CLAS - NOTE : 95 - 018

August 17, 1995

# A 'C' Program that Formats and Translates Binary Data Files Logged from the CLAS Toroidal Magnet

*Authors: Kelly Alvord, Peter Bonneau, Meenatchi Jagasivamani*

***Abstract:*** The program Taclcsv.C is designed to read a binary input file which consists of signals from the CLAS toroidal magnet. The program converts the binary values to decimal values and prints the results either to the screen or to a user-specified output file. Modifications to the program make it user-friendly and allow for specifically formatted data to be printed. These modifications are presented in this paper.

# Introduction

### *Torodial Magnet*

The CLAS toroidal magnet consists of six current carrying coils. The magnet is cooled with liquid helium, in order to make it superconductive. The pressure and temperature of the liquid helium before and after input, the temperature of the six coils, the pressure in the vacuum tank surrounding the helium, are crucial variables for the magnet and must be monitored constantly. To accomplish this, signals are read by a computer every 5 minutes and stored in a binary file format. Directories are created for each signal source. For example, a directory called PI1028 has been created for the files holding pressure signal of a tank. For any given variable, a new file is created each day, to be stored in its respective directory. A computer program, **Taclcsv.C**, has been designed to read the binary files and provide specific values of variable which are of interest to the user.

### *Taclcsv.c Program*

The C program 'Taclcsv' is designed to extract data from the toroidal magnet's databases. It allows the user to choose **only** the pieces of data s/he is interested in analyzing. The data is logged to a file created at the beginning of each day, that contains only that day's readings. The file name is the date on which the data was taken. All the readings are written to this file in a 32-bit binary format, which is later read by Taclcsv.c and translated and output for the user in the desired format. The user may choose to output all the readings to an output source, such as another file or the screen. It is possible to output the data taken after a given time from the start of the file. Debugging information is also an option. The time period that is desired must be input as a subtraction

amount from the current day. For example, it is possible to get all the information from -19 days ago. The program lists the time at which all the readings were taken on that day and the corresponding data (values) of the instruments.

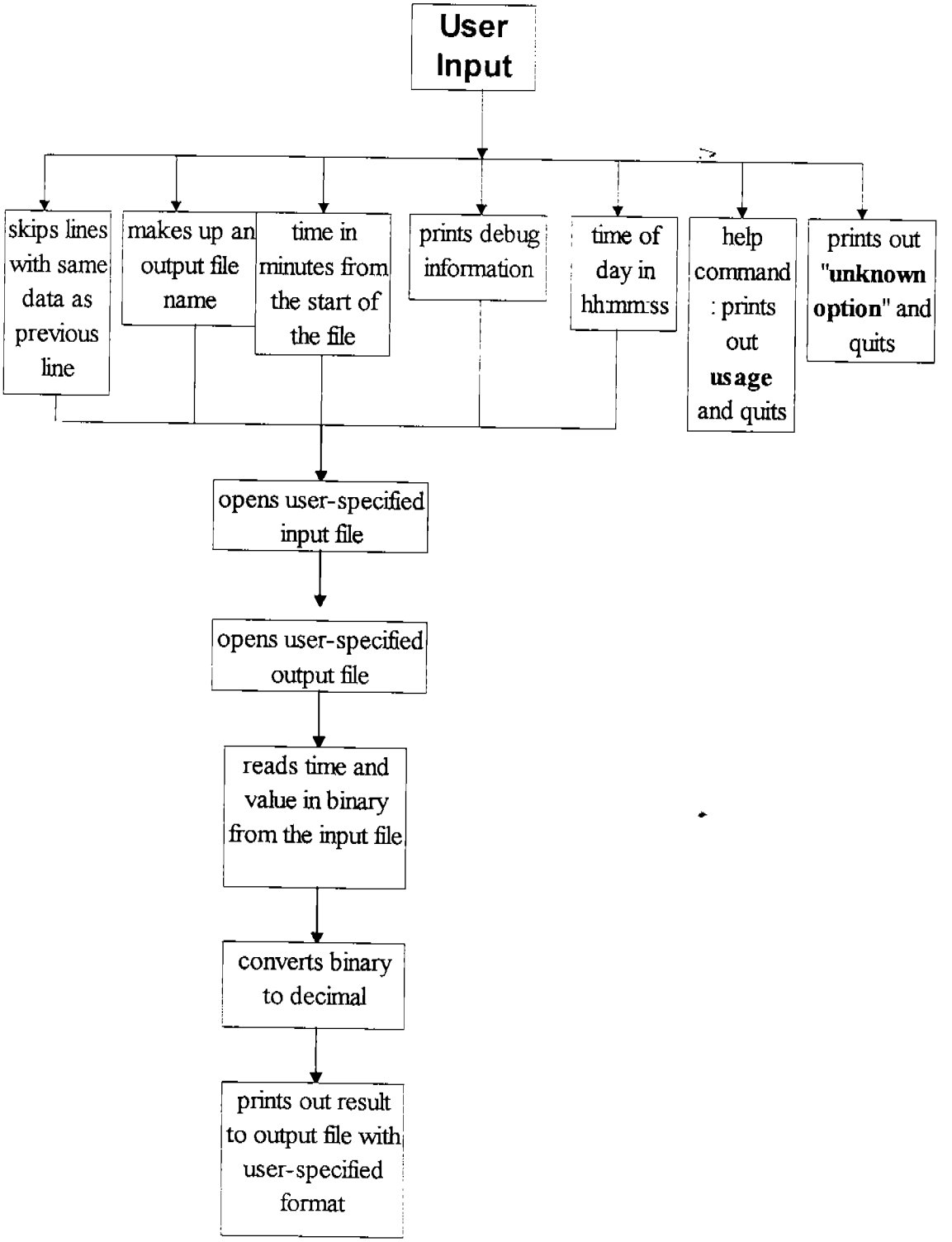
This program is useful in its present state, however, some additional options are added to make it easier and more efficient. The add-ons include:

1) allowing the user to receive the average of the data, and the recorded minimum and maximum value during the given period of time.
2) calculating the differential during a period of time, that is the amount of change from the highest reading to the lowest reading.
3) allowing only the specific data choice to be printed to the screen, thus not including the default list of **all** data.
4) allowing the requested time period to be inputed by calendar dates.
5) allowing the time period to be either
    a) past date to current date (inputing only one date),
    b) only one past date (inputing only one date),
    c) between two dates in the past. (Inputting two dates).
6) outputting the data in columns for better readability and also so that the data can be transferred into the spreadsheet based program EXCEL.

