8-AW

SPE

Society of Petroleum Engineers

SPE 20327

General Computerized Well Control Kill Sheet for Drilling Operations With Graphical Display Capabilities

H.C.F. Leitão Jr. and E.E. Maidla, Unicamp; A.T. Bourgoyne, Louisiana State U.; and A.F. Negrão, Petrobrás

SPE Members

This paper was prepared for presentation at the Fifth SPE Petroleum Computer Conference held in Denver, Colorado, June 25–28, 1990.

ABSTRACT

A rigorous method of pressure calculations was used in a general well control procedure for drilling operations with the drillbit on bottom. A computer program was developed to handle the calculations and graphics interactively to allow speed and flexibility of choices throughout a kick situation. A well control procedure was developed to handle from vertical to horizontal wells for many drilling situations. All pertinent equations are presented and unlike other publications the computer source code is given in the Appendix.

INTRODUCTION

Although the vast majority of kicks are controlled, much research is still to be done to investigate the different scenarios of kick occurrence and the flow behavior while pumping it out, considering the different fluids in the well (while taking the kick) and its spatial configuration.

In particular, for drilling, two are the commonly accepted methods for well control operations: the Driller's Method and the Wait and Weight Method [1].

The Driller's Method consists in circulating out the kick before weighting up the drilling fluid while the Wait and Weight Method consists in circulating out the kick after weighting it up. Simultaneous circulating and weighting up the drilling fluid have also been studied but it is not of common use due to the additional accounting required. A qualitative example for the drillpipe pressure schedules for all three methods are shown in Fig. 1.

To aid the rig personnel, several kill sheets have been developed [2, 3], most of which consider a vertical well, a uniform distribution of pressure loss within the drillstring (from the bit, throughout the drillstring to the pump) and that the pressure loss changes, after weighting up the drilling fluid, are solely dependent upon its final density. Furthermore, it is a common practice in the industry today to periodically train the rig personal on well control practices (this is particularly true for offshore operations) that include among other training, to fill out correctly the kill sheets to avoid incorrect calculations during actual well control operations. This training to fill out the kill sheets could take a day or two depending on the different scenarios investigated (land rigs, floating vessels, etc).

The objective of this work was to develop a general and rigorous method of well control that can be easily used for any king of rig during drilling operations with the bit on bottom, and to automate the calculations through the use of a computer program that also handles interactive graphics with all necessary information for circulating out the kick.

RIGOROUS METHOD

This method considers all the information for usual well control operations (reduced circulating pressures, borehole and pipe geometries, mud properties, etc.) and also makes use of the rheological properties of the new drilling fluid, and the directional survey data to determine the true vertical depths (see tables 1 through 3 for details).

The method is based on the following procedure:

First the mud properties (rheology and density), borehole and pipe geometry, and flow rates are used to calculate the pressure loss through the surface equipment [4] (that lies after the surface pressure gage), the drillstring (drillpipe, heavy weight, collars, etc.), the bit, and throughout the different annular sections for the old mud in the well (Fig. 2).

Then the circulating pressure at the kill rate $(P_r)_m$ mea-

References and illustrations at end of paper sured prior to taking the kick, is used to determine a correction factor (c_f) that is calculated as:

$$c_f = \frac{(P_r)_m}{(P_r)_c} \qquad (1)$$

where $(P_r)_c$ is the calculated reduced pressure at kill rate, and c_f accounts for all modeling errors and unknown variables disconsidered by the pressure loss equations.

This correction factor is used to calculate the pressure loss for the new drilling fluid.

A hydrostatic pressure schedule is then calculated for each stroke for both old and new drilling fluids using the average angle method to calculate the vertical depths (reason wy the azimuth readings are not neede, as seen in table 4).

The drillpipe pressure schedule P_{dp} is then calculated for killing the well by correcting the shut in drillpipe pressure reading for pressure loss (due to changes in drilling fluid rheology) and hydrostatic pressure changes (due to drilling fluid density changes), Fig. 3.

$$(P_{dp})_i = (P_{dp})_s + (P_{ds})_c + (\Delta P_f)_i - (\Delta P_h)_i \qquad (2)$$

The next step is to display the results in graphic and table form as will be shown in the numerical examples.

The procedure outlined above showed that the well was subdivided into sections and each one studied separately as regards to pressure changes due to hydrostatic and frictional effects. A quantitative measure of the quality of the pressure predictions can be made by comparing calculated pressure loss to the reduced circulating pressure measured at kill rate and calculate c_f. If c_f is close to unity, this means that good predictions were achieved. Evidently this implies in good pressure loss predictions and this can only be obtained through the use of good rheological models used appropriately.

PRESSURE LOSS CALCULATIONS

The Power-Law rheological model was used for pressure loss calculations because it can easily handle from two to six readings of the FANN viscometer 35-A commonly used in the field.

The calculations where performed according to the following:

1) FANN readings of rotational speed and angular deflections were transformed to shear stress and shear rate using the following relationships:

$$\tau = 5.1405\Theta \qquad (3)$$

$$\dot{\gamma} = 1.703\omega \qquad (4)$$

Although the above relationships were derived for Newtonian fluids, they show to be acceptable approximations for some Non-Newtonian fluids also. Just to illustrate this point, the FANN viscometer data of Table 4, for the old mud, was used and the shear rate was calculated using the above equations and the ones suggested by Yang and Krieger [5]. The results showed that the shear rate errors were: 5.29%, 3.45%, 1.30%, 2.10%, 2.60%, 3.42% for ω values of 3, 6, 100, 200, 300, 600 rpm, respectively. These errors didn't however produce great differences in the calculations of K and n.

2) Using the minimum square method, the best straight line was fitted to the plot of $\log(\tau)$ and $\log(\dot{\gamma})$ to determine K (that is found in $dynesec^n/cm^2$ and is mutliplied by 100 to obtain eq-cp) and n.

3) Based on the pipe of annulus geometry, and flow rate, the average velocity was calculated:

Mean velocity for the pipe:

$$v = \frac{q}{2.448d^2} \qquad (5)$$

where:
v = velocity [ft/s]
q = flow rate [gal/min]
d = pipe diameter [in]

Mean velocity for the annulus:

$$v = \frac{q}{2.448({d_2}^2 - {d_1}^2)} \qquad (6)$$

Based on Dodge and Metzner's [6] work the following turbulence flow correlation criteria was used:

$$N_{Re} = \frac{89100\rho v^{(2-n)}}{k}\left(\frac{0.0416d}{3+\frac{1}{n}}\right)^n \qquad (7)$$

$$N_{Re} = \frac{109000\rho v^{(2-n)}}{k}\left(\frac{0.0208(d_2 - d_1)}{2+\frac{1}{n}}\right)^n \qquad (8)$$

where:
N_{Re} = Reynolds number [d-less]
ρ = Drilling fluid density [lb/gal]

The critical Reynolds number was obtained from Dodge and Metzner correlation and simplified to the following values:

for $n < 0.2$ $\quad (N_{Re})_{cr} = 4200$

for $0.2 \le n \le 0.45$ $\quad (N_{Re})_{cr} = 5960 - 8800n$

for $n > 0.45$ $\quad (N_{Re})_{cr} = 2000$

The turbulent criteria was such: When the Reynolds number exceeded the critical Reynolds number, turbulent flow was assumed, otherwise laminar flow was assumed. In using such a criteria to distinguish laminar from turbulent flow, there will be a discontinuity in the pressure loss calculations as immediatly before the critical Reynolds number one set of equations are used, and immediatly after, another set are used. This can be observed during the drillpipe start up schedule when turbulent flow is achieved: first there is a linear pressure increase (for vertical wells), corresponding to the laminar flow equations, followed by a discontinuity in pressure and a quadratic pressure response corresponding to the turbulent flow equations. Obviously this discontinuity could be eliminated by changing the laminar to turbulent flow criteria, by calculating the pressure loss using both equations and always selecting the highest pressure loss, nevertheless the authors prefered to use Dodge and Metzner's criteria described above.

The frictional pressure loss was then calculated as such [7]:

For laminar flow in pipes:

$$\left(\frac{dP}{dD}\right)_f = \frac{kv^n\left(\frac{3+\frac{1}{n}}{0.0416}\right)^n}{144000d^{(1+n)}} \quad (9)$$

For laminar flow in the annulus:

$$\left(\frac{dP}{dD}\right)_f = \frac{kv^n\left(\frac{2+\frac{1}{n}}{0.0208}\right)^n}{144000(d_2-d_1)^{(1+n)}} \quad (10)$$

For turbulent flow in pipes:

$$(\frac{dP}{dD})_f = \frac{f\rho v^2}{25.8d} \quad (11)$$

For turbulent flow in the annulus:

$$(\frac{dP}{dD})_f = \frac{f\rho v^2}{21.1(d_2-d_1)} \quad (12)$$

where f is given by:

$$\sqrt{\frac{1}{f}} = \frac{4.0}{n^{0.75}}\log\left(N_{Re}f^{(1-\frac{n}{2})}\right) - \frac{0.395}{n^{1.2}} \quad (13)$$

INTERPRETATION OF THE CASING SURFACE PRESSURE GAGE READINGS

The casing surface pressure measurement limitations for well control operations is qualitatively illustrated in Fig. 4. To avoid exceeding the burst rating pressure at point A, the kill line pressure surface gage should be used as the choke line could be full of drilling fluid, gas and drilling fluid, or only gas, and therefore a maximum allowable choke line pressure would not be an accurate estimate of the pressure at point A, as it depends on these fluids and their contribution to hydrostatic and frictional pressure loss. Just as an illustration, at 5118 ft water depth (which has already been drilled in Brazil) the hydrostatic change alone totals about 2660 psi if we consider the replacement of a 10 lb/gal mud by gas and disconsidered the hydrostatic contribution of the gas column. In this situation, if the choke operator would have the information of maximum allowable surface choke pressure calculated with the drilling fluid in the choke line, he could open unnecessarily the choke while gas enters the choke line and allow for further kicks. The same logic applies to the choke operator that was given a maximum allowable casing gage pressure base on the choke line full of gas and therefore could burst the casing while pumping out the kick with the choke line full of drilling fluid.

A similar problem, but much harder to solve, is the knowledge of the casing surface pressure that should not be exceeded to avoid fracturing the weakest formation below the casing shoe. Again the scenarios are similar, but this time there is no static column of liquid to measure the pressure at that point.

COMPUTER PROGRAM

A computer program was written in Turbo C to handle all the calculations and display the necessary information in an interactive graphic mode. The C language was chosen basically because of its graphic capabilities allowing the user to run the executable files on any Pc, under DOS, with all commonly used graphic cards (Hercules and compatible, CGA, EGA, VGA).

A listing of the source code is given in Appendix A.

Similar to any kill sheet, the program will start by requesting all pre-kick information as listed in Table 1. It can handle a variety of different situations. A pre-kick information diagram is shown in Figs 5 and 6, for land rigs and deep water floating vessels respectively. Although both diagrams appear vertically all depths shown refer to measured depths. Some of the data requested is not used by the main program as is the case of the formation pressure and its respective depth, as to use this to calculate a maximum allowable surface gage pressure could be quite erroneous as discussed previously. Nevertheless, it is on file to be used in the future.

For the next step the program requests kick information as shown in table 2, after which it calls for the new mud properties as shown in table 3.

The main program then calculates two pressure schedules: one for the drill pipe pressure, and one for the casing pressure (for start up operations that can become critical for well control in deep waters). It then displays this information in a graphic form as shown in Fig. 7. At this point, there are several options for the user that can:

a) follow through with the kill procedure using the default graph shown on the screen. This graph displays the drillpipe pressure against the number of strokes. Many times the table form is easier to use and therefore is shown on the right hand side of the screen.

b) zoom in on part of the graph (Fig. 7) for a better resolution.

c) display the casing pressure schedule that will be necessary for starting up the pump while controlling a kick on a floating vessel.

d) select the drillpipe pressure schedule graph with the complete history of all drilling fluids used to control the well (for the case of a simultaneous well control procedures).

NUMERICAL EXAMPLE

A horizontal well drilled from a floating vessel was chosen, as the numerical example, to illustrate the drillpipe pressure behavior in horizontal wells.

The data used in this example is shown in Tables 4 and 5 and Fig. 08, that also includes the number of strokes necessary to pump the drilling fluid through each drillstring section, to help the interpretation of the graphs.

