The Boston Computer Society TI-99/ 4A User Group Meeting Newsletter January 1989

Edited by J. Peter Hoddie

The February Meeting

The February Meeting of the Boston Computer Society's TI-99/4A User Group will take place on Wednesday February 15 at 7:30 PM. As always, we'll be meeting at the Massachusetts College of Art on Huntington Avenue in Boston. We will attempt to start the meeting closer to 7:30 than 8:00 but this has become a challenge in recent months. For cancellation information due to the weather call the office at 367-8080 or me at 375-6003.

The meeting topic has been determined. The problem is that I'm getting old and I can't remember what it is. It is written down on the cover of the newsletter that you'll be getting in the mail in the next week or so. So whatever is says there, that's what we'll be doing - for at least part of the meeting. One thing we'll be doing for certain, is the public debut of my newest program. At the December meeting I took lots of suggestions from those in attendence as to what you would like to see. Corson's request is included as an article in this issue. Another suggestion that came out of that meeting is currently under development. It is an extremely exciting program, that many of you will certainly be interested in. A significant portion is already operational, and by next month I fully intend to have a working version (though probably not finished) to show.

The Fourth Annual New England TI Fayuh is still scheduled for April 1 from 10 to 5 in Woburn Massachusetts at the Ramada in that is off I95. This week mailings to many user groups and dealers are going out. If a user group or dealers wants information on exhibiting at the show, drop us a note and we'll get it right out to you. Everyone is welcome. Watch this newsletter for more details.

Fest-West is being held the weekend following our February meeting in San Diego. We are making arrangements so that the BCS can be represented. The last two shows (Los Angeles and Las Vegas) I handled the BCS stuff off the Genial Computerware table. It looks like this year the BCS stuff (mostly the software library) will be handled by Mike Wright, whose assistance in Chicago the past two years has been invaluable

Intro to the UCSD P-System

By Ron Williams

This Month I will show you a few new statements of the Pascal programming language. The first statement is the Case statement, it is very close to the On-Goto or On-Gosub statement in Basic.

The Case statement is shown below:

```
CASE number of
   1 : write('one');
   2 : write('two');
   3 : write('three');
   4 : write('four');
   5 : write('five');
   6 : write('six');
   7 : write('seven');
   8 : write('eight');
   9 : write('nine');
  10 : write('ten');
end;
```

The identifer number is declared as as an integer type number. If the value of the identifer is any number 1-10 the Case statement will cause the statement following the Colon : to be executed for example if the value of number was 3 the program will print out the string "three". In UCSD Pascal this statement is ok to be used in a program but in standard Pascal and possibly other Pascal languages this statement may not work if the value of the number is another value and not 1-10 but in UCSD Pascal if the value is not found the statement will simply not print out anything and there will be no errors. This could become a problem if you are trying to convert this statement to be used with another Pascal language. The If statement is close to the same statement in Basic but it is much simpler to use.

An example is as follows:

The Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

```
If number = 1
  then
     write('one')
  else
     write('number not equal to one');
```

This example shows that the IF statement can also include an else clause if needed also the statement may use begin and end statements if a condition is met and there are a lot of things to be done. The If statement can also call a Procedure or a Function within a program but I will go into more detail about Procedures and Functions later. The If statement can also be a lot more complex than this simple example, other boolean operators may be used like OR, AND, NOT to make this statement one of the most useful in Pascal the statement may look like this:

```
If (number = 1) or (number = 3)
  then
    writeln('stop');
```

The If statement will be used a lot but keep in mind that in some cases the CASE statement may work a lot better like in the first example I showed you if you were to write this using If statements it would be very long and drawn out so use the statement that will work with the least effort. Well so long until next month.

Random Ramblings

By J. Peter Hoddie

This newsletter is produced using a Macintosh computer and various pieces of software. The process is something like this. Lots of people write their articles on their TI's. They upload them to the TI99 BBS and then the articles are downloaded to the Mac where they are loaded into Microsoft Word and cleaned up and formatted a bit. These files are then pasted into the newsletter template in Aldus Pagemaker and then printed on the LaserWriter. The whole process is pretty straight forward but getting to be a bit on the tedious side. The part that takes the most time is cleaning up the files from the TI so that they look OK on in the newsletter. I have developed some techniques which can speed this process up considerably. I will outline here the rules that should be followed when writing newsletter articles to make life as painless as possible for those of us charged with the duty of putting your newsletter together.