***Program Structure***

The program is started with the name of the program (Taclcsv.c) and an option chosen for a specific format. Current options include skip lines with same data as previous line, print debug information, makes up an output file name, time in minutes from the start of file, time of day in hh:mm:ss. There are other operations that are performed if the user requires help or if an incorrect option is entered. After an appropriate option is chosen, the user-specified input and

output (if any) files are opened. The value and its respective time are read in binary from the input file. Binary is converted to decimal and printed out in the user-specified format. A flowchart outlining the program structure has been included. Also the program and a line-by-line commentary are included.




**Flowchart of program Taclcsv.c**

# Taclcsv.C

## A: Initializing/ Declaration

```
A1./*  Taclcsv.c: for formatting log files into comma separated variables. */
A2./* Format of 32-bit FP numbers:
A3.     top bit (31)= sign
A4.     bit 30- don't know what this does
A5.     its 29-23= signed exponent (powers of 2)
A6.     its 22 to 0= 24 bit number, with bit 23 understood to be 1
A7.*/
A8.#include <stdio.h>
A9.#include <time.h>
A10.char buffer[64], dummy[16];
A11.main (ac,av)
A12.int ac;
A13.char *av[];
A14.{
A15.  FILE *Fpi, *Fpo;
A16.  int m,n, tmp, mday, bit, shft;
A17.  int flgskp, flgdbg, flgmin, flgdat, flgnam, flgtim, flgshft, step;
A18.  long tim, timt, lstd=-9999999L, dat, tstart;
A19.  float f1,f2;
A20.  char byte[8], tnam[32], inam[16], onam[16], tbuf[16, *dptr, *sptr, *tptr;
A21.  struct tm  *tmptr;
A22.  time_t timet;
A23.  flgskp=flgdbg=flgmin=flgdat=flgnam=flgtim= 0;
A24.  mday=0;
A25.  Fpo=stdout;
A26.  timet=(time_t)timt;
A27.  time(&timet);
A28.  tmptr=localtime(&timet);
A29.  timt=1000L * tmptr->tm_year +tmptr->tm_yday +1;
```

# Taclcsv.C

## B: Interpret Command Line

```
B1.   /* Interpret command line: */
B2.   if (ac==1) error ("taclcsv -h for help");
B3.   for (step=1; *av[step] == '-'; ++step);
B4.   {       if (*(sptr=(av[step]+1))=='s') flgskp =1;
B5.           else if (*(av[step]+1)=='d') flgdbg=1;
B6.           else if (*(av[step]+1)=='o') flgnam=1;
B7.           else if (*(av[step]+1)=='m') flgmin=1;
B8.           else if (*(av[step]+1)=='t') flgtim=1;
B9.           else if (*sptr=='h')
B10.          {       fprintf (stderr,
B11.          "\t Usage: taclcsvv [-s] [-m] [-o] [-d] [-t] var [/yyddd] [outfile]; options:
              \n);
B12.                  fprintf(stderr,"-s, skip lines with same data as previous line
                      \n");
B13.                  fprintf(stderr,"-o, make up an output file name \n");
B14.                  fprintf(stderr,"-m, time in minutes from start of file \n");
B15.                  fprintf(stderr,"-t, time of day in hh:mm:ss\n");
B16.                  fprintf(stderr,"(-m -t gives hh:mm from start of file)\n");
B17.                  fprintf(stderr,"-N,N=number of days ago\n");
B18.                  fprintf(stderr,"-d, print debug info\n");
B19.                  fprintf(stderr,"Today is day %5ld\n", timt);
B20.                  exit(0);
B21.          }
B22.          else
B23.                  {       if (sscanf(sptr,"%d, &tmp)!=1)
B24.                                  fprintf(stderr,"unknown option: %c\n:,*sptr);
B25.                          else mday=tmp;
B26.                  }
B27.  }
B28.  if (flgdbg) fprintf (stderr,"options %d%d%d%d, step=%d\n",flgskp,
                                  flgdbg,flgnam, flgmin,step);
```

# Taclcsv.C

## C: Open Input/Output File

```
C1./* Open input file: */
C2.    for (sptr = av[step]; *sptr != '\0'; ++sptr)
C3.    if ((*sptr == '/') || (*sptr == '\\'))
C4.    {
C5.            sprintf(inam, "%s", av[step]);
C6.            break;
C7.    }
C8.    else  sprintf(inam, "%s/%5ld", av[step], timt - mday);
C9.    if ((Fpi = fopen(inam, "rb")) == NULL)  error(inam);
C10./* Open output file: */
C11.   if (flgnam != 0)        /* make an output file name var.ddd */
C12.   {
C13.           sprintf(tnam, "%s", av[step]);
C14.           for (tptr = tnam; *tptr != '\0'; ++tptr) ;
C15.           dptr = tptr - 3;          /* point at day */
C16.           for (tptr -= 3; tptr > tnam; --tptr)  if (*tptr == '/')  break;
C17.           /* back to end of directory */
C18.           for (*tptr = '\0'; tptr >= tnam; --tptr)  if (*tptr == '/')  break;       /* to
                                                                    start of directory */
C19.           ++tptr;/* point at variable name */
C20.           if ((int)(dptr - tptr) > 11)
C21.                   tptr = dptr - 11;       /* else too long for DOS */
C22.           sprintf(onam, "%s.%s", tptr, dptr);
C23.           if (flgdbg != 0)  fprintf(stderr, "%s, %s, making %s\n", tptr, dptr,
                                                                    onam);
C24.           if ((Fpo = fopen(onam, "wt")) == NULL)  error(onam);
C25.   }
C26.   else if (ac> step+1)
C27.   if ((Fpo=fopen(av[step+1], “wt”))==“wt”))==NULL) error (av[step+1]);
```