The first step was to feed in all data up to the properties of the old mud. The program then informed that the kill mud weight would have to be at least 10.8 lb/gal (with no safety margin included).

A weight of 9.5 lb/gal was then selected to illustrate the simplicity of choosing a simultaneous well control method since the computer program handles all accounting easily.

The drillpipe pressure schedule for this situation is shown in Fig 09. Notice that the program provides a numerical table besides the graphic display; furthermore it informs the number of strokes to reach the bit and provides several other display possibilities through a menu listing at the bottom of the graph. Analyzing the graph itself, the following can be observed:

a) To achieve the steady state stroke rate of 40 spm, the pump took 80 strokes (that was an input to the program) or 2 minutes, for which the drillpipe pressure schedule went from 810 to 1466 psi.

b) A pressure decline is observed from 80 to about 940 strokes after which pressure starts increasing due to the effect of the new mud entering the horizontal section that produces higher pressure loss than the old mud being displaced. This effect can be better observed while zooming in on strokes 80 to 1200 as shown in Fig. 10. This behavior is characteristic of horizontal wells and is quite different from the conventional experience with vertical systems.

c) While the new drilling fluid approaches the bit, there are several gradient changes and one discontinuity shown on the graph (better seen in Fig. 10), that show clearly the effect of the new mud entering the heavy weight drillpipe (after $\approx$ 888 strokes), the drill collars (after $\approx$ 998 strokes), the other heavy weight drillpipe (after $\approx$ 1021 strokes), the directional equipment (after $\approx$ 1167 strokes) and the bit, these last two can only be seen in Fig. 11 that zoom's in on 1153 to 1177 strokes.

d)The start up schedule for the surface pressure gage (Fig. 12)shows a discontinuity due to the change of laminar to turbulant flow equations as previouly discussed.

In addition two this drilling fluid, two other drillings fluids where also used: one of 10.5 lb/gal (that is still below the kill mud weight suggested of 10.8 lb/gal) that was introduced after 400 strokes, and another one of 11.4 lb/gal introduced after 800 strokes.

The combined effect of all three drilling fluids is shown in Figs 13 and 14. Again a rather unusual drillpipe pressure schedule is displayed due to the directional nature of the well and the drillstring sections.

CONCLUSIONS

The computer program showed to be adequate and flexible allowing to monitor many scenarios quite easily, and that unusual results were obtained for horizontal wells showing the need of including the directional profile in the well control procedure.

NOMENCLATURE

c_f = correction factor [d-less]
d = diameter [in]
D = depth [ft]
f = friction factor [d-less]
K = consistency index [eq-cp]
n = flow behavior index [d-less]
P = pressure [psi]
P_f = hydrodynamic pressure loss [psi]
P_h = hydrostatic pressure [psi]
P_{dp} = drillpipe pressure [psi]
P_{ds} = drillsting pressure loss (from the bit to the pump)
P_r = reduced pressure [psi]
q = flow rate [gal/min]
v = mean velocity [ft/s]
$\dot{\gamma}$ = shear rate [s^{-1}]
Θ = FANN deflection [deg.]
ρ = density [lb/gal]
τ = shear stress [$\frac{dyne}{cm^2}$]
ω = FANN speed [rpm]

subscripts

c = calculated
c_r = critical
f = hydrodynamic friction
i = given stroke number
m = measured
s = shut in
cr = critical

ACKNOWLEDGEMENTS

The authors would like to thank the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), for its support. The authors would also like to thank Professor Cesar Costapinto Santana (Unicamp - Chemical Engineering Department), Ana Eleonora Paixão and Paulo de Tarso Vieira e Rosa (Unicamp - Chemical Engineering Department), Carlos Henrique Marques de Sa (Unicamp - Petrobras), Weimar Lazaro (Unicamp - Petrobras), Marcelo Matheus (Unicamp), for all their help and fruitful discussions. Armando Arruda's (Gepron - Unicamp) help in preparing some of the figures is also greatly appreciated. The authors are specially thankfull to Fabio de Andrade Netto for all his assistance in developing the graphic software used in the computer program.

REFERENCES

1. Moore, Preston L., Drilling Practices Manual, second edition, Pennwell Publishing Company, Tulsa, Oklahoma pp. 515-545, 1986.

2. Oliveira, Paulo C.; Arruda, Augusto M. P.; Negrão, Alvaro F., "Prevenção e Controle de Kicks," Petrobras internal publication, Sedes, 1988, (in Portuguese).

3. Louisiana State University Well Control School Manual, Baton Rouge, Louisiana.

4. Hughes Practical Hydraulics, Hughes Tools Company, Houston, Texas, 1976.

5. Yang, T. M. T.; Krieger, I., M., "Comparison of Methods for Calculating Shear Rates in Coaxial Viscometers," The Society of Theology Inc., John Wiley & Sons Inc., Journal of Rheology, pp. 413-421, 1978.

6. Dodge, D. G.; Metzner, A. B., "Turbulent Flow of Non-Newtonian Systems," AIChE J., 5, 189, 1959.

7. Bourgoyne Jr., A. T.; Chenevert, M. E.; Milheim, K. K.; young Jr., F. S., Applied Drilling Engineering, SPE textbook series, Vol 2, pp152-155, 1986.