1. When entering your article make sure you use a left margin of zero. It is a major hassle to go through and pull all of those leading spaces out.

2. The newsletter is formatted with a blank line between each paragraph. If you don't put it in, I have to and sometimes I miss one. Also section heading don't usually get a blank line after them, just one above.

3. If you are just writing a straight article, i.e. no tables or programs, then just make sure that there is a carriage return (that little c/r symbol) at the end of each paragraph. Or at least a blank line between each paragraph. These are the only safe approaches to dividing paragraphs. Anything else may cause problems.

4. If you are mixing text and program/tables it gets slightly interesting. The Mac reformats things on the fly which means that each line of code must be a separate paragraph. However it is a holy pain to put a carriage return at the end of every line of code. Therefore I worked out this odd system. I specified two new formatter dot commands that get interpreted on the Mac end of things. They are .MT (for TEXT MODE) and .MP (for PROGRAM MODE). We assume you start in text mode. When you get to the program portion of your article include a line with .MP on it. When the program is over, include a .MT, and that's it. You can have as many MP and MT commands as you like in the article. I realize that this is a bit of a hassle, but it is nothing compared to what I sometimes have to go through to accomplish the same thing manually.

5. Don't bother with any other formatting commands since they all get tossed away by the Mac.

6. Don't archive your article. If the article is archived, I have to download it to my 9640, unarchive it, reupload to the BBS, and then download it onto the Mac. This is counter-productive.

Hopefully one day this will be a bit easier. This isn't really all that complicated. If you have any questions, please let me know. Your articles are greatly appreciated.

By the way, if you'd like to write for the newsletter and don't have a modem you can provide us with a disk and we'll transfer the file to the bulletin board for you. If you want to write the article by hand, I'll even type it in for you. Anything to encourage you to contribute to this newsletter!

c.COLUMN

by Donald L.Mahler

In looking over the last few c.columns, I find that I have been devoting almost all of the space to c99 on the 9640, which probably finds a rather limited audience. So this month we shall go back to working with things that work on either machine. As you rember, *name is a "pointer" and "points" to value at address "name". *(name+1) will be the value at the next address; since an integer uses two bytes, it will actually be TWO higher! Take a look at this program:

```
/* pointer arith   pal_c*/
extern printf();
main()
(int *number,i;   *number=2;
for(i=1;i<=10;i++)
  { *(number+i)=2*(*(number+i-1));}
for(i=0;i<=10;i++)
{printf("The %d th term is %d
;",i+1,*(number+i));
printf("the address is %u \n\n",(number+i));
}}
```

The value at address "number" is set at 2, and higher powers of 2 are placed at subsequent even addresses.

We can use pointers with arrays. When we remember that the NAME of an array is a pointer to the first element of the array, the following program seems obvious:

```
/* arrays and pointers p124_c */
extern printf();
main()
{int item[5],i,*point;
for(i=0;i<=4;++i)
     item[i] = i * i;
point = &item[0];  /*address of item[0] */
printf("Output array two ways:");
for(i=0;i<=4;++i)
  printf("\n%d  %d",item[i],*(point+i));
printf("\n\nOutput elements 2, 3, and 4 of
array");
point = &item[2];
for(i=0;i<=2;++i)
   printf("\n%d  %d",*(point+i),item[2+i]);
}
```