# TacIcsv.C

## D: Read Data in 8 byte blocks

```
D1./*Read data in 8 byte blocks, to end of file.*/
D2.for (n=0;;++n)
D3.{    if ((tmp=getc(Fpi))==EOF) break;
D4.byte[0]=(char)tmp;
D5.for (m = 1 ; m < 8; ++m)  byte[m] = getc(Fpi);        /* 8 bytes */
D6.tim = 0xff & (long)byte[0];
D7.tim = (tim << 8) + (0xff & (long)byte[1]);
D8.tim = (tim << 8) + (0xff & (long)byte[2]);
D9.tim = (tim << 8) + (0xff & (long)byte[3]);
D10.timet = (time_t)tim;
D11.if (flgmin)
D12.{           if (n == 0)
D13.            {       tstart = tim;   /* first line: time in seconds from 1970 */
D14.                    (void)fprintf(Fpo, "\'%s", ctime(&timet));
D15.            }                       /* first line header: 'time */
D16.            tim -= tstart;  /* time in seconds from start of data */
D17.            tim /= 60;
D18.}
D19.else  if (n == 0)/* first line header: 'time */
D20.(void)fprintf(Fpo, "\'%s", ctime(&timet));
D21.if (flgtim)
D22.{           if (flgmin)  sprintf(tbuf, "%2d:%2d", tim/60, tim%60);
D23.            else
D24.            {               timet = (time_t)tim;
D25.                            (void)sprintf(buffer, "%s", ctime(&timet));
D26.                            sscanf(buffer, "%s%s%s%s", dummy,  dummy,
                                                    dummy,  tbuf);
D27.            }
D28.}
D29.else  sprintf(tbuf, "%ld", tim);
D30.dat = 0xff & (long)byte[4];
D31.dat = (dat << 8) + (0xff & (long)byte[5]);
D32.dat = (dat << 8) + (0xff & (long)byte[6]);
                        dat = (dat << 8) + (0xff & (long)byte[7]);
```

# Taclcsv.C

## E: Translate Binary Data

```
E1./* Translate 32-bit FP: */
E2.bit = (0xff & (int)byte[4]) >> 6;  /* top 2 bits */
E3.shft = (int)((0x3f800000L & dat) >> 23);       /* next 7 bits = exponent */
E4.if ((shft & 0x40) == 0x40)  shft |= 0xffffff80;   /* extend sign */
E5.if (dat == 0L)  f1 = 0.0;  /* special case, I think */
E6.else  f1 = (float)(0x800000L | (0x7fffffL & dat));      /* add top bit */
E7.if ((2 & bit) == 2)  f1 = -f1;       /* sign bit */
E8.f2 = 4194304.0;          /* = (2**22) */
E9.flgshft = 0;
E10.if (shft < 0)
E11.{           if (shft < -15)  flgshft = 1;
E12.            else  f2 *= (float)(1 << (-shft));
E13.}
E14.else
E15.{           if (shft > 15)  flgshft = 1;
E16.            else  f2 /= (float)(1 << shft);
E17.}
E18.if ((flgskp != 0) && (dat == lstd))  continue;
E19.lstd = dat;
E20.if (f2 == 0.0)  f2 = f1;
E21.else  f2 = f1/f2;
E22.if (flgdbg != 0)
E23.{           (void)fprintf(stderr, "%4d:%5ld, %8lx; ", n, tbuf, dat);
E24.            (void)fprintf(stderr, "bits=%1x%1x, shft=%3d, num=0x%lx=%.0f,
                        %f\n", bit>>1, 1&bit, shft, (long)f1, f1, (f2 == 0.0) ? f1 : f1/f2);
                                                }
```

# Taclcsv.C

## F: Print Result

```
F1./* print result: */
F2.     if (flgshft)
F3.     {
F4.             (void)fprintf(Fpo, "%s, ???\n", tbuf);
F5.             (void)fprintf(stderr, "shft = %d, dat = %8lx\n", shft, dat);
F6.     }
F7.     else
F8.     {               if (f2 > 0.9999)
F9.                     {
F10.                            if (f2 > 99.9)
F11.                            (void)fprintf(Fpo, "%s, %.3f\n", tbuf, f2);
F12.                            else if (f2 > 9.99)
F13.                            (void)fprintf(Fpo, "%s, %.4f\n", tbuf, f2);
F14.                            else  (void)fprintf(Fpo, "%s, %.5f\n", tbuf, f2);
F15.                    }
F16.                    else if (f2 < -0.9999)
F17.                    {
F18.                            if (f2 < -99.9)
F19.                            (void)fprintf(Fpo, "%s, %.3f\n", tbuf, f2);
F20.                            else if (f2 < -9.99)
F21.                            (void)fprintf(Fpo, "%s, %.4f\n", tbuf, f2);
F22.                            else  (void)fprintf(Fpo, "%s, %.5f\n", tbuf, f2);
F23.                    }
F24.                    else  (void)fprintf(Fpo, "%s, %.6f\n", tbuf, f2);
F25.    }
F26.}
F27.timet = (time_t)tim;
F28.if (flgskp)  (void)fprintf(Fpo, "\'%s", ctime(&timet));
F29.}
F30.error(s)
F31.char        *s;
F32.{
F33.perror(s);
F34.exit(1);
F35.}
F36./* end taclcsv.c  */
```

# Taclcsv.C Commentary

## A: Initializing / Declaration Page

**A1** Comment line. Tells reader the name of program and purpose

**A2-7** Comment line. Tells the reader about the structure of the data that is worked with in this program.

--32 bit Floating point number

--bit #31 is designated sign (+/-) of number

--bit #30

--bit #29-23 signed exponent (powers of two)

--bits #22-0 24 bit number with bit #23 understood to be 1

**A8** #include<stdio.h> Command that accesses the library stdio.h. this library file contains all normal and standard functions. Is commonly a part of every c program.

**A9** #include<time.h> Command that accesses the library containing all the time functions.

**A10** Declarations of the arrays BUFFER to contain 64 fields of information, and DUMMY to contain 16 fields.

**A11** main(ac,av). This is the signal to the computer that this is the beginning of the main program. It must be in every program. Ac, and av represent the information being sent into the program. It is not necessary to send the program anything. Ac represents the number of arguments inputted by the user at the prompt, including the program name. Av represents each argument in an array form, not including the program name. Ex. Av[2]='-o'

**A12** Declaration for the variable ac, that is sent into the main program. ac=integer

**A13** Declaration for the variable av, that is sent into the main program. av=pointer to an array with variable number of fields.

**A14** { Open bracket representing the inside of the main program. Encompasses the entire main program.

**A15** FILE File is a type statement that functions as telling the computer how to store the data. * represents a pointer. Pointers are used to point to information and move through a variable field without changing values.

**A16** Declaration for integer variables. **Instructs** computer to reserve 8 bits for an integer value.

**A17** Declaration for integer variables.

**A18** Declaration for long integer variables. Long is a type statement, which differs from integer only in the number of bits reserved for the number.