Appendix.

```
/*=======================================================================*/
/*                                                                       */
/*  Program module: Well Control Sheet     ( CTRLKICK.PRJ )              */
/*                                                                       */
/*  Used to compile the progam and prepare the graphics hard             */
/*                                                                       */
/*  By:By Fabio / Helio / Eric - DEP - UNICAMP                           */
/*                                                                       */
/*  Date: 03/21/1990                                                     */
/*                                                                       */
/*=======================================================================*/
c:\tc\lib\graphics.lib
c:\tc\cga.obj
c:\tc\egavga.obj
c:\tc\herc.obj
c:\tc\litt.obj
ctrlkick.c
```

```
/*=========================================================================*/
/*                                                                         */
/*  Program module: Well Control Sheet        ( PREPARE.BAT )              */
/*                                                                         */
/*  Used to input information and calculation of pressure schedule         */
/*                                                                         */
/*  By: Fabio / Helio / Eric - DEP - UNICAMP                               */
/*                                                                         */
/*  Date: 03/21/1990                                                       */
/*                                                                         */
/*=========================================================================*/
copy kickgraf.c c:\tc\include\kickgraf.c
bgiobj c:\tc\cga  c:\tc\cga
bgiobj c:\tc\egavga c:\tc\egavga
bgiobj c:\tc\herc c:\tc\herc
bgiobj c:\tc\litt c:\tc\litt
```

```
/*=======================================================================*/
/*                                                                       */
/*  Program module: Well Control Sheet     ( CTRLKICK.C )                */
/*                                                                       */
/*  Used to input information and calculation of pressure schedule       */
/*                                                                       */
/*  By:By Helio / Eric - DEP - UNICAMP                                   */
/*                                                                       */
/*  Date: 03/21/1990                                                     */
/*                                                                       */
/*=======================================================================*/

#include <math.h>
#include <stdio.h>
#include <kickgraf.c>
```

```
void HOLE_sections();
void CHOKE_LINE_data();
void DRILLSTRING_data();
void BIT_data();
void SURFACE_PLUMBING_data();
void PUMP_data();
void CIRCULATING_pressures();
void SAFETY_data();
void DIRECTIONAL_data();
void MUD_data();
void KICK_data();
void NEW_mud_wt();
void VERTICAL_depth();
void LOSSES_stroke();
void STROKES_sec();
void LOSS_sec();
float LOSS_bit();
float LOSS_surf();
float INTERPOL();
float TOTAL_loss();
void Cf_calc();
void Init_new_mud ();
void PUMP_start();
void DROP_string();
void RE_prepar();
void CHOKE_strtup_schedule();
void PLEASE_WAIT();
void POWER_LAW();
float PRESSURE ();
float FRICTION ();

char  yes_no;
int   ttl_strgs, surf_coef, kill_rate, stkstart, mud, ndata,
      stations,stk_srfc,strokes, b_mud, stk_mud[10], stk_c,
      n_mud, stks, stk_last, stkmx[10], stk_sc[10], hole_sections ;
float Cf, Cf_chk, hole_len[4], hole_diam[4], chk_diam, chk_len,
      chk_frction, mdepth[50], inclination[50], vdepth[50],
      nzl_area, zero = 0., strg_len[10], strg_indiam[10],
      strg_exdiam[10], rdc_pressure, csgburst, burstfct, brstnmbr,
      airgap, BOP_press, fracture, fracture_depth, pitgain,
      mud_wt[10], Drop[10001], vol_surf, vol_stk, loss[10][10],
      flow, SICP, SIDPP, dial[6],rpm[6],k[10],n[10],CSG[151];

main()

{
  HOLE sections();
   CHOKE_LINE_data(),
   DRILLSTRING_data();
   BIT_data();
   SURFACE_PLUMBING_data();
   PUMP_data();

   SAFETY_data();
   DIRECTIONAL_data();
   VERTICAL_depth();
   STROKES_sec();

   do  {
        CIRCULATING_pressures();
        mud   =  0 ;
        MUD_data();
        KICK_data();
        NEW_mud_wt();
        PLEASE_WAIT();
        POWER_LAW ( &ndata, &rpm, &dial, &k[mud], &n[mud] );
        CHOKE_strtup_schedule();
        LOSS_sec(flow);
        Cf_calc();

        do  {

             Init_new_mud();
             PLEASE_WAIT();
             DROP_string();


             Graph ((float)(strokes+stks),
                    (float)(strokes-stk_c-1),(float)stkstart);

             printf ( " \n\n\n Do you wish to pump a NEW MUD [Y/N] ? ===> ");
             yes_no = getche ();
             yes_no = toupper ( yes_no );

             if ( yes_no != 'N' )         RE_prepar();

        }   while ( yes_no != 'N' );

        printf ( "\n\n\n Do you wish performe a new Kick ? [Y/N] ===> ");
        yes_no = getche()  ;
        yes_no = toupper ( yes_no ) ;

  }  while ( yes_no != 'N' );

}

/*----------------------------------------------------------------------*/
void HOLE_sections()                                /* Input for well     */
                                                    /* configuration      */
{
 int i;
 clrscr ();
 printf ( "\n\n\n HOLE SECTIONS DATA \n\n\n");
 printf ( "\n Total number of hole sections......... ===> ");
 scanf  ( "%d" , &hole_sections );
 printf ( "\n\n Note: ");
 printf ( "\n              Input the hole sections ");
 printf ( "\n              starting from surface . \n");
 for    ( i=0 ; i < hole_sections ; i++ )  {
    printf ( "\n\n Length of section [%1d]........... (ft) ===> ",i+1);
    scanf  ( "%f" , &hole_len[i]);
    printf ( "\n Diameter of section [%1d]......... (in) ===> ",i+1);
    scanf  ( "%f" , &hole_diam[i]);
 }
  do    {
         printf ( "\n\n\n Do you wish to change any");
         printf ( "\n hole section information  [Y/N]  ?  " );
         yes_no = getche ();
         yes_no = toupper ( yes_no );
         if  ( yes_no == 'Y' )  {
           printf("\n\n\n Date number ........................... ===> ");
           scanf  ( "%d" , &i );
           i--;
           printf("\n\n Length of section [%d]........... (ft) ===> ",i+1);
           scanf ( "%f" , &hole_len[i]);
           printf("\n Diameter of  section[%d]......... (in) ===> ",i+1);
           scanf ( "%f" , &hole_diam[i]);
         }
   }  while ( yes_no == 'Y' );
```

```
}
/*---------------------------------------------------------------------------*/
void CHOKE_LINE_data()                              /* Input of Choke Line   */
                                                    /* data                  */
{
  clrscr ();
  printf ( "\n\n\n CHOKE LINE DATA \n\n\n");
  printf ( "\n Choke line diameter.............. (in) ===> ");
  scanf  ( "%f" , &chk_diam );
  printf ( "\n Length of choke line.............(ft) ===> ");
  scanf  ( "%f" , &chk_len );
}
/*---------------------------------------------------------------------------*/
void DRILLSTRING_data()                            /* Input of Drill String */
                                                    /* data                  */
{
  int     i, j ;

  clrscr ();
  printf ( "\n\n\n DRILLSTRING SECTIONS DATA \n\n\n");
  printf ( "\n\n Total number of drillstring sections.. ===> ");
  scanf  ( "%d" , &ttl_strgs );
  printf ( "\n\n Note: ");
  printf ( "\n        Input the drillstring sections");
  printf ( "\n        starting from the surface.\n");

  for    ( i=ttl_strgs - 1 ; i >=0 ; i-- )  {
    j = ttl_strgs - i;
    printf ( "\n\n Length of section [%d]........... (ft) ===> ", j );
    scanf  ( "%f" , &strg_len[i]);
    printf ( "\n I.D. of section [%d]............. (in) ===> ", j );
    scanf  ( "%f" , &strg_indiam[i]);
    printf ( "\n O.D. of section [%d]............. (in) ===> ", j );
    scanf  ( "%f" , &strg_exdiam[i]);
  }
   do   {
         printf ( "\n\n\n Do you wish to change any");
         printf ( "\n drillstring information  [Y/N]  ?  " );
         yes_no = getche ();
         yes_no = toupper ( yes_no );
         if  ( yes_no == 'Y' )  {
         printf ( "\n\n\n Section number ........................ ===> ");
           scanf  ( "%d" , &i );
           j = ttl_strgs - i ;
           printf("\n\n Length of section [%d]............ (ft) ===> ",i);
           scanf ( "%f" , &strg_len[j]);
           printf("\n I.D. of section [%d]............ (in) ===> ",i);
           scanf ( "%f" , &strg_indiam[j]);
           printf("\n O.D. of section [%d]............... (in) ===> ",i);
           scanf ( "%f" , &strg_exdiam[j]);
         }
   } while ( yes_no == 'Y' );
}
/*---------------------------------------------------------------------------*/
void BIT_data()                                     /* Input of Bit data     */

{
  int   ttl_nzls, nzl_diam[10], i;

  float PI = 3.141592654;

  clrscr ();
  printf ( "\n\n\n BIT DATA \n\n\n");
  printf ( "\n Total number of nozzles.............. ===> ");
  scanf  ( "%d" , &ttl_nzls );
  nzl_area = 0. ;
  for      ( i=0 ; i < ttl_nzls ; i++ )  {
    printf ( "\n\n Nozzle[%d] diameter.......... (1/32 in) ===> ",i+1);
    scanf  ( "%d" , &nzl_diam[i]);
    nzl_area = nzl_area + PI * nzl_diam[i]*nzl_diam[i] / 4096. ;
  }
}
/*---------------------------------------------------------------------------*/
void SURFACE_PLUMBING_data()                        /* Input of Surface      */
                                                    /*        Fitting data   */
{
  int    is;

  clrscr ();
  printf ( "\n\n\n SURFACE PLUMBING DATA \n\n\n");
  printf ( "\n Surface plumbing types table\n");
  printf ( "\n  ___________________________________________________");
  printf ( "\n |case|    Tube   |    Hose   |  Swivel   |  Kelly   |");
  printf ( "\n |    | len   ID  | len   ID  | len   ID  | len   ID |");
  printf ( "\n |    |(ft)  (in) |(ft)  (in) |(ft)  (in) |(ft)  (in)|");
  printf ( "\n |____|___________|___________|___________|__________|");
  printf ( "\n |  1 | 40    3.0 | 45    2.0 | 4     2.0 | 40   2.25|");
  printf ( "\n |  2 | 40    3.5 | 55    2.5 | 5     2.5 | 40   3.25|");
  printf ( "\n |  3 | 40    4.0 | 55    3.0 | 5     2.5 | 40   3.35|");
  printf ( "\n |  4 | 45    4.0 | 55    3.0 | 6     3.0 | 40   4.00|");
  printf ( "\n |  5 | other case: input coef. & volume             |");
  printf ( "\n |____|______________________________________________|");
  printf ( "\n\n                                case ===> ");
  scanf  ( "%d",&is);
  switch ( is ) {
     case 1:
       surf_coef = 19;
       vol_surf  = 30.94672764;
       break;
     case 2:
       surf_coef =  7;
       vol_surf  = 51.25488015;
       break;
     case 3:
       surf_coef =  4;
       vol_surf  = 65.89796594;
       break;
     case 4:
       surf_coef =  3;
       vol_surf  = 77.88701788;
       break;
     case 5:
       printf ( "\n Coef. of loss in the surface plumbing.  ===> ");
       scanf  ( "%d" , &surf_coef );
       printf ( "\n Surface plumbing volume..........(gal) ===> ");
       scanf  ( "%f" , &vol_surf );
       break;
  }
}
/*---------------------------------------------------------------------------*/
void PUMP_data()                                    /* Input of Pumps data   */

{
  char  c;
  int   i;
  float rod, lnr_sz, stk_lgth, effic ;

  clrscr ();
  printf ( "\n\n\n PUMP DATA \n\n\n");
  printf ( "\n Pump Type   :    [ D ] Duplex  (double acting).");
  printf ( "\n                  [ T ] Triplex (single acting).  ? ===> ");
  c = toupper ( getche () ) ;
  printf ( "\n\n\n Liner size.............. (in) ===> ");
  scanf  ( "%f" , &lnr_sz );
  printf ( "\n Stroke length........... (in) ===> ");
  scanf  ( "%f" , &stk_lgth );
  i = 1;
  rod = 0;
  vol_stk = 98.039445;
  if ( c == 'D' )  {
     printf ( "\n Rod diameter............ (in) ===> ");
     scanf  ( "%f", &rod );
     vol_stk = 147.059167;
     i = 2;
  }
  printf ( "\n Pump efficiency..(0 < ef < 1) ===> ");
  scanf  ( "%f" , &effic );
  vol_stk = ( i * lnr_sz*lnr_sz - rod*rod ) * stk_lgth / vol_stk * effic;

}
/*---------------------------------------------------------------------------*/
void CIRCULATING_pressures()

{
  clrscr ();
  printf ( "\n\n\n CIRCULATING PRESSURES \n\n\n");
  printf ( "\n\n Note:");
  printf ( "\n          At Kill rate ( through annulus ).\n");
  printf ( "\n\n Reduced pressure................. (psi) ===> ");
  scanf  ( "%f" , &rdc_pressure );
  printf ( "\n Kill rate................... (stk/min) ===> ");
  scanf  ( "%d" , &kill_rate );
  flow   = vol_stk * kill_rate ;
  printf ( "\n Strokes to stab. the Kill rate.. (stk) ===> ");
  scanf  ( "%d", &stkstart );
  printf ( "\n\n Note: ");
  printf ( "\n          For floating vessels.");
  printf ( "\n          Enter '0' for others.\n");
  printf ( "\n Choke line friction at Kill Rate (psi) ===> ");
  scanf  ( "%f", &chk_frction );

}
