j]))
                   * (A_prime^.mtx[i,j]-B_prime^.mtx[i,j]);
            Sv  := Uv/(1+Uv) + ln(1+Uv);
         End;
      S := 0;
      For i := 1 to m Do
         Begin
            K_val^.mtx[i,j]:=Fugal^.mtx[i,j]/Fugav^.mtx[i,j];
            S := S + (K_val^.mtx[i,j] * X^.mtx[i,j]);
         End;
      S := S - 1.0;
   End;

   PROCEDURE DiffFunc;
   VAR
      i, jj                    : integer;
      val1,val2,val3,val4      : real;
      KD,c,d,sigma,Ao,Bo,Br    : real;
   Begin
      SD := 0;
      For i := 1 to m Do
         Begin
            KD  := Uv/(1-Uv)-Ul/(1-Ul)+(ABl*Sl)-(ABv*Sv);
            KD  := KD/T[k,j];
            SD  := SD + (K_val^.mtx[i,j]*KD*X^.mtx[i,j]);
         End;
```

```
    End;

    PROCEDURE Newton_Method;              { Adjust T[j] }
    VAR
       T_pre : real;                      { Previous value of T }
    Begin
       clrscr;
       writeln('T[j-1]':20,'Sum[j-1]':13,'T[j]':12);
       Repeat
          Func;
          DiffFunc;
          T_pre := T[k,j];
          T[k,j] := (T[k,j] - S/SD);
          writeln('It = ',It-1,' j = ',j,T_pre:10:4,S:11:4,
                     T[k,j]:15:4);
       Until ABS(T[k,j] - T_pre) < 0.001;
    End;
Begin           { Of ComputeNewT }
    k := 1;
    For j := 1 To N Do
       Begin
          Func;
          While (ABS(S) >= 0.00001) Do
             Newton_method;
       End;
    For j := 1 to N Do
       TT[It-1,j] := T[k,j];
End;            { Of ComputeNewT }
```

VITA

Mr. Ruangridh Wongwandanee graduaged high school from Prapathom Wittayalai in 1985 and received a Bachelor Degree in Chemical Engineering from the Department of Chemical Engineering, Faculty of Engineering, Prince of Songkla University in 1990. After then he subsequently studied for a requirement of the Master's Degree in Chemical Engineering at the Department of Chemical Engineering, Faculty of Engineering, Chulalongkorn University from 1992 till 1994.

He also has an experience in working as facility engineer at AMD (Thailand) Co., Ltd. in 1990-1991.