**A19** Declaration for floating variables. Float is a type statement that is used for numbers including a decimal point.

**A20** Declaration for character arrays, and character pointers. *=pointer; [n] represents the number of fields reserved in the array.

**A21** Declaration statement for a user defined type. This type is now called tm(which is standard for time functions) and contains a pointer. Struct is the command for declaring user defined variables.

**A22** Declaration for time variable. time_t is a type statement, like int and char.

**A23** Beginning of initialization portion of program. Initializes all variables to be zero (false).

**A24** Initializes variable to zero.

**A25** Assigns Fpo(file pointer) to the predefined file STDOUT (standard output =screen)

**A26** This statement will change the variable timt into a time_t type and place it in the variable timet.

**A27** time is a function that returns the current standard time. This statement will assign the current time to the address of the variable timet.

**A28** Localtime is a function that produces the local time (including daylight savings time) and returns that into the address of timet. This value will then be pointed to by the pointer tmptr.

**A29** This statement puts the time into the desired format. The current years, given as a number of years after 1900 year (timptr->tm_year) is multiplied by 1000 and added to the number of the day past January 1st (tmptr->tm_yday). 1 is then added to account for the current day. For example if the date was Dec 15, 1995, (on a non leap year) this statement would first multiply 1000 by 95= 95000. Then 349 would be added to this =95349. Finally the current day would be added resulting in 95350. this value would be stored in timt.

# TacIcsv.C Commentary

## B: Interpret command line

**B1** Comment line that signals the beginning of the section that interprets input from the user.

**B2** If statement that checks to see if the user has entered only one argument, and gives an error if necessary. Ac is the variable sent that represents the number of arguments inputted.

**B3** Beginning of for loop that is reading the characters in the input line.

**B4** { open bracket enclosing for loop. If statement that checks if the character pointer after the '-' is an 's'. If true, then it assigns variable flgskp to be 1.

**B5** else if statement that checks to see if the pointer is a 'd' (debugging info) and assigns flgdbg to be 1 if true.

**B6** else if statement that checks to see if the pointer is a 'o' (output file name) and assigns flgnam to be 1 if true.

**B7** else if statement that checks to see if the pointer is a 'm' (time in minutes) and assigns flgmin to be 1 if true.

**B8** else if statement that checks to see if the pointer is a 't' (time of day) and assigns flgtim to be 1 if true.

**B9** else if the character is 'h'

**B10** {open bracket for else if statement for 'help' print to stream stderr (which is screen) the line that shows possible commands. Note: *fprintf* is used when a specific stream is desired, and allows the user to choose where it will be printed.

**B11** print statement that gives the function of the 's' character

**B12** print statement that gives the function of the 'o' character

**B13** print statement that gives the function of the 'm' character

**B14** print statement that gives the function of the 't' character

**B15** print statement that gives the function of both 'm' and 't' together

**B16** print statement that gives the function of the 'N' character

**B17** print statement that gives the function of the 'd' character

**B18** print statement that prints out the current date for the user

**B19** exit(0) the exit command will stop the program. The 0 means that the computer has stopped without an error. 1 means that an error has occurred that caused the program to stop.

**B20** } closed bracket of else if help loop.

**B21** else from help if loop. (means if the command entered is not one of the given choices)

**B22** { open bracket-else

**B23** if statement that checks to see if a integer number read from stderr is != 1.

**B24** if true then print to stderr a message stating unknown option

**B25** if false then assign temp to mday.

**B26** } closed bracket-else

**B27** }closed bracket-for loop (B3)

**B28** if statement that prints all the values of the variables to stderr for debugging purposes flgdbg is true.

# Taclcsv.C Commentary

## Open input and output files

**C1.** Comment line which marks the function of the section. Open input file.

**C2.** For loop which sets standard pointer to parameter of input filename, initially. The loop will go on only if the condition is met, that is until the pointer has not met the end of string. While this condition is true, the pointer will move to the next character.

**C3.** If statement which runts its statements if the conditions are true, which is that there is a forward slash or a backward slash (/ \\) before the file name (for opening file)

**C4.** { Open bracket for if statement

**C5.** Prints a stream (the filename parameter av[step]) to an array inam.

**C6.** Statement that quits if loop;

**C7.** } closed bracket -if

**C8.** else statement which gets executed if the condition of the if statement (**C3**) is false. The filename (av[step]) and the formatted date (**A29**) will be written to the array inam.

**C9.** tries to open the input file, using the fopen command. It assigns the file to Fpi and sets the mode to read binary. If the file is empty, it gives an error message by executing the subroutine "error".

**C10.** comment line - labels next section "open input file".

**C11.** if statement with condition that must be true to get executed. The condition is that the variable flgnam must not be true to get executed. This will only be true if the option "o" is chosen which makes an output file name var.ddd as the following comment line states.

**C12.** { Open bracket for if statement

**C13.** prints the parameter which will hold the output file name to the array tnam.

**C14.** for loop with initial statement that sets the character pointer tptr to the first character of the array tnam. While pointer has not reached the end of the array, the pointer will move to the next character. The for loop ends in the same line without any statements to be executed. The goal of the for loop must for setting the pointer tptr to the last character of the array tnam, where the loop stops.

**C15.** statement which sets the pointer dptr 3 spaces before tptr, i.e., 3 spaces before the end of array which will be make dptr point to the first letter of the day. The comment

line states that dptr will "point at day". The day is the extension part(.ddd) in the filename.

**C16.** for loop with initial statement setting tptr 3 spaces before its current location which will point it to first character of the day in the formatted date file name. While the condition that tptr be greater than the value of tnam is true, the pointer moves one place to the left. There is a if statement inside the for loop, which checks to see if a backslash is there. A backslash will mean that the file is in a directory. If a backslash is there, then the for loop will end.

**C17.** comment line that states that the pointer is pointing to the end of directory.

**C18.** for loop with initial statement setting tptr to point at the end of the array tnam. The condition is that tptr be greater than or equal to tnam and while this is true the pointer gets moved 1 space to the left. The if statement in the for loop checks for a backslash-which refers to a directory- in the array. If a backslash is found, the if statement will quit.

**C19.** statement which increments pointer one space to the right so that the pointer is pointing at the variable name, as the comment block states.