/*---------------------------------------------------------------------------*/
void SAFETY_data()

{
  clrscr ();
  printf ( "\n\n\n SAFETY DATA \n\n\n");
  printf ( "\n Casing Burst Rating............... (psi) ===> ");
  scanf  ( "%f", csgburst );
  printf ( "\n Burst safety factor................... ===> ");
  scanf  ( "%f", burstfct );
  printf ( "\n Casing top depth.................(ft) ===> ");
  scanf  ( "%d",brstnmbr);
  printf ( "\n\n Note: ");
  printf ( "\n          For floating vessels.");
  printf ( "\n          Enter '0' for others.\n");
  printf ( "\n Air Gap...........................(ft) ===> ");
  scanf  ( "%f", airgap );
  printf ( "\n Formation fracture pressure.....(psi) ===> ");
  scanf  ( "%f", fracture );
  printf ( "\n At measured depth................(ft) ===> ");
  scanf  ( "%f", fracture_depth );
  printf ( "\n BOP rating..................... (psi) ===> ");
  scanf  ( "%f" , &BOP_press );
}
/*---------------------------------------------------------------------------*/
void DIRECTIONAL_data()                            /* Input directional     */
                                                    /* data                  */
{
  int    j;

  clrscr ();
  printf ( "\n\n\n DIRECTIONAL DATA \n\n\n " );
  printf ( "\n\n Note: ");
  printf ( "\n        The first station is the surface.");
  printf ( "\n        The last one is the bottom hole.");
  printf ( "\n        Enter '0' for verticall wells.");
  printf ( "\n\n\n Number of stations..................... ===> ");
  scanf  ( "%d" , &stations );
  if ( stations == 0 )  {
     mdepth[0] = 0 ;
     inclination[0] = 0 ;
     inclination[1] = 0 ;
     printf ( "\n Total Vertical Depth .............(ft) ===> ");
     scanf  ( "%f", &mdepth[1] );
     stations = 2;
  }
  else {
     for ( j=0 ; j < stations ; j++ )   {
     printf ( "\n\n Measured Depth [%d]................(ft) ===> ", j+1 );
     scanf  ( "%f" , &mdepth[j] );
     printf ( "\n Inclination [%d]............(degrees) ===> ", j+1 );
     scanf  ( "%f" , &inclination[j] );
     }
  }
   do    {
          printf ( "\n\n Do you wish to change any");
          printf ( "\n directional data  [Y/N]  ?  " );
          yes_no = getche ();
          yes_no = toupper ( yes_no );
          if   ( yes_no == 'Y' )  {
            printf ( "\n\n\n Station number........................ ===> ");
            scanf  ( "%d" , &j );
            j--;
            printf("\n\n Measured Depth [%d]..................(ft) ===> ",j+1);
            scanf  ( "%f" , &mdepth[j] );
            printf ("\n Inclination [%d]..............(degrees) ===> ",j+1);
            scanf  ( "%f" , &inclination[j] );
          }
  } while ( yes_no == 'Y' );
}
/*---------------------------------------------------------------------------*/
void MUD_data()                                     /* Input of Pumps data   */
{
  int    j;

  clrscr ();
  printf ( "\n\n\n MUD PROPERTIES DATA \n\n\n" );
  printf ( "\n Mud weigth...................(lbs/gal) ===> ");
  scanf  ( "%f", &mud_wt[mud] );
  printf ( "\n Rheology Data" );
  printf ( "\n\n Note: Readings from  FANN VG METER 35 A");
  printf ( "\n\n\n Number of experimetal data........ ===> " );
  scanf  ( "%d" , &ndata );
  for ( j=0 ; j < ndata ; j++ )  {
     printf ( "\n\n Speed [%d].......................(RPM) ===> ", j+1 );
     scanf  ( "%f" , &rpm[j] );
     printf ( "\n Deflection [%d]..............(degrees) ===> ", j+1 );
     scanf  ( "%f" , &dial[j] );
  }
```

```
   do   {
        printf ( "\n\n\n Do you wish to change any");
        printf ( "\n mud property              [Y/N] ?  " );
        yes_no = getche ();
        yes_no = toupper ( yes_no );
        if  ( yes_no == 'Y' )  {
           printf ( "\n\n\n Data number............................ ===> ");
           scanf  ( "%d" , &j );
           j--;
           printf ( "\n\n Speed [%d]........................(RPM) ===> ", j+1);
           scanf  ( "%f" , &rpm[j] );
           printf ( "\n Deflection [%d]...............(degrees) ===> ", j+1 );
           scanf  ( "%f" , &dial[j] );
        }
   }  while ( yes_no == 'Y' );
}
/*-----------------------------------------------------------------------*/
void KICK_data()                                  /* Input of kick data    */
{
  clrscr ();
  printf ( "\n\n\n KICK INFORMATION \n\n\n");
  printf ( "\n Pit gain......................... (bbl) ===> ");
  scanf  ( "%f", &pitgain );
  printf ( "\n Shut-in drill pipe pressure SIDPP(psi) ===> ");
  scanf  ( "%f", &SIDPP );
  printf ( "\n Shut-in casing pressure SICP.....(psi) ===> ");
  scanf  ( "%f", &SICP );
}
/*-----------------------------------------------------------------------*/
void NEW_mud_wt()

{
  float nmwt;
  nmwt = mud_wt[0] + SIDPP / vdepth[stations-1] * 19.25;
  clrscr();
  printf ( "\n\n\n KILL   MUD   WEIGTH   SUGGESTED");
  printf ( "\n ( No safety margin considered ) ===> %f",nmwt);
  printf ( "\n\n\n\n\n\n\n\n\n\n Press any key to continue ===> ");
  while ( !kbhit()) ;
}
/*-----------------------------------------------------------------------*/
void VERTICAL_depth()                             /* vertical depth        */
                                                  /* calculation           */
{
  int   i;

  vdepth[0] = mdepth[0];
  for ( i = 1 ; i < stations ; i++ )
       vdepth[i] = vdepth[i-1] + ( mdepth[i] - mdepth[i-1] ) *
                   cos (( inclination[i] + inclination[i-1]) / 114.591559);
}

/*-----------------------------------------------------------------------*/
float INTERPOL(md)                                /* linear interpolation  */
                                                  /* calculation           */
float md;

{
  int   i;

  if ( md <= mdepth[0] )       return vdepth[0];
  for ( i = 1 ; i < stations ; i++ )  {
       if ( md <= mdepth[i] )    {
          md = (md - mdepth[i-1])/(mdepth[i]-mdepth[i-1])*
                (vdepth[i]-vdepth[i-1]) + vdepth[i-1];
          return md;
       }
  }
  return vdepth[stations-1];
}
/*-----------------------------------------------------------------------*/
void LOSSES_stroke()                              /* losses calculation for*/
                                                  /* each stroke           */
{
  int   i = 0 , j, stkmud, stksc;
  float base, length, vlength;
  base          = mdepth [ stations - 1 ];
  Drop[strokes] = 0 ;
  if ( SICP < chk_frction )       Drop[strokes] = chk_frction - SICP;
  Drop[strokes] = Drop[strokes] + SIDPP + LOSS_bit(b_mud,flow)  * Cf ;
  j = b_mud;
  if ( stk_mud[j] == 0 )   j++;
  stkmud = stk_mud[j];
  stksc = stk_sc[i];
  do  {
       if ( stkmud > stksc )  {
            length  = strg_len[i]*(float)stksc/(float)stk_sc[i];
            vlength = INTERPOL(base) - INTERPOL(base-length);
            Drop[strokes]=Drop[strokes] +
                          loss[j][i] * length * Cf -
                          (mud_wt[j]-mud_wt[0]) * vlength / 19.25;
            base   = base   - length;
            stkmud = stkmud - stksc;
            i++;
            stksc = stk_sc[i];
       }
       else  {
            length  = strg_len[i]*(float)stkmud/(float)stk_sc[i];
            vlength = INTERPOL(base) - INTERPOL(base-length);
            Drop[strokes]=Drop[strokes] +
                          loss[j][i] * length * Cf -
                          (mud_wt[j]-mud_wt[0]) * vlength / 19.25;
            base  = base  - length;
            stksc = stksc - stkmud;
            if ( !stksc )  {
                 i++;
                 stksc = stk_sc[i];
            }
            j++;
            stkmud = stk_mud[j];
       }
  }  while ( i < ttl_strgs );

      if ( stk_mud[b_mud+n_mud-1]<stk_srfc)  {
           Drop[strokes] = Drop[strokes]+((float)stk_mud[mud]/(float)stk_srfc*
                           LOSS_surf(mud,flow)+
                           ((float)stk_srfc-(float)stk_mud[mud]) /
                           (float)stk_srfc* LOSS_surf(mud-1,flow)) * Cf ;
       }
       else
           Drop[strokes] = Drop[strokes] + LOSS_surf(mud,flow) *  Cf ;


}
/*-----------------------------------------------------------------------*/
void STROKES_sec()                                /* strokes  /  section   */
                                                  /* calculation           */
{
  int   i;

  float PI = 3.141592654;

  stk_srfc = vol_surf/vol_stk;
  stk_c       = stk_srfc;
  for ( i=0 ; i<ttl_strgs ; i++ ) {
       stk_sc[i]= strg_len[i] * strg_indiam[i] * strg_indiam[i] * PI/
                  77. / vol_stk;
       stk_c     = stk_c + stk_sc[i];
  }
}

/*-----------------------------------------------------------------------*/
float LOSS_bit(mud,flow)                          /* losses at bit         */
                                                  /* calculation           */
int    mud;
float flow;

{
  return (flow * flow * mud_wt[mud] / 10858 / nzl_area);
}

/*-----------------------------------------------------------------------*/
float LOSS_surf(mud,flow)                         /* losses at surface fit.*/
                                                  /* calculation           */
int   mud;
float flow;

{
 return (surf_coef * pow ( flow,1.85543 ) * 1.028525E-5 *
         mud_wt[mud]);
}
/*-----------------------------------------------------------------------*/
float TOTAL_loss()                                /* total losses          */
                                                  /* calculation           */
{
  int     i, j ;

  float  total_losses, prof, auxstr, auxwell, Drop;

  total_losses = LOSS_bit(mud,flow);
  total_losses = total_losses + LOSS_surf(mud,flow);
  for ( i=0 ; i < ttl_strgs ; i++ )
       total_losses = total_losses + loss[mud][i] * strg_len[i];
  j = hole_sections - 1 ;
  prof = mdepth [ stations - 1 ];
  auxstr = strg_len [ 0 ];
  auxwell = hole_len [ hole_sections - 1 ];
  for ( i=0 ; i < ttl_strgs ; i++ ) {
       if ( (prof - auxstr) > (prof - auxwell) ) {
             prof = prof - auxstr;
             auxwell = auxwell - auxstr;
             Drop     = PRESSURE (  &flow, &mud_wt[0], &k[0], &n[0],
                                    &hole_diam[j], &strg_exdiam[i] );
             total_losses = total_losses + auxstr * Drop;
             auxstr = strg_len [i-1];

        }
        else  {

             prof = prof - auxwell;
             auxstr = auxstr - auxwell;
             Drop     = PRESSURE (  &flow, &mud_wt[0], &k[0], &n[0],
                                    &hole_diam[j], &strg_exdiam[i] );
             total_losses = total_losses + auxwell * Drop;
             j--;
             auxwell = hole_len [j];

        }
  }
  return total_losses;
}
/*-----------------------------------------------------------------------*/
void CHOKE_strtup_schedule()

{
  int i;
  float qstart;

  if ( chk_len == 0 || chk_diam == 0 || stkstart == 0 ) return ;
  if ( chk_frction == 0 )                    Cf_chk = 1 ;
  else     {

     Cf_chk = chk_frction / chk_len /
              PRESSURE ( &flow, &mud_wt[0], &k[0], &n[0], &chk_diam, &zero );

  }
  CSG [ 0 ]         = SICP;
  for ( i = 1 ; i <= stkstart ; i++ )  {
     qstart  = flow * (float)i / (float)stkstart;
     CSG [i] =SICP-chk_len*Cf_chk*PRESSURE ( &qstart, &mud_wt[0], &k[0],
                                             &n[0], &chk_diam, &zero );
     if ( CSG[i] < 0 )  CSG[i] = 0 ;
  }
  chk_frction=PRESSURE ( &flow, &mud_wt[0], &k[0], &n[0], &chk_diam, &zero )
              * Cf_chk * chk_len ;
}


/*-----------------------------------------------------------------------*/
void LOSS_sec(flow)                               /* losses per section    */
                                                  /* calculation           */
float flow;

{
  int   i;

  for ( i=0 ; i < ttl_strgs ; i++ )

       loss[mud][i] = PRESSURE ( &flow, &mud_wt[mud], &k[mud], &n[mud],
                                 &strg_indiam[i], &zero );


}
/*-----------------------------------------------------------------------*/
void Cf_calc()

{
      if ( !rdc_pressure )  {
         Cf = 1;
      }
      else  {
         Cf = rdc_pressure / TOTAL_loss();
      }
      strokes            = 0      ;
      Drop[strokes]      = SIDPP  ;
      strokes++                   ;
      b_mud              = 0      ;
      stk_mud [b_mud]    = stk_c-1;
      n_mud              = 2      ;
      stkmx [b_mud]      = stk_c  ;
      stk_mud [ 1 ]      = 1      ;
      stkmx   [ 1 ]      = stk_c  ;
```

```
}
/*-----------------------------------------------------------------------*/
void Init_new_mud ()

{

  mud++;
  MUD_data();
  POWER_LAW ( &ndata, &rpm, &dial, &k[mud], &n[mud] );
  LOSS_sec(flow);
  stks = 1 ;

}
/*-----------------------------------------------------------------------*/
void PUMP_start()
{
  int i=0, j, stkmud, stksc;
  float loss[2][10], qstart, chk_frction, base, length, vlength;
  base          = mdepth [ stations - 1 ];
  qstart = flow * (float) strokes / (float)stkstart;
  chk_frction = PRESSURE ( &qstart, &mud_wt[0], &k[0], &n[0], &chk_diam,
                           &zero ) * Cf_chk * chk_len ;
  for ( j = 0 ; j < ttl_strgs ; j++ )     {
       loss[0][j] = PRESSURE (  &qstart, &mud_wt[0], &k[0], &n[0],