**C20.** if loop with condition that checks if the difference in the spacing between tptr and dptr is greater than 11.

**C21.** statement of if statement that cuts extra characters out of the variable name because its too long for DOS. The comment line states that DOS can only accept filenames that are less than 11 characters.

**C22.** prints the filename with the name and extension to the array onam.

**C23.** if statement that executes if the variable flgdbg is not equal to zero (option "d"). When executed, it prints out a line stating that the filename is being created.

**C24.** if statement that checks to see if the output file exists by opening it. If it doesn't exist, then an error message is printed out.

**C25.** close bracket - if

**C26.** else if statement which will only get executed if the main if statement's condition was found to be false, that is if flgnam was equal to zero(if an option other than "o" was choosen), and if the else if statement's condition is found to be true, which is that if there is one more parameter.

**C27.** if statement inside else if statement with condition that tries to opens the parameter, assuming that the parameter is file. If the file does not exist, an error message is printed to the screen and the program is exited.

# Taclcsv.C Commentary

## D: Reading data

**D1.** comment line states function - "read data in 8 byte blocks, to end of file"

**D2.** for statement with initial condition setting the integer variable n to be zero (for counting purposes. There is no condition in the for statement, so it will increment by one each time it repeats.

**D3.** open bracket - for statement. if statement with condition of getc(Fpi), which reads a character from the opened file, Fpi, and checks to see if it is EOF(end of file character). If it is, then the for statement quits. This is the only way out of the for loop since it doesn't have any conditions for the for. The character acquired is assigned to variable tmp.

**D4.** The variable byte is an array of characters which can hold 8 bits. Since the first 8 bits have already been read in the if statement (**D3**), and has been assigned to tmp, byte[0] is assigned to tmp and any extra bits are truncated using (char)tmp command.

**D5.** for statement that goes from 1 to 7 with increment of 1 with each repetition. Each time it loops, byte[1] thru byte[7] are assigned binary data of 8 bits each.

**D6.** assigns byte[0] to tim (long -- 32 bits). The command 0xff & byte[0] intializes byte[0].

**D7. - D9.** shifts tm 8 bits to the left (to allow for next 8 bits) and adds it with initialized byte.

**D10.** assigns value of tim(which is the binary value read from file representing the time) to timet by converting it to the same type as time_t.

**D11.** if statement with condition being that flgmin be any number other than zero

**D12.** begin bracket of if statement. Another if statement is inside the if statement with condition that n has to be zero. This loop will get executed on the first time, to print time on the first line.

**D13.** begin bracket of if statement. assigns tim to tstart, followed by comment block that says - "first line: time in seconds from 1970*/

**D14.** void command tells the compiler that the following function is not going to return a value. fprintf prints to a file, Fpo, a string from ctime(&timet).

**D15.** close bracket - if statement. comment line states - "first line header: 'time'"

**D16.** subracts start and tim, and assigns the difference to tim.

**D17.** divides tim by 60 and assigns the quotient to tim.

**D18.** close bracket - if statement.

**D19.** else if statement that gets executed only if flgmin was equal to zero ( the if statement didn't get executed), and if n is equal to zero, the first time. Comment line following it says - "first line header: 'time'".

**D20.** statement inside else if statement that doesn't return a value (void), and prints a string time(&timet) to a file, Fpo.

**D21.** if statement with condition that flgtim be any number that does not equal to zero.

**D22.** open bracket - if statement. If statement with condition that flgmin be any number that does not equal to zero. If condition is true, then tim divided by 60 and tim mod 60(remainder of tim/60), are printed to the array, tbuf.

**D23.** else statement that gets executed if the if statement's condition is false.

**D24.** begin bracket - else statement. assigns tim to timet changing the format of tim, but the value and type of tim is unchanged.

**D25.** void states that no value is returned by the function sprintf, which prints the string ctime(&timet)) to the array buffer.

**D26.** reads 4 strings from the array buffer and assigns them to the variables dummy and tbuf.

**D27.** close bracket - else statement

**D28.** close bracket - if statement.

**D29.** else statement that prints the long integer tim to the array tbuf.

**D30. - D33.** follows same format as lines **D6 - D9**, replaced with dat and byte[4] thru byte[7] instead of tim and byte [0] thru byte[3], respectively.

# Taclcsv.C Commentary

## E: Translate 32 bit

**E1** Comment line—Section that translates the 32 bit floating point number.

**E2** Open and reads the top two bits (sign of the number) by using the & function with the hexadecimal (Oxff) which in binary is 1111 1111. This will take the binary code in byte[4] and add it to the 0xff resulting in a 1 only if both numbers are 1 and a 0 when both are not. The number is then shifter 6 units to the right and assigned to the variable bit.

**E3** This opens and reads the next 7 bits that are used for the exponent. The value of variable dat is added(&) to 0x3f800000, which in binary is 0011 1111 1000 0000 0000 0000 0000 0000. This in effect opens the 7 digit exponent. The result is shifted 23 spaces to the right and placed in the variable shft.

**E4** This line extends the sign so that it will be effective in different types of machines. First shft is &-ed to 0x40 and then compared to 0x40. If this is true then shft will perform a bitwise | (or) with 0xffffff80 and then be placed back into the variable shft. If the sign of the number is set to 1 then this statement will also set the top bit to be 1 so that the machine is sure to understand the sign of the exponent.

**E5** This is a special case when the dat is zero. If it is then f1 is set to be 0.0.

**E6** else if dat is not zero then dat is &-ed (and) to 0x7ffff and the result is |-ed (bitwise or) to 0x800000. This is then placed into the variable of f1. This connects both the exponent and then top bit.

**E7** If bit and 2 (01) are &-ed and the result is equal to 01 then the sign of f1 is reversed.

**E8** Assigns the value of 4194304.0 to f2. ($2^{22}$) This will be used for normalizing the numbers at a later time.

**E9** Assigns flgshft to be 0.

**E10-13** If shft is less than 0 (negative number) then it checks to see if it is less than -15. If it is, flgshft is set to 1, else negative shft is shifted 1 space to the left (to knock of the sign bit) and then multiplied to f2. The result is placed back into the variable f2. This is the negative case of shft.

**E14-17** Else is shft is greater than 0 and greater than 15 then flgshft is set to 1. Else shft is shifted 1 space to the left and then is divided by f2 and the value is set back to f2. This is the positive case of shft.

**E18** If statement that checks to see if flgskp is not equal to 0 and dat = lstd. && is a logical AND command which returns 0 if false and 1 if true. If this statement returns

true then the command continue caused execuion to jump immediately to the beginning of the loop, skipping the remaining lines.

**E19** Assignment statement that transfers the value of dat to lstd.

**E20** If statement that checks the value of f2. If its 0.0 then f1 becomes f2.

**E21** else if f2 does not equal 0 then f1/f2

**E22** If flgdbh is not equal to 0. This statement checks tosee whether a debugging choice was entered by the user.

**E23** Open { bracket-- fprintf statement that prints the value of n, tubuf, dat, to 4 decimal places, 5 long decimal places, 8 long hexidecimal numbers respectively. The (void) tells the compute that the functions will not return a value.

**E24** fprintf statement that prints to stderr, all the variable values including bit(shifted one position), bit, shft, f1, f2 . The phrase "(f2==0)?f1:f1/f2" is a conditional statement that is read by the computer like an if/else statement. It is understood to be "If f2=0 then print f1 else print f1/f2"

**E25** closed bracket }

# Taclcsv.C Commentary

## F: Print results

**F1.** comment line - states function of next section "print result".

**F2.** if statement that gets executed if flgshft doesn't equal to zero.

**F3.** begin bracket - if statement.

**F4.** function doesn't return a value but prints the tbuf and "???" to the file Fpo.

**F5.** function doesn't return a value but prints an error message with the values of shft and dat to the screen.

**F6.** close bracket - if statement.

**F7.** else statement - gets executed if if statement's condition was false.

**F8.** open bracket - else statement. if statement with condition that f2 be greater than 0.9999.

**F9.** open bracket - if statement

**F10.**if statement with condition that f2 be greater than 99.9

**F11.**function doesn't return a value but prints tbuf and f2 to the file Fpo.

**F12.**else if statement with condition that f2 be greater than 9.99

**F13.**function doesn't return a value but prints tbuf and f2 to the file Fpo.

**F14.**else statement with a function that doesn't return a value but prints tbuf and f2 to the file Fpo.

**F15.**close bracket - if statement

**F16.** else if statement with condition that f2 be lesser than -0.9999.

**F17.**open bracket - else if statement

**F18.**if statement with condition that f2 be lesser than -99.9

**F19.**function doesn't return a value but prints tbuf and f2 to the file Fpo.

**F20.**else if statement with condition that f2 be lesser than -9.99

**F21.**function doesn't return a value but prints tbuf and f2 to the file Fpo.

**F22.**else statement with a function that doesn't return a value but prints tbuf and f2 to the file Fpo.

**F23.**close bracket - else if statement

**F24.**else statement with a function that doesn't return a value but prints the string ctime(&timet) to the file Fpo.

**F25.**close bracket - else statement

**F26.**close bracket - for loop

**F27.**assigns value of tim to timet and changes the format of tim. The value and type of tim is unchanged.

**F28.**if statement with condition that flgskp be not equal to zero. If executed, the fprintf function doesn't return a value but it prints out the string ctime(&timet) to the file Fpo.

**F29.**close bracket for main() function.

**F30.** beginning of subprogram. Initial declaration of program. (s) is the value that is to be sent to and returned bythe subprogram.

**F31.** declaration of the type of the variable s to be character. Also declares a pointer to the value.

**F32.** { open bracket

**F33.** Perror is a function that prints the strings *s onto stderr along with a colon and space f followed by an error message.

**F34.** exit function that stops the program due to an error

**F35.** closed bracket } subprogram

**F36.** Comment line stating the end of the taclcsv.c program.

## Added Features

*The following features have been added to program. A copy of the new version is also included.*

**Page A.** The variables flgmaxmin, flgavg, sum, max, and min are declared and initialized for later usage in the program. These are used to calculate the average, minimum, and maximum of the values. Five variables, year, month, date, yday, and leap have been declared and initialized. These variables are used to calculate the date wanted and convert it into the file name. The variable mday has been removed from declaration and initialization, because it doesn't have any more use. The variable, i, is added which is used for a for statement.

**Page B.** The if statement has been changed to a switch statement, which achieves the same goal, but is easier to read and understand. There is two choices added which allow for calculating the max, min, average, and differential. The two new choices are included in the debug statement, which shows the options chosen. The option of entering the number of days ago has been changed to make the program more practically applicable. The new option asks for the year, month, and date. A section has been added in the else statement that will calculate the day entered in terms of how many days since January 1, and multiply by 1000 which will result in the file name for that day. The operation, timt-mday, has been changed to timt, because it is unnecessary (the variable mday has already been removed).

**Page C.** No changes.

**Page D.** No changes.

**Page E.** Four if statement has been added in the end of the translating section, that calculates the sum of the values, finds the maximum and minimum value, and stores the respective time values (tbuf) using a for statement. The

variables max and min are initialized, the first time, to the value of the data (f2).

**Page F.** An if statement has been added to calculate the average from the sum, and print the result. An else if statement follows to print the max and min, with their corresponding tbuf values, and to calculate and print the differential. The

**Page G.**pre-existing if and else statements have been changed to else if statements and are attached to the newly created if statement. This allows for printing out the result only if requested, otherwise, the value would be printed even if only averaging was requested. In order to print the data in a series of two sets(four columns), and also to be able to import the program into the spreadsheet based program, EXCEL, tabs have been inserted between each value. An if statement has been appended that would shift the cursor to the next line with **every-other-loop.** This allows for the printing of two sets of data on one line.