                                &strg_indiam[j], &zero );
       loss[1][j] = PRESSURE (  &qstart, &mud_wt[1], &k[1], &n[1],
                                &strg_indiam[j], &zero );
  }
  Drop[strokes] = 0 ;
  if ( SICP < chk_frction )       Drop[strokes] = chk_frction - SICP;
  Drop[strokes] = SIDPP + LOSS_bit(b_mud,qstart) * Cf ;
  j = b_mud;
  if ( stk_mud[j] == 0 )   j++;
  stkmud = stk_mud[j];
  stksc = stk_sc[i];
  do  {
       if ( stkmud > stksc )  {
            length  = strg_len[i]*(float)stksc/(float)stk_sc[i];
            vlength = INTERPOL(base) - INTERPOL(base-length);
            Drop[strokes]=Drop[strokes] +
                          loss[j][i] * length * Cf -
                          (mud_wt[j]-mud_wt[0]) * vlength / 19.25;
            base   = base   - length;
            stkmud = stkmud - stksc;
            i++;
            stksc = stk_sc[i];
       }
       else  {
            length  = strg_len[i]*(float)stkmud/(float)stk_sc[i];
            vlength = INTERPOL(base) - INTERPOL(base-length);
            Drop[strokes]=Drop[strokes] +
                          loss[j][i] * length * Cf -
                          (mud_wt[j]-mud_wt[0]) * vlength / 19.25;
            base  = base  - length;
            stksc = stksc - stkmud;
            if ( !stksc )  {
                 i++;
                 stksc = stk_sc[i];
            }
            j++;
            stkmud = stk_mud[j];
       }
   }  while ( i < ttl_strgs );

       if ( stk_mud[b_mud+n_mud-1]<stk_srfc)  {
          Drop[strokes] = Drop[strokes]+((float)stk_mud[mud]/
                          (float)stk_srfc* LOSS_surf(mud,qstart)+
                          ((float)stk_srfc-(float)stk_mud[mud]) /
                          (float)stk_srfc*LOSS_surf(mud-1,qstart))* Cf ;
       }
       else
          Drop[strokes] = Drop[strokes] + LOSS_surf(mud,qstart) *  Cf ;
}
/*-----------------------------------------------------------------------*/
void DROP_string()

{
  int i;

  do  {

       if ( strokes < stkstart )
          PUMP_start();
       else
          LOSSES_stroke();
       strokes++;
       stks++;
       stk_mud [ b_mud ]--;
       if (stk_mud [ b_mud ] < 0 )  {
          b_mud++;
          stk_mud [b_mud]--;
          n_mud--;
       }
       if (stk_mud[b_mud+n_mud-1]==stkmx[b_mud+n_mud-1])
          n_mud++;
          stk_mud [b_mud+n_mud-1]++;
  } while ( stks <= stk_c+1 );
  strokes--;
  stks = .2 * stk_c;
  for ( i=1 ; i <= stks ; i++ )
        Drop[strokes + i] = Drop[strokes];

}
/*-----------------------------------------------------------------------*/
void RE_prepar()

{
  int i, alfa;

  printf ( "\n\n\n Beginning at stroke number ? ===> ");
  scanf  ( "%d", &alfa );
  alfa++;
  stkmx[b_mud+n_mud-1] = alfa - strokes + stk_c - 1 ;
  stk_mud [b_mud + n_mud - 1 ] = stkmx[b_mud+n_mud-1];
  strokes = alfa ;
  alfa     = stk_c - stkmx[b_mud+n_mud-1] - 1 ;
  n_mud    = n_mud + 1 ;
  stkmx [ b_mud + n_mud - 1 ] = stk_c ;
  stk_mud [ b_mud + n_mud - 1 ] = 1 ;
  for ( i = 3 ;; i++ )  {

       if ( stkmx [ b_mud + n_mud - i ] >= alfa )  {
            b_mud = b_mud - 1 ;
            n_mud = n_mud + 1 ;
            stk_mud [ b_mud ] = alfa ;
            break;
       }
       else  {
          b_mud = b_mud - 1 ;
          n_mud = n_mud + 1 ;
          stk_mud [ b_mud + n_mud - i ] = stkmx [ b_mud + n_mud - i ];
          alfa  = alfa - stkmx [ b_mud + n_mud - i ] ;
       }
    }
}
/*-----------------------------------------------------------------------*/
void PLEASE_WAIT()

{
  clrscr();
  printf ( "\n\n\n\n\n\n\n\n\n\n\n                              PLEASE  WAIT  ");
}
/*-----------------------------------------------------------------------*/
void POWER_LAW( ndata, rpm, dial, k, n )

int   *ndata;

float *rpm, *dial, *k, *n;

{
  int   i;
  float sx = 0, sy = 0, sxy = 0, sxx = 0;
  for ( i = 0 ; i < *ndata ; i++ )  {
       rpm [i] = log ( 1.703  * rpm[i]  );
       dial[i] = log ( 5.1405 * dial[i] );
       sx       = sx  + rpm [i];
       sy       = sy  + dial[i];
       sxy      = sxy + rpm [i] * dial[i];
       sxx      = sxx + rpm [i] * rpm [i];
  }
  *n = ( sxy - sy * sx / *ndata ) / ( sxx - sx * sx / *ndata );
  *k = exp ( ( sy - *n * sx ) / *ndata ) * 100 ;
}
/*-----------------------------------------------------------------------*/
float PRESSURE ( flow, mud_wt, k, n, OD, ID )

float *flow, *mud_wt, *k, *n, *OD, *ID;
{
  float Reynolds, vel, NRec = 2000 ;
  Reynolds = *mud_wt / *k;
  vel      = *flow / 2.448;
  if ( *ID )  {
     vel       = vel / ( *OD * *OD - *ID * *ID );
     Reynolds = 109000 * Reynolds * pow ( vel, 2 - *n );
     Reynolds =Reynolds * pow ( .0208 * ( *OD - *ID)/(2+1/ *n), *n);
  }
  else  {
     vel       = vel / *OD / *OD ;
     Reynolds = 89100 * Reynolds * pow ( vel, 2 - *n );
     Reynolds = Reynolds * pow ( .0416 * *OD / ( 3 + 1 / *n ), *n );
  }
  if ( *n >= .2 && *n < .45 )   NRec = 5960 - 8800 * *n;
  if ( *n < .2 )                NRec = 4200          ;
  if ( Reynolds < NRec )  {
     vel = *k * pow ( vel, *n ) / 144000;
     if ( *ID )
        return vel * pow( (2+1/ *n)/.0208, *n)/pow( *OD- *ID, 1+ *n);
     else
        return vel * pow( (3+1/ *n)/.0416, *n)/pow( *OD , 1 + *n);
  }
  else {
     vel = FRICTION ( &Reynolds, &*n ) * *mud_wt * vel * vel;
     if ( *ID )
        return vel / 21.1 / ( *OD -  *ID );
     else
        return vel / 25.8 / *OD ;
  }
}

/*-----------------------------------------------------------------------*/
float FRICTION ( Reynolds, n )

float *Reynolds, *n;

{
  int   i = 1 ;
  float f = .001, swap, final ;

  do {
       swap  = 4 / pow ( *n, .75 ) ;
       swap  = swap * log10 ( *Reynolds * pow ( f, 1 - *n/2 ) );
       swap  = swap - .395 / pow ( *n, 1.2 );
       final = 1 /  swap / swap ;
       if ( fabs( final - f ) <= .00001 )
          i = 0;
       else
          f = final;
  } while (i);
  return final;
}

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

/*=======================================================================*/
/*                                                                       */
/*  Program module: Well Control Sheet                                   */
/*                                                                       */
/*  Used to output graphic information from the main program.            */
/*                                                                       */
/*  By: Fabio de Andrade Netto - DEP - UNICAMP                           */
/*                                                                       */
/*  Date: 03/21/1990                                                     */
/*                                                                       */
/*=======================================================================*/

#include <conio.h>
#include <graphics.h>
#include <alloc.h>
#include <math.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define ESC       0x1b                     /* Define the escape key        */
#define F1        0x3b                     /* Define the function keys     */
#define F2        0x3c
#define F3        0x3d
#define F4        0x3e
#define UP        0x48                     /* Define navigation keys       */
#define DOWN      0x50
#define PGUP      0x49
#define PGDN      0x51
#define TRUE      1                        /* Define some handy constants  */
#define FALSE     0                        /* Define some handy constants  */
#define PRECISION 5                        /* Define some handy constants  */

int    GraphDriver;              /* Graphics device driver               */
int    GraphMode;                /* Graphics mode value                  */
int    MaxColors;                /* The maximum # of colors available    */
int    ErrorCode;                /* Reports any graphics errors          */
int    MaxSizeX, MaxSizeY;       /* # of pixels to draw the picture      */
div_t  Res;                      /* Struct for int divisions             */
char   key;                      /* Holds a key code                     */
float  MinX, MaxX, MinY, MaxY;   /* The limits of the sheet              */
```

```
float  Step;                          /* Table division                    */
void   *buffer;                       /* Image pointer                     */
int    First_Time;                    /* Define some handy variables       */
int    Control, Escape;               /* Define some handy variables       */
float MinXHist, MaxXCSG;
extern float Drop[];
extern float CSG[];
extern int stk_c;

/*----------------------------------------------------------------------*/
/*      Function prototypes                                             */
/*----------------------------------------------------------------------*/


void Graph(float MaxStrokes, float MinDefault, float MaxCasing);
void Initialize(void);
void MainWindow(char *header);
void MenuLine(char *msg);
void DrawBorder(void);
void DrawAxis(float XLeft, float XRight, float YBottom, float YTop,
              int Arrows, int Lines, char *Xaxisname, char *Yaxisname,
              char *Header);
void Function(float XLeft, float XRight, float YBottom, float YTop, int CPS);
void Base_Screen(void);
void Hist(float MinXVect);
void Zoom(void);
void Table(float MinXVect, float MaxXVect, char *YSide, int CPS);
void Flux_Control(void);
void CPS(void);
float Search_Max_Drop(float MinXVect, float MaxXVect);
float Search_Min_Drop(float MinXVect, float MaxXVect);
float Search_Max_CSG(void);
float Search_Min_CSG(void);


/*----------------------------------------------------------------------*/
/*      GRAPH: Function called by the main program.                     */
/*----------------------------------------------------------------------*/

void Graph(float MaxStrokes, float MinDefault, float MaxCasing)

{
  Initialize();                       /* Set system into Graphics mode      */
  First_Time = TRUE;
  Step = 10;
  MinX = 0;
  MinY = 0;
  MaxX = MaxStrokes;
  MaxY = (Search_Max_Drop(MinX, MaxX));
  MinXHist = MinDefault;
  MaxXCSG = MaxCasing;
  Flux_Control();
  closegraph();
}

/*----------------------------------------------------------------------*/
/*      INITIALIZE: Initializes the graphics system and reports         */
/*      any errors which occured.                                       */
/*----------------------------------------------------------------------*/

void Initialize(void)

{
  if (registerbgidriver(CGA_driver) < 0) exit(1);
  if (registerbgidriver(EGAVGA_driver) < 0) exit(1);
  if (registerbgidriver(Herc_driver) < 0) exit(1);
  if (registerbgifont(small_font) < 0) exit(1);
  GraphDriver = DETECT;                 /* Request auto-detection         */
  initgraph(&GraphDriver, &GraphMode, "");
  ErrorCode = graphresult();            /* Read result of initialization*/
  if(ErrorCode != grOk){                /* Error occured during init      */
    printf(" Graphics System Error: %s\n", grapherrormsg( ErrorCode ) );
    exit(1);
  }
  MaxColors = getmaxcolor() + 1;        /* Read maximum number of colors*/
}

/*----------------------------------------------------------------------*/
/*      MAINWINDOW: Establish the main window for the output and set    */
/*      a viewport.                                                     */
/*----------------------------------------------------------------------*/

void MainWindow(char *header)

{
  int height;
  setcolor(WHITE);                   /* Set current color to white   */
  settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
  settextjustify( CENTER_TEXT, TOP_TEXT);
  height = textheight("H" );            /* Get basic text height          */
  setviewport(0, 0, getmaxx(), height+4, 1);
  clearviewport();                      /* Clear graphics screen          */
  DrawBorder();
  outtextxy(getmaxx()/2, 3, header);
  setviewport(0, height+4, getmaxx(), getmaxy()-(height+4), 1);
  DrawBorder();
  setviewport(1, height+5, getmaxx()-1, getmaxy()-(height+5), 1);
  clearviewport();
}

/*----------------------------------------------------------------------*/
/*      MENULINE: Display a menu line at the bottom of the screen.      */
/*----------------------------------------------------------------------*/

void MenuLine(char *msg)

{
  int height;

  setcolor(WHITE);                   /* Set current color to white   */
  settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
  settextjustify(CENTER_TEXT, TOP_TEXT);
  setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
  setfillstyle(EMPTY_FILL, 0);
  height = textheight("H");              /* Detemine current height        */
  setviewport(0, getmaxy()-(height+4), getmaxx(), getmaxy(), 1);
  DrawBorder();
  outtextxy(getmaxx()/2, 3, msg);
  setviewport(1, height+5, getmaxx()-1, getmaxy()-(height+4), 1);
}

/*----------------------------------------------------------------------*/
/*      DRAWBORDER: Draw a solid single line around the current         */
/*      viewport.                                                       */
/*----------------------------------------------------------------------*/

void DrawBorder(void)

{
  struct viewporttype view;

  setcolor(WHITE);                   /* Set current color to white   */
  setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
  getviewsettings(&view);
  rectangle(0, 0, view.right-view.left, view.bottom-view.top);
}

/*----------------------------------------------------------------------*/
/*      DRAWAXIS: Used to draw XY graphics where parameters  are        */
/*      sent to the program by the user.                                */
/*----------------------------------------------------------------------*/

void DrawAxis(float XLeft, float XRight, float YBottom, float YTop,
              int Arrows, int Lines, char *Xaxisname, char *Yaxisname,
              char *Header)

{
  int hordisp, verdisp;
  int width, numtickshor, numticksver, i, j;
  int tickspace = 40;
  char numprint[10];
  float incrhor, incrver, horrequ, verrequ;
  struct viewporttype view;

  getviewsettings(&view);
  hordisp = view.right-view.left;    /* Size in pixels to draw the axis */
  verdisp = view.bottom-view.top;
  horrequ = XRight - XLeft;  /* Real size of the axis */
  verrequ = YTop - YBottom;
  line(10, verdisp-10, hordisp-10, verdisp-10);
  line(10, 10, 10, verdisp-10);
  if (Arrows) {
    line(hordisp-15, verdisp-10, hordisp-15, verdisp-7);
    line(hordisp-15, verdisp-7, hordisp-10, verdisp-10);
    line(10, 10, 7, 15);
    line(7, 15, 10, 15);
  }
  setusercharsize(1,1,1,1);
  settextstyle(SMALL_FONT, HORIZ_DIR, USER_CHAR_SIZE);
  settextjustify(LEFT_TEXT, BOTTOM_TEXT);
  width = textwidth(Xaxisname);
  setcolor(LIGHTRED);
  outtextxy(hordisp-width-5, verdisp-12, Xaxisname);
  outtextxy(13, 10, Yaxisname);
  setcolor(YELLOW);
  settextjustify(CENTER_TEXT, BOTTOM_TEXT);
  setusercharsize(3,2,3,2);
  settextstyle(SMALL_FONT, HORIZ_DIR, USER_CHAR_SIZE);
  outtextxy(hordisp/2, 15, Header);
  setcolor(WHITE);
  Res = div((hordisp-40), tickspace);  numtickshor = Res.quot;
  Res = div((verdisp-40), tickspace);  numticksver = Res.quot;
  MaxSizeX = numtickshor*tickspace;
  MaxSizeY = numticksver*tickspace;
  if (Lines) {
    setcolor(RED);
    setlinestyle(DOTTED_LINE,1,0);
  }
  for(i=1; i<=numtickshor; i++) {
    line(i*tickspace+10, verdisp-11, i*tickspace+10, verdisp-9);
    if (Lines)
      line(i*tickspace+10, verdisp-11, i*tickspace+10, verdisp-MaxSizeY-10);
  }
  for(i=1; i<=numticksver; i++) {
    line(9, verdisp-i*tickspace-10, 11, verdisp-i*tickspace-10);
    if (Lines) {
      line(11,verdisp-i*tickspace-10, MaxSizeX+10, verdisp-i*tickspace-10);
    }
  }
  setcolor(LIGHTGREEN);
  setlinestyle(SOLID_LINE,1,1);
  incrhor = (double) horrequ/numtickshor;
  incrver = (double) verrequ/numticksver;
  settextstyle(SMALL_FONT, HORIZ_DIR, USER_CHAR_SIZE);
  settextjustify(LEFT_TEXT, BOTTOM_TEXT);
  setusercharsize(18,19,6,7);
  width = textwidth("H");
  for(i=0; i<=numtickshor; i++) {
    gcvt(XLeft+incrhor*i, PRECISION, numprint);
    outtextxy(i*tickspace+10, verdisp-2, numprint);
  }
  settextstyle(SMALL_FONT, VERT_DIR, USER_CHAR_SIZE);
  settextjustify(CENTER_TEXT, BOTTOM_TEXT);
  width = textwidth("H");
  for(i=0; i<=numticksver; i++) {
    gcvt(YBottom+incrver*i, PRECISION, numprint);
    outtextxy(3, verdisp-i*tickspace-10, numprint);
    }
}

/*----------------------------------------------------------------------*/
/*      FUNCTION: Used to draw XY functions where parameters  are       */
/*      sent to the program by the user.                                */
/*----------------------------------------------------------------------*/

void Function(float XLeft, float XRight, float YBottom, float YTop,
              int CPS)

{
  int verdisp, hordisp, i;
  float incrhor, incrver, horrequ, verrequ;
  float x, y;

  struct viewporttype view;

  getviewsettings(&view);
  verdisp = view.bottom-view.top;
  horrequ = XRight - XLeft;  /* Real size of the function */
  verrequ = YTop - YBottom;
  setcolor(GREEN);
  setviewport(view.left, view.top, view.right-10, view.bottom-10, 1);
  setlinestyle(SOLID_LINE,1,1);
  incrhor = (double) MaxSizeX/horrequ;
  incrver = (double) MaxSizeY/verrequ;
  for(i=floor(XLeft); i<=floor(XRight); i++) {
    x=(double) incrhor*(i-XLeft)+10.;
    if (CPS) y=CSG[i];
      else y=Drop[i];
    y=verdisp-(y-YBottom)*incrver-10;
    if(i==floor(XLeft)) {
      putpixel(floor(x), floor(y), 2);
      moveto(floor(x), floor(y));
    }
    else lineto(floor(x), floor(y));
  }
}

/*----------------------------------------------------------------------*/
/*      FLUX_CONTROL: Control the user choices.                         */
/*----------------------------------------------------------------------*/
```

```
void Flux_Control(void)

{
  Escape = FALSE;
  if (First_Time) Base_Screen();
  MenuLine("F1-Main  F2-Last Mud  F3-Zoom  F4-Casing  ESC-Exit");
  while(!Escape) {
    switch(key) {
      case F1  : Base_Screen(); break;
      case F2  : Hist(MinXHist); break;
      case F3  : Zoom(); break;
      case F4  : CPS(); break;
      case ESC : Escape = TRUE; break;
      default  : key = 0; break;
    }
  }
}

/*----------------------------------------------------------------*/
/*      TABLE: Constructs an information table containing data    */
/*      about the graphic.                                        */
/*----------------------------------------------------------------*/

void Table(float MinXVect, float MaxXVect, char *YSide, int CPS)

{
  int i, sup_lim;
  float XIni;
  char numprint[10];

  setcolor(LIGHTCYAN);
  setviewport(getmaxx()-110, 15, getmaxx()-1, 145, 0);
  settextstyle(SMALL_FONT, HORIZ_DIR, 4);
  settextjustify(LEFT_TEXT, BOTTOM_TEXT);
  clearviewport();
  rectangle(0, 0, 106, 110);
  line(53, 0, 53, 110);
  for(i=0; i<=10; i++) line(0, i*10, 106, i*10);
  outtextxy(12, 8, "#Strk");
  outtextxy(70, 8, YSide);
  outtextxy(40, 120, "KEYS: ");
  outtextxy(0, 130, "Scroll: UP & DOWN");
  outtextxy(0, 140, "Step: PgUP & PgDN");
  if (First_Time) {
    buffer = malloc(imagesize(0, 0, 106, 110));
    getimage(0, 0, 106, 110, buffer);
    First_Time = FALSE;
  }
  XIni = MinXVect;
  Control = FALSE;
  gcvt(XIni, PRECISION, numprint);
  outtextxy(4, 18, numprint);
  if (CPS) gcvt(CSG[XIni], PRECISION, numprint);
    else gcvt(Drop[XIni], PRECISION, numprint);
  outtextxy(57, 18, numprint);
  do {
    key = getch();
    switch(key) {
      case UP   : XIni += Step; break;
      case DOWN : XIni -= Step; break;
      case PGUP : Step *= 10.; break;
      case PGDN : Step /= 10.; break;
    }
    if((key==F1)||(key==F2)||(key==F3)||(key==F4)||(key==ESC)) Control=TRUE;
      else if((key==UP)||(key==DOWN)||(key==PGUP)||(key==PGDN)) {
        if (Step<1) Step=1;
        if (XIni<MinXVect) XIni = MinXVect;
        sup_lim = 1;
        while (((XIni+sup_lim*Step)<=MaxXVect)&&(sup_lim<10)) sup_lim++;
        putimage(0, 0, buffer, COPY_PUT);
        for(i=0; i<sup_lim; i++) {
          if (XIni<=MaxXVect){
            gcvt(XIni+Step*i, PRECISION, numprint);
            outtextxy(4, 18+10*i, numprint);
            if (CPS) gcvt(CSG[XIni+floor(Step*i)], PRECISION, numprint);
              else gcvt(Drop[XIni+floor(Step*i)], PRECISION, numprint);
            outtextxy(57, 18+10*i, numprint);
          }
          else {
            gcvt(XIni, PRECISION, numprint);
            outtextxy(4, 18, numprint);
            outtextxy(57, 18, "?????");
          }
        }
      }
  } while(!Control);
}

/*----------------------------------------------------------------*/
/*      BASE_SCREEN: Perform the graphic output of the program.   */
/*----------------------------------------------------------------*/

void Base_Screen(void)

{
  int i;
  float MaxYBase;
  char numprint[10];


  struct viewporttype view;

  MainWindow("Well Control Sheet - DEP UNICAMP - MAIN SCREEN");
  getviewsettings(&view);
  setviewport(view.left, view.top, getmaxx()-100, view.bottom, 0);
  MaxYBase = 1.1*Search_Max_Drop(MinX, MaxX);
  DrawAxis(MinX, MaxX, MinY, MaxYBase, 1, 1, "#Strk", "DPP (psi)",
           "Drill Pipe Pressure Schedule");
  Function(MinX, MaxX, MinY, MaxYBase, 0);
  setcolor(getmaxcolor());
  setusercharsize(1,1,1,1);
  settextstyle(SMALL_FONT, HORIZ_DIR, USER_CHAR_SIZE);
  gcvt(stk_c, PRECISION, numprint);
  outtextxy(MaxSizeX/2-5, 28, "Strokes to bit:");
  outtextxy(MaxSizeX/2+55, 28, numprint);
  if (First_Time)
     MenuLine("F1-Main  F2-Last Mud  F3-Zoom  F4-Casing  ESC-Exit");
  Table(MinX, MaxX, "DPP", 0);
}


/*----------------------------------------------------------------*/
/*      MUD HISTORY: Shows the last mud information.              */
/*----------------------------------------------------------------*/

void Hist(float MinXVect)

{
  int i;
  float MaxYHist, MinYHist, Delta;

  struct viewporttype view;

  MainWindow("Well Control Sheet - DEP UNICAMP - LAST MUD");
  getviewsettings(&view);
  setviewport(view.left, view.top, getmaxx()-100, view.bottom, 0);
  MaxYHist = Search_Max_Drop(MinXVect, MaxX);
  MinYHist = Search_Min_Drop(MinXVect, MaxX);
  Delta = (MaxYHist-MinYHist)/10;
  MaxYHist += Delta;
  MinYHist -= Delta;
  DrawAxis(MinXVect, MaxX, MinYHist, /*Drop[floor(MinYHist)]*/ MaxYHist,
           1, 1, "#Strk", "DPP (psi)", "Drill Pipe Pressure Schedule");
  Function(MinXVect, MaxX, MinYHist, MaxYHist, 0);
  Table(MinXVect, MaxX, "DPP", 0);
}

/*----------------------------------------------------------------*/
/*      ZOOM: Gets a portion of the screen and enlarge it.        */
/*----------------------------------------------------------------*/

void Zoom (void)

{
  int i;
  float MinXZoom, MaxXZoom, MinYZoom, MaxYZoom, Delta, Temp;

  struct viewporttype view;

  restorecrtmode();
  clrscr();
  printf("Please enter the folowing data:\n\nInitial Stroke #: ");
  scanf("%f", &MinXZoom);
  printf("\n\nFinal Stroke #: ");
  scanf("%f", &MaxXZoom);
  if (MinXZoom > MaxXZoom) {
    Temp = MinXZoom;
    MinXZoom = MaxXZoom;
    MaxXZoom = Temp;
  }
  if(MinXZoom<MinX) MinXZoom = MinX;
  if(MaxXZoom>MaxX) MaxXZoom = MaxX;
  MinYZoom = Search_Min_Drop(MinXZoom, MaxXZoom);
  MaxYZoom = Search_Max_Drop(MinXZoom, MaxXZoom);
  Delta = (MaxYZoom-MinYZoom)/10;
  MaxYZoom += Delta;
  MinYZoom -= Delta;
  setgraphmode(GraphMode);
  MainWindow("Well Control Sheet - DEP UNICAMP - ZOOM");
  getviewsettings(&view);
  setviewport(view.left, view.top, getmaxx()-100, view.bottom, 0);
  DrawAxis(MinXZoom, MaxXZoom, MinYZoom, MaxYZoom, 1, 1, "#Strk",
           "DPP (psi)", "Drill Pipe Pressure Schedule");
  Function(MinXZoom, MaxXZoom, MinYZoom, MaxYZoom, 0);
  MenuLine("F1-Main  F2-Last Mud  F3-Zoom  F4-Casing  ESC-Exit");
  Table(MinXZoom, MaxXZoom, "DPP", 0);
}

/*----------------------------------------------------------------*/
/*      CPS: Shows the casing pressure schedule.                  */
/*----------------------------------------------------------------*/

void CPS(void)

{
  int i;
  float MinYCPS, MaxYCPS, Delta;

  struct viewporttype view;

  MainWindow("Well Control Sheet - DEP UNICAMP - START UP");
  getviewsettings(&view);        setviewport(view.left, view.top,
                                        getmaxx()-100, view.bottom, 0);
  MinYCPS = Search_Min_CSG();
  MaxYCPS = Search_Max_CSG();
  Delta = (MaxYCPS-MinYCPS)/10;
  MaxYCPS += Delta;
  MinYCPS -= Delta;
  DrawAxis(0, MaxXCSG, MinYCPS, MaxYCPS, 1, 1, "#Strk", "CSG (psi)",
           "Casing Pressure Schedule");
  Function(0, MaxXCSG, MinYCPS, MaxYCPS, 1);
  Table(0, MaxXCSG, "CSG", 1);
}

/*----------------------------------------------------------------*/
/*      SEARCH_MAX_DROP: Searches the largest drop value.         */
/*----------------------------------------------------------------*/

float Search Max Drop(float MinXVect, float MaxXVect)


{
  int i;
  float MaxYVect;

  MaxYVect = 0;
  for(i=floor(MinXVect); i<=floor(MaxXVect); i++)
      if (Drop[i] > MaxYVect) MaxYVect = Drop[i];
  return(MaxYVect);
}

/*----------------------------------------------------------------*/
/*      SEARCH_MIN_DROP: Searches the smallest drop value.        */
/*----------------------------------------------------------------*/

float Search_Min_Drop(float MinXVect, float MaxXVect)

{
  int i;
  float MinYVect;

  MinYVect = Drop[floor(MinXVect)];
  for(i=floor(MinXVect); i<=floor(MaxXVect); i++)
      if (Drop[i] < MinYVect) MinYVect = Drop[i];
  if(MinYVect<0) return(MinYVect);
    else return(MinYVect);
}

/*----------------------------------------------------------------*/
/*      SEARCH_MAX_CSG: Searches the largest CSG value.           */
/*----------------------------------------------------------------*/

float Search_Max_CSG(void)

{
  int i;
  float MaxYVect;

  MaxYVect = 0;
  for(i=0; i<=floor(MaxXCSG); i++) if (CSG[i] > MaxYVect) MaxYVect = CSG[i];
  return(MaxYVect);
```

```
}

/*------------------------------------------------------------------------*/
/*      SEARCH_MIN_CSG: Searches the smallest CSG value.                  */
/*------------------------------------------------------------------------*/

float Search_Min_CSG(void)

{
  int i;
  float MinYVect;

  MinYVect = CSG[0];
  for(i=0; i<=floor(MaxXCSG); i++) if (CSG[i] < MinYVect) MinYVect = CSG[i];
  if(MinYVect<0) return(MinYVect);
    else return(MinYVect);
}
```