## New Version

```
A1./* Kelly A. Alvord  & Meenatchi Jagasivamani*/
A2./*Revised version of TACLCSV.C  */
A3./* Mentor :  Amrit Yegneswaran   12/B125 */
A4./*          CEBAF   PHYSICS DIVISION    HALL B  */
A5./*This program formates log files into comma separated variables*/
A6./*The data is read in from the files in a 32-bit floating point format */
A7./*  bit 32                    sign for mantissa
A8.    bit 31                    not used
A9.    bit 30                    sign for exponent
A10.   bit 29 - 24               exponent
A11.   bit 23 - 1                mantissa*/

A12.#include <stdio.h>
A13.#include <time.h>
A14.char buffer[64], dummy[16];
A15.main (ac,av)
A16.int ac;
A17.char *av[];
A18.{
A19.FILE *Fpi, *Fpo;
A20.int m,n, i, tmp, bit, shft, year, month, date, yday;
A21.int flgskp, flgdbg, flgmin, flgdat, flgnam, flgtim, flgmaxmin, flgavg, flgshft,
    step;
A22.long tim, timt, lstd=-9999999L, dat, tstart;
A23.float sum, max, min, f1,f2, leap;
A24.char byte[8], tnam[32], inam[16], onam[16], tbuf[16], tmax[16], tmin[16],
                              *dptr, *sptr, *tptr;

A25.struct tm  *tmptr;
A26.time_t timet;
A27.flgskp=flgdbg=flgmin=flgdat=flgnam=flgavg=flgtim= flgmaxmin=0;
A28.Fpo=stdout;
A29.time(&timet);
A30.tmptr=localtime(&timet);
A31.year=month=date=yday=leap=0;
A32.timt = 1000L * tmptr->tm_year + tmptr->tm_yday + 1L;
```

```
B1.   /* Interpret command line: */
B2.   if (ac==1) error ("taclcsv -h for help");
B3.   for (step=1; *av[step] == '-'; ++step);
B4.   {     switch(*(sptr=(av[step]+1))
B5.         {     case 's':  flgskp =1;
B6.               case 'd': flgdbg=1;
B7.               case 'o': flgnam=1;
B8.               case 'm': flgmin=1;
B9.               case 't': flgtim=1;
B10.              case 'x': flgmaxmin=1;
B11.              case 'a': flgavg=1;
B12.              case 'h':
B13.              {     fprintf (stderr,"\t Usage: taclcsv [-s] [-m] [-o] [-d] [-t] [-x]
                              [-a] var [/yyddd] [outfile]; options: \n);
B14.                    fprintf(stderr,"-s, skip lines with same data as
                              previous line \n");
B15.                    fprintf(stderr,"-o, make up an output file name \n");
B16.                    fprintf(stderr,"-m, time in minutes from start of file \n");
B17.                    fprintf(stderr,"-x, prints minimum, maximum, and
                              differential value with its respective time\n");
B18.                    fprintf(stderr,"-a", prints the average of values\");
B19.                    fprintf(stderr,"-t, time of day in hh:mm:ss\n");
B20.                    fprintf(stderr,"(-m -t gives hh:mm from start of file)\n");
B21.                    fprintf(stderr,"-year, month, date of day wanted\n");
B22.                    fprintf(stderr,"-d, print debug info\n");
B23.                    fprintf(stderr,"Today is day %5ld\n", timt);
B24.                    exit(0);
B25.              }
B26.              default:
B27.              {     if (sscanf(sptr,"%d, &tmp)!=1)
B28.                          fprintf(stderr,"unknown option: %c\n:,*sptr);
B29.                    else
B30.                    {
B31.                          sscanf(sptr, "%d %d %d", &year, &month, &date);
B32.                          switch(month)
B33.                          {
B34.                                case  1: yday=(((month-1)*30)+date);
B35.                                case  2: yday=(((month-1)*30)+date) +1;
B36.                                case  3: yday=(((month-1)*30)+date)-1;
B37.                                case  4: yday=(((month-1)*30)+date);
B38.                                case  5: yday=(((month-1)*30)+date);
B39.                                case  6: yday=(((month-1)*30)+date)+1;
B40.                                case  7: yday=(((month-1)*30)+date)+1;
```

```
B41.                                    case  8: yday=(((month-1)*30)+date)+2;
B42.                                    case  9: yday=(((month-1)*30)+date)+3;
B43.                                    case 10: yday=(((month-1)*30)+date)+3;
B44.                                    case 11: yday=(((month-1)*30)+date)+4;
B45.                                    case 12: yday=(((month-1)*30)+date)+4;
B46.                                }
B47.                                leap=year%4;
B48.                                if (leap==0) yday++;
B49.                                timt = 1000L *year + yday;
B50.                            }
B51.                        }
B52.                }
B53.        }
B54.        if (flgdbg) fprintf (stderr,"options %d%d%d%d%d%d, step=%d\n",flgskp,
                                        flgavg, flgmaxmin, flgdbg,flgnam, flgmin,step);
```

## Taclcsv.c C: Open Input/Output File

```
C1./* Open input file: */
C2.    for (sptr = av[step]; *sptr != '\0'; ++sptr)
C3.    if ((*sptr == '/') || (*sptr == '\\'))
C4.    {
C5.            sprintf(inam, "%s", av[step]);
C6.            break;
C7.    }
C8.    else  sprintf(inam, "%s/%5ld", av[step], timt);
C9.    if ((Fpi = fopen(inam, "rb")) == NULL)  error(inam);
C10./* Open output file: */
C11.   if (flgnam != 0)        /* make an output file name var.ddd */
C12.   {
C13.           sprintf(tnam, "%s", av[step]);
C14.           for (tptr = tnam; *tptr != '\0'; ++tptr) ;
C15.           dptr = tptr - 3;          /* point at day */
C16.           for (tptr -= 3; tptr > tnam; --tptr)  if (*tptr == '/')  break;
C17.           /* back to end of directory */
C18.           for (*tptr = '\0'; tptr >= tnam; –tptr)  if (*tptr == '/')  break;       /* to
                                                                    start of directory */
C19.           ++tptr;/* point at variable name */
C20.           if ((int)(dptr - tptr) > 11)
C21.                   tptr = dptr - 11;        /* else too long for DOS */
C22.           sprintf(onam, "%s.%s", tptr, dptr);
C23.           if (flgdbg != 0)  fprintf(stderr, "%s, %s, making %s\n", tptr, dptr,
                                                                    onam);
C24.           if ((Fpo = fopen(onam, "wt")) == NULL)  error(onam);
C25.   }
C26.   else if (ac> step+1)
C27.   if ((Fpo=fopen(av[step+1], “wt”))==“wt”))==NULL) error (av[step+1]);
```