TABLE 1

PRE-KICK INFORMATION		
ITEM	DESCRIPTION	UNITS
1.	- HOLE SECTIONS	
	. LENGTH	ft
	. DIAMETER	in
2.	- DRILLSTRING SECTIONS	
	. LENGTH	ft
	. OD	in
	. ID	in
3.	- BIT DATA	
	. NOZZLE SIZES	1/32 in
4.	- SURFACE PLUMBING	
	. MENU SELECTION (SCREEN)	
5.	- PUMP DATA	
	. TYPE (MENU)	
	. LINER SIZE	in
	. STROKE LENGTH	in
	. ROD DIAMETER (Double Acting pumps only)	in
	. PUMP EFFICIENCY	d-less
6.	- CIRCULATING PRESSURES	
	. AT KILL RATE (THROUGH ANNULUS)	
	.. REDUCED PRESSURE	psi
	.. KILL RATE	spm
	. CHOKE LINE FRICTION AT KILL RATE (for floating vessels)	psi
7.	- SAFETY DATA	
	. CASING BURST RATING	psi
	. CASING HOLE SECTION NUMBER (item 1)	d-less
	. AIR GAP (for floating vessels)	
	. FORMATION FRACTURE PRESSURE	psi
	. FORMATION DEPTH	ft
	. BOP RATING	psi
8.	- DIRECTIONAL DATA	
	. MEASURED DEPTH	ft
	. INCLINATION	Deg.

TABLE 2

KICK INFORMATION		
ITEM	DESCRIPTION	UNITS
1	- MUD PROPERTIES	
	. MUD WEIGHT	lb/GAL
	. FANN READINGS	
	.. SPEED	RPM
	.. DEFLECTION	Deg.
2	- PIT GAIN	bbl
3	- SHUT IN PRESSURES	
	. CASING (SICP)	psi
	. DRILLPIPE (SIDPP)	psi

TABLE 4

DIRECTIONAL DATA	
MEASURED DEPTH (ft)	INCLINATION (Deg)
0	0
4200	0
4400	4
4600	8
4800	12
5000	16
5200	20
5400	24
5600	28
5800	32
6000	36
6200	40
6400	44
6600	48
6800	52
7000	56
7200	60
7400	64
7600	68
7800	72
8000	76
8200	78
8400	82
8600	86
8800	90
10000	90

TABLE 3

NEW MUD PROPERTIES		
ITEM	DESCRIPTION	UNITS
1	- MUD WEIGHT	lb/GAL
	- FANN READINGS	
	.. SPEED	RPM
	.. DEFLECTION	Deg.

TABLE 5

MUD PROPERTIES				
	OLD MUD	NEW MUD 1	NEW MUD 2	NEW MUD 3
MUD WEIGHT (lb/Gal)	8.6	9.5	10.5	11.4
FANN VISCOMETER DATA				
SPEED (RPM)	DIAL READINGS (Deg.)			
	OLD MUD	NEW MUD 1	NEW MUD 2	NEW MUD 3
3	1,0	1,0	1,5	1,5
6	1,5	2,0	2,0	3,0
100	15,0	18,0	22,0	24,0
200	26,0	31,0	37,0	39,5
300	35,0	41,0	50,0	53,0
600	56,0	67,0	81,0	88,0

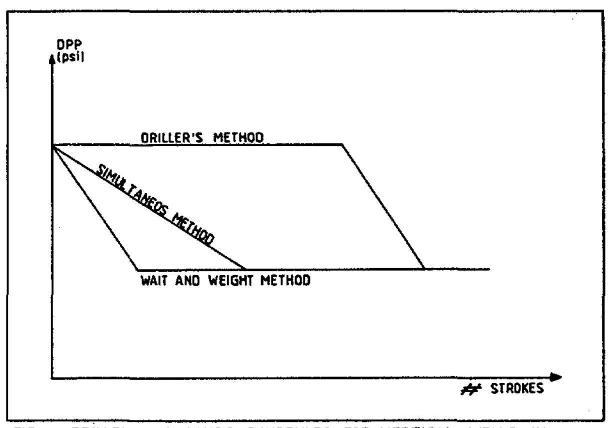

FIG.01: DRILLPIPE PRESSURE SCHEDULES FOR VERTICAL WELLS IN COMMON USE.

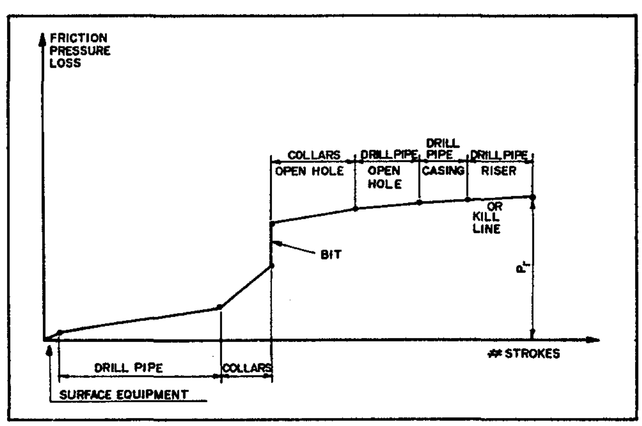

FIG. 02: FRICTION PRESSURE LOSSES CONSIDED IN THE RIGOROUS METHOD

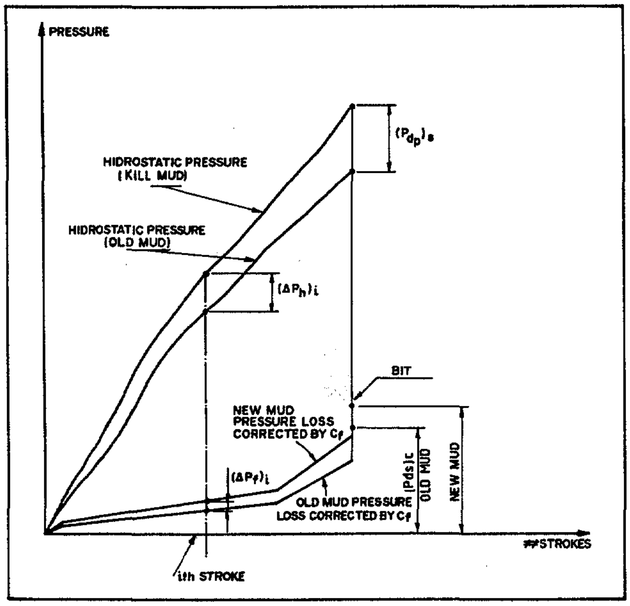

FIG. 03: THE DIFFERENT COMPONENTS CONSIDERED IN THE CALCULATIONS OF THE DRILL PIPE PRESSURE SCHEDULE

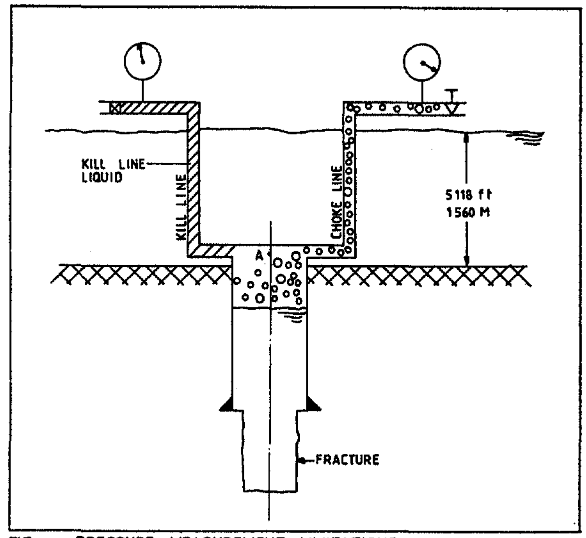

FIG.04: PRESSURE MEASUREMENT LIMITATIONS FOR WELL CONTROL OPERATIONS IN DEEP WATERS.

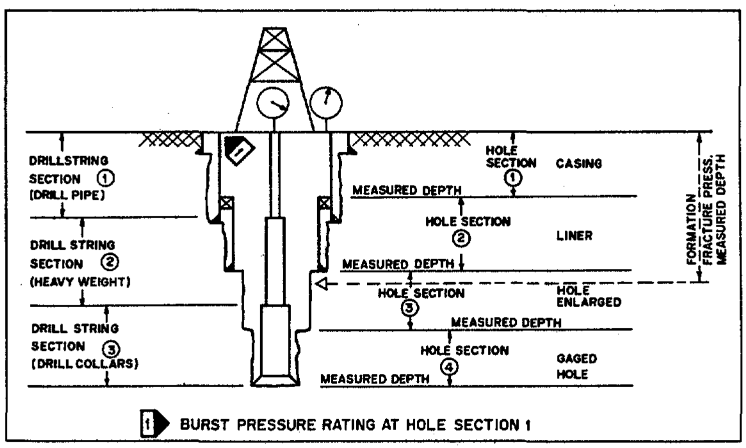

FIG.05: PRE-KICK INFORMATION DIAGRAM FOR LAND RIGS

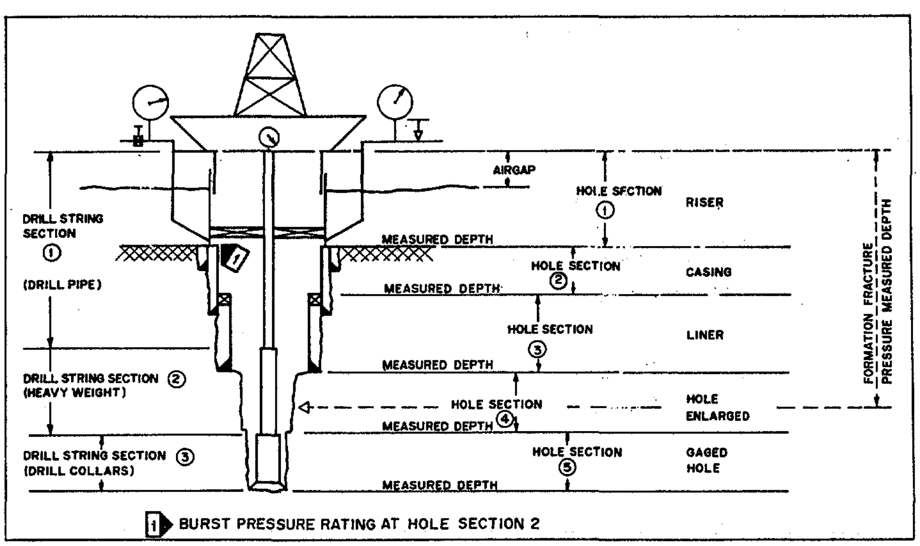

FIG.06: PRE-KICK INFORMATION DIAGRAM FOR FLOATING VESSELS

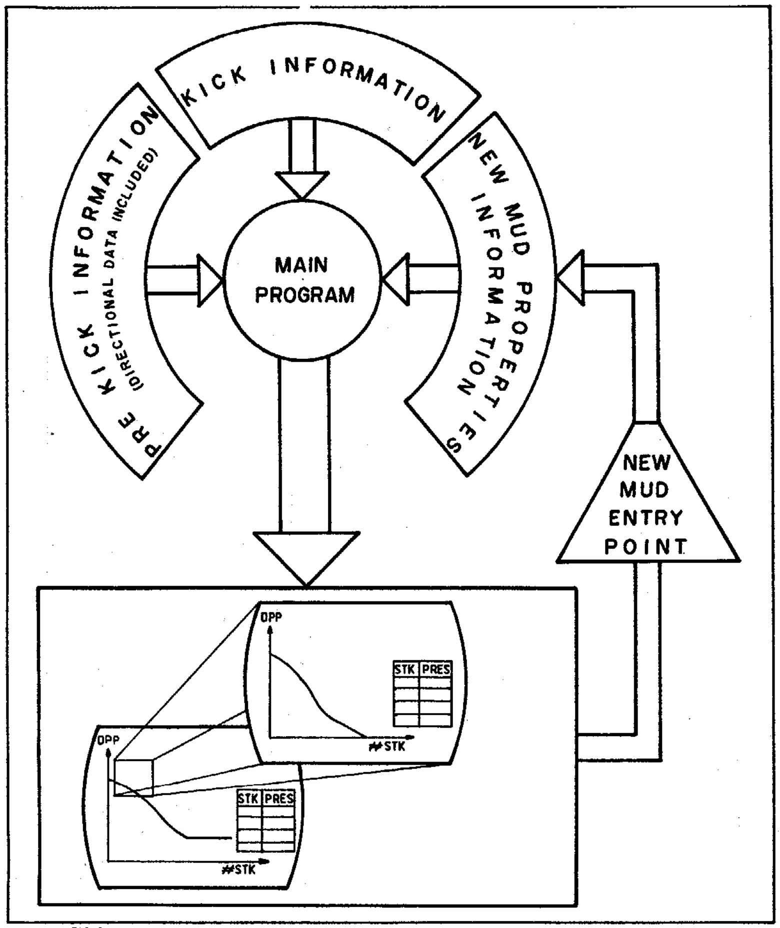

FIG.07 INFORMATION FLOW CHART

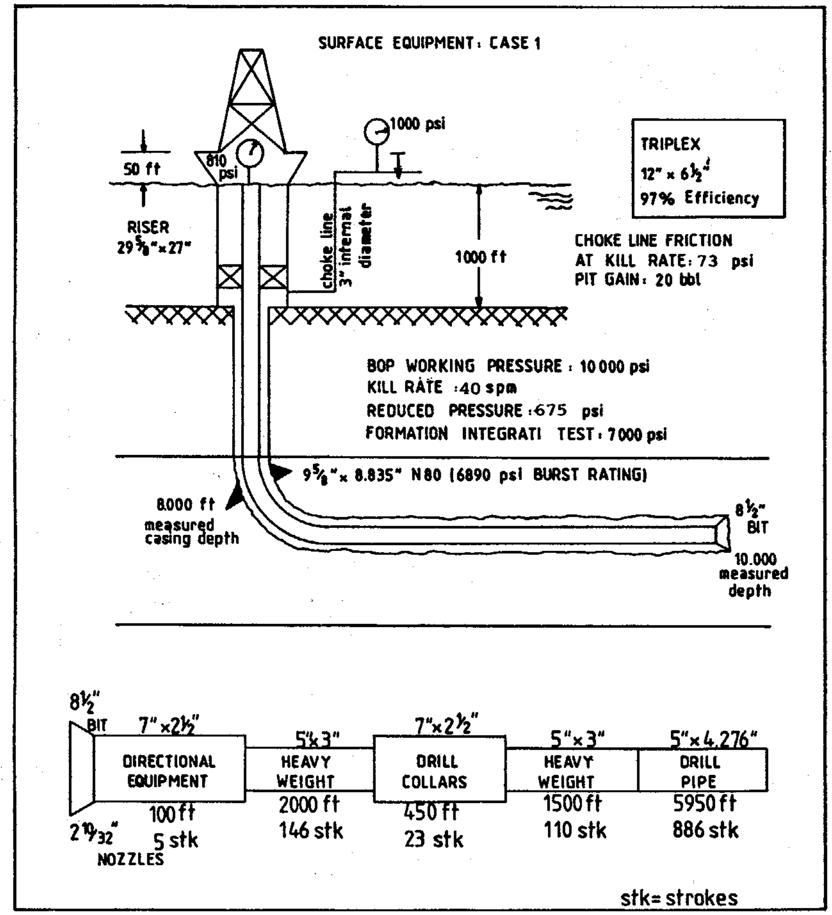

FIG. 08: INPUT DATA FOR CASE STUDY #1

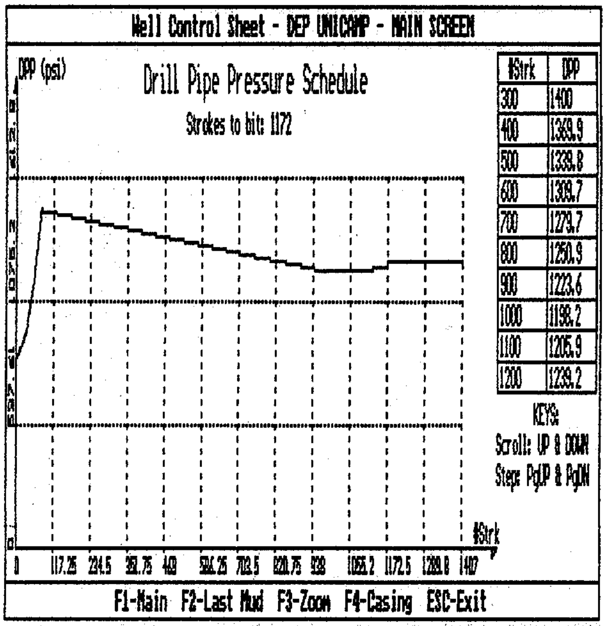

FIG.09: DRILLPIPE PRESSURE SCHEDULE FOR THE FIRST DRILLING FLUID OF 9.5 lb/gal.

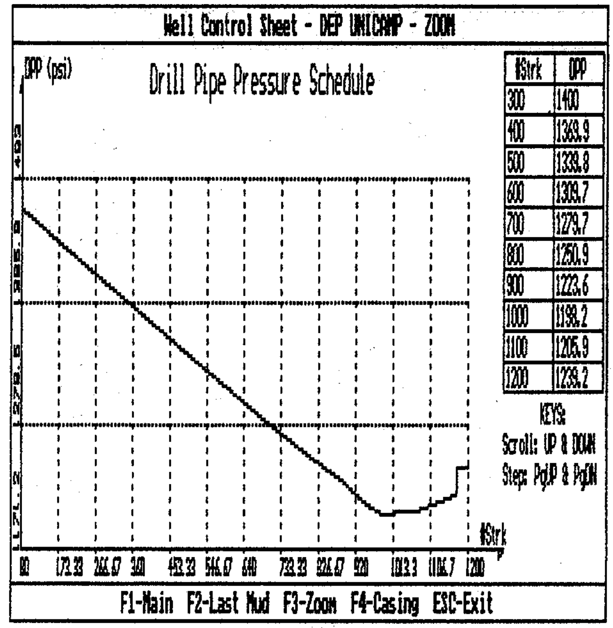

FIG.10: ZOOM ON 80 TO 1200 STROKES FOR THE FIRST DRILLING FLUID OF 9.5 lb/gal.

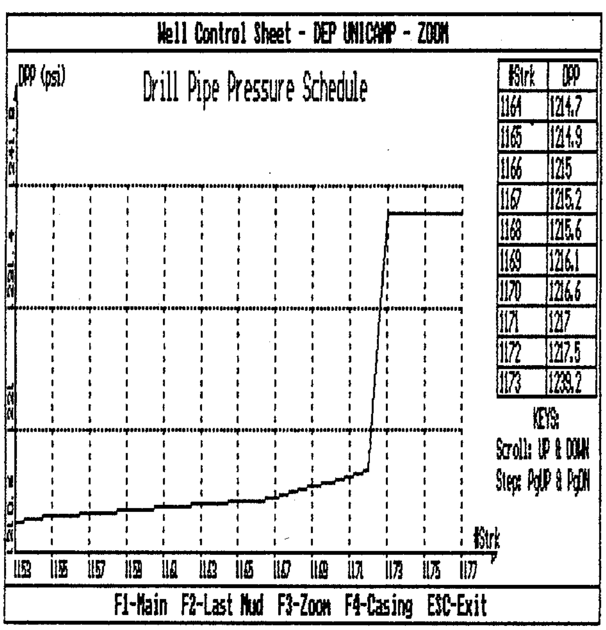

FIG.11: PRESSURE CHANGES AT THE BIT FOR THE FIRST DRILLING FLUID OF 9.5 lb/gal.

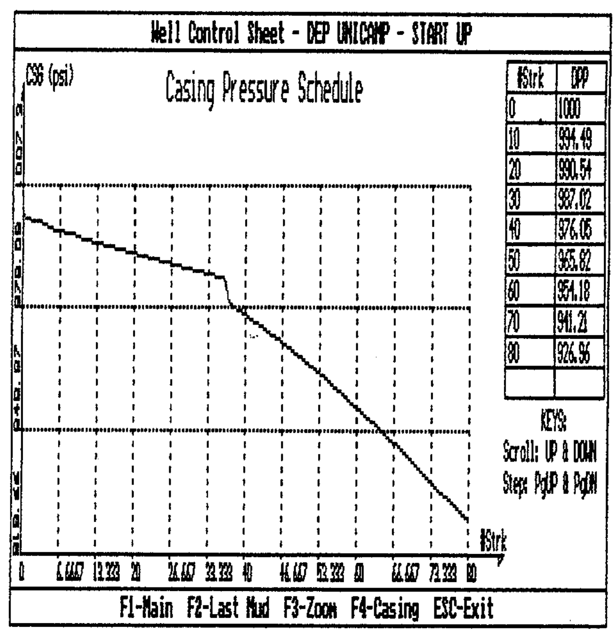

FIG.12: CASING PRESSURE SCHEDULE FOR THE FIRST DRILLING FLUID OF 9.5 lb/gal

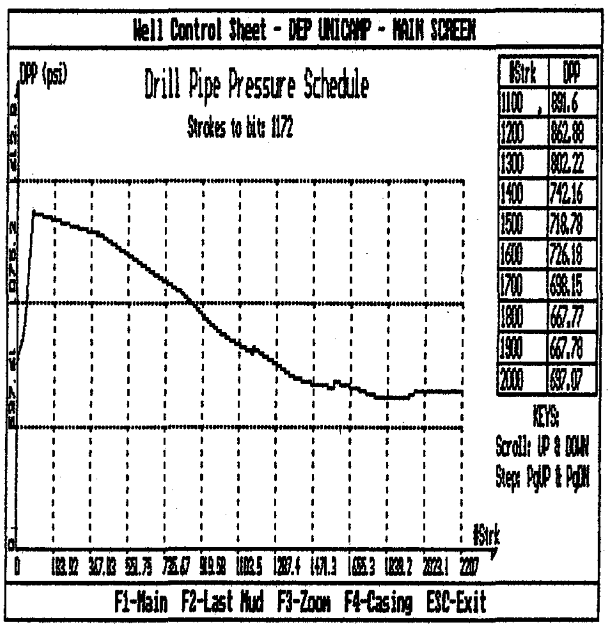

FIG.13: DRILLPIPE PRESSURE SCHEDULE FOR ALL THE DRILLING FLUIDS SHOWN IN TABLE 5 (400 STROKES APART.

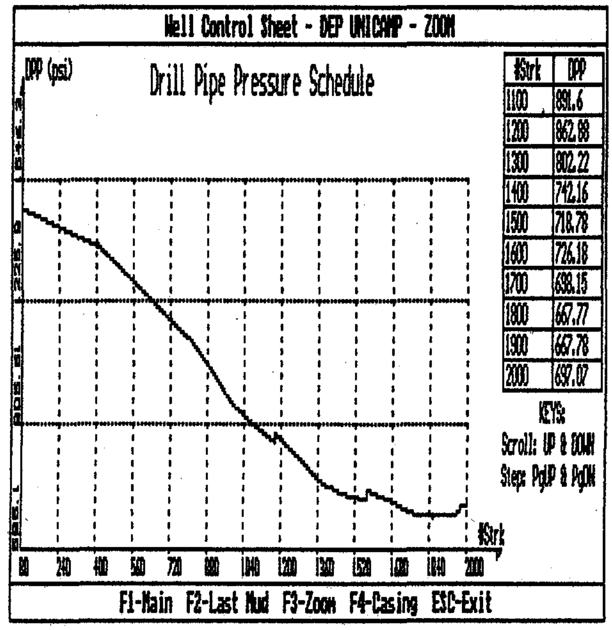

FIG. 14: ZOOM ON 80 TO 2000 STROKES FOR ALL THE DRILLING FLUIDS.