## Taclcsv.c D: Read Data in 8 byte blocks

```
D1./*Read data in 8 byte blocks, to end of file.*/
D2.for (n=0;;++n)
D3.{    if ((tmp=getc(Fpi))==EOF) break;
D4.byte[0]=(char)tmp;
D5.for (m = 1 ; m < 8; ++m)  byte[m] = getc(Fpi);        /* 8 bytes */
D6.tim = 0xff & (long)byte[0];
D7.tim = (tim << 8) + (0xff & (long)byte[1]);
D8.tim = (tim << 8) + (0xff & (long)byte[2]);
D9.tim = (tim << 8) + (0xff & (long)byte[3]);
D10.timet = (time_t)tim;
D11.if (flgmin)
D12.{           if (n == 0)
D13.            {       tstart = tim;   /* first line: time in seconds from 1970 */
D14.                    (void)fprintf(Fpo, "\'%s", ctime(&timet));
D15.            }                       /* first line header: 'time */
D16.            tim -= tstart;  /* time in seconds from start of data */
D17.            tim /= 60;
D18.}
D19.else  if (n == 0)/* first line header: 'time */
D20.(void)fprintf(Fpo, "\'%s", ctime(&timet));
D21.if (flgtim)
D22.{           if (flgmin)  sprintf(tbuf, "%2d:%2d", tim/60, tim%60);
D23.            else
D24.            {       `       timet = (time_t)tim;
D25.                            (void)sprintf(buffer, "%s", ctime(&timet));
D26.                            sscanf(buffer, "%s%s%s%s", dummy,  dummy,
                                                    dummy,  tbuf);
D27.            }
D28.}
D29.else  sprintf(tbuf, "%ld", tim);
D30.dat = 0xff & (long)byte[4];
D31.dat = (dat << 8) + (0xff & (long)byte[5]);
D32.dat = (dat << 8) + (0xff & (long)byte[6]);
D33.dat = (dat << 8) + (0xff & (long)byte[7]);
```

## Taclcsv.c E page: Translate Binary Data

```
E1./* Translate 32-bit FP: */
E2.bit = (0xff & (int)byte[4]) >> 6;  /* top 2 bits */
E3.shft = (int)((0x3f800000L & dat) >> 23);       /* next 7 bits = exponent */
E4.if ((shft & 0x40) == 0x40)  shft |= 0xffffff80;   /* extend sign */
E5.if (dat == 0L)  f1 = 0.0;  /* special case, I think */
E6.else  f1 = (float)(0x800000L | (0x7fffffL & dat));       /* add top bit */
E7.if ((2 & bit) == 2)  f1 = -f1;       /* sign bit */
E8.f2 = 4194304.0;          /* = (2**22) */
E9.flgshft = 0;
E10.if (shft < 0)
E11.{           if (shft < -15)  flgshft = 1;
E12.            else  f2 *= (float)(1 << (-shft));
E13.}
E14.else
E15.{           if (shft > 15)  flgshft = 1;
E16.            else  f2 /= (float)(1 << shft);
E17.}
E18.if ((flgskp != 0) && (dat == lstd))  continue;
E19.lstd = dat;
E20.if (f2 == 0.0)  f2 = f1;
E21.else  f2 = f1/f2;
E22.if (flgdbg != 0)
E23.{           (void)fprintf(stderr, "%4d:%5ld, %8lx; ", n, tbuf, dat);
E24.            (void)fprintf(stderr, "bits=%1x%1x, shft=%3d, num=0x%lx=%.0f,
                        %f\n", bit>>1, 1&bit, shft, (long)f1, f1, (f2 == 0.0) ? f1 : f1/f2);
E25.}
E26.If (flgavg) sum = sum +f2;
E27.if (n==0) min = max = f2;
E28.if (f2>max)
E29.{           max = f2;
E30.            for( i = 0; i==16; ++i)
E31.                    tmax[i] = tbuf[i];
E32.}
E33.if (f2<min)
E34.{           min = f2;
E35.            for (i=0; i==16; ++i)
E36.                    tmin[i] = tbuf[i];
E37.}
```

## Taclcsv.c F page: Print Result

```
F1./* print result: */
F2.    if (flgavg)
F3.            (void)fprintf(Fpo, "The average of the data is %f\n", sum/(n+1));
F4.    else if (flgmaxmin)
F5.    {
F6.            (void)fprintf(Fpo, "The maximum value, %f, occured at time %s",
                        max, tmax);
F7.            (void)fprintf(Fpo, "The minimum value, %f, occured at time %s",
                        min, tmin);
F8.            (void)fprintf(Fpo, "The differential is %f", max-min);
F9.    }
F10.   else if (flgshft)
F11.   {
F12.           (void)fprintf(Fpo, "%s, ???\n", tbuf);
F13.           (void)fprintf(stderr, "shft = %d, dat = %8lx\n", shft, dat);
F14.   }
F15.   else
F16.   {              if (f2 > 0.9999)
F17.                   {
F18.                           if (f2 > 99.9)
F19.                           (void)fprintf(Fpo, "%s, \t%.3f", tbuf, f2);
F20.                           else if (f2 > 9.99)
F21.                           (void)fprintf(Fpo, "%s,\t %.4f", tbuf, f2);
F22.                           else  (void)fprintf(Fpo, "%s, \t%.5f", tbuf, f2);
F23.                   }
F24.                   else if (f2 < -0.9999)
F25.                   {
F26.                           if (f2 < -99.9)
F27.                           (void)fprintf(Fpo, "%s, \t%.3f", tbuf, f2);
F28.                           else if (f2 < -9.99)
F29.                           (void)fprintf(Fpo, "%s,\t %.4f", tbuf, f2);
F30.                           else  (void)fprintf(Fpo, "%s,\t %.5f", tbuf, f2);
F31.                   }
F32.                   else  (void)fprintf(Fpo, "%s,\t %.6f", tbuf, f2);
F33.   }
F34.If ((n%2) !=0) (void)fprintf(Fpo, "\n");   /*if odd values ptr goes to next line*/
F35.else (void) fprintf(Fpo, "\t");          /*tabs for even n values to make columns*/
F36.
F37.
F38.}
```

```
F39.timet = (time_t)tim;
F40.if (flgskp)  (void)fprintf(Fpo, "\'%s", ctime(&timet));
F41.
F42.
F43.}
F44.error(s)
F45.char       *s;
F46.{
F47.perror(s);
F48.exit(1);
F49.}
F50./* end taclcsv.c  */
```

# Conclusions

Currently, the program is in the process of being compiled. These modifications have been made to produce the following changes upon execution of the program:

1. Calculate and print the average of the set of data.
2. Determine and print the maximum and minimum value with their respective time periods.
3. Evaluate the differential between the maximum and the minimum value.
4. Only print the requested data, instead of the default listing of all of the data.
5. Output the data in columns for better readability and for transfer into the EXCEL program.
6. Allow the user to input the wanted day by calendar date, instead of a difference between the current date and the date wanted.
7. Revision of the source code for better readability and understanding through the addition of commands, elimination of unused portions in the program, and the replacement of functions.