In the final program, we use alloc to reserve space. Note that we assign the size of the array from WITHIN the program:

```
/* ARRAY OF POINTERS */
/* USING ALLOC */
/* MOD FROM CHIRLIAN BY DLM*/
#define MAX 10
#include dsk.atoi
extern printf();
extern malloc();
main()
{int *grade[MAX],i,j,n,nt,temp;
char buf[80];
printf("Enter num students:");
if((n=atoi(gets(buf)))>MAX)
     {printf("Num too large; prog
terminated");
      exit(); }

printf("\n Enter num tests");
nt=atoi(gets(buf));
temp=(nt+1)*2;
 for(i=1;i<=n;++i)
     grade[i-1] = alloc(temp);
printf("Enter indicated grade:");
for(i=1;i<=n;i++)

 {printf("\n student number %3d",i);
   for(j=1;j<=nt;++j)

     {printf("\n%3d:    ",j);
      *(grade[i-1]+j-1)=atoi(gets(buf));  } }

for(i=1;i<=n;++i)
{  *(grade[i-1]+nt)=0;
for (j=0;j<=nt-1;j++)
   *(grade[i-1]+nt)=*(grade[i-
1]+nt)+*(grade[i-1]+j);  }
printf("\n Student number        Average");
for(i=1;i<=n;++i)
{temp=*(grade[i-1]+nt);
*(grade[i-1]+nt)=*(grade[i-1]+nt)/nt;
if(2*(temp%nt)>=nt) ++*(grade[i-1]+nt);
printf("\n    %3d
%3d",i,*(grade[i-1]+nt));
} }

/* alloc()  */
#define ALLOCBYTES 256 /* size of available
space in bytes (should be even) */
int  allocfirst = 0;       /* is free
position set */
char allocbuf[ALLOCBYTES]; /* storage for
alloc    */
char *allocp;              /* next free
position   */

alloc(n)
    int n;
 { if(!allocfirst)
      { ++allocfirst;
        allocp = allocbuf; }
    if(is_odd(n))
        ++n; /* make sure it's even */
    if(allocp + n <= allocbuf + ALLOCBYTES)
      { allocp = allocp + n;
        return(allocp - n); }
      else
        return(0); }
free(p)
    char *p;
  { if(p >= allocbuf & allocbuf + ALLOCBYTES)
        allocp = p;  }
is_odd(n)
    int n;
  { return(n % 2); }
```

We can also use arrays with functions, as the following program shows:

```
/*ARRAYS AND FUNCTIONS */
/*modified from Chirlian by dlm */
extern printf();
main()
{int list1[10],list2[10],i,sum;
/*build the first array */
for(i=0;i<=9;i++)
     able[i] = 1 +(i*2);
sum=fixer(list1,list2);
printf("Sum = %5d",sum);  /* original array
*/
/* print out new array   */
for(i=0;i<=9;i++)
     printf("\n%5d",list2[i]);}

fixer(old,new)
int old[],new[];
{int total,i;
total=0;
for(i=0;i<=9;++i)
{total=total+old[i]; /* add elements */
/*compute new array-each element is */
/* three times size of orig element */
 new[i]=3 * old[i]; }
return total; }
```

Notice that the function has changed the value of the variables!

Column Manipulator

By J. Peter Hoddie

Written on January 5, 1988
at the request of Wm. Corson Wyman

Column Manipulator is a little Extended BASIC program designed to allow a series of text manipulations to be executed on an entire text file. The program only operates in batch mode. An input file must be created with TI-Writer or MY-Word and run through the program at the prompt for "Procedure file." The format for commands is very rigid (no fancy parsing, I wrote this in 60 minutes). The procedure file has a header portion which can be used to define the input and output files as well as the fields. Each field is defined by a starting position (starting from 1) and a length (better be greater than zero!). The format for the procedure file is the command as the first thing on the line, in capital letters. Parameters for the command then follow, each separated by one space. There may be no spaces to the left of the command and trailing spaces are ignored. The general format is

```
COMMAND parameter1 parameter2
```

Defining Fields

To define your fields, use the field command in the procedure file. There may be up to 40 fields. The general format is

```
FIELD field_number field_start field_length
```

The field number is from 1 to 40. The field_start is the starting column number (starting at 1) of the field. The field_length is the number of characters in the field. You may redefine any field, but only the last field definition is used.

The Rest of the Procedure

This utility provides 5 commands for manipulating each line of text. These commands must appear between BEGIN and END commands in your procedure file. You should set up your fields first. Thus the general format of the procedure file is

```
FIELD 1 a b
FIELD 2 c d
FIELD 3 e f
BEGIN
END
```

The set of commands between the BEGIN and END will be executed for every line of the text in the input file with the results being sent to the output file.

Defining Input and Output Files

You may specify the input and output files using the INPUT and OUTPUT commands in your procedure file. These must occur before the BEGIN statement. The formats for each are the same

```
INPUT filename
OUTPUT filename
```

If these commands are not present in your procedure file you will be prompted for them.

The Text Manipulation Commands

```
DELETE field_number
```

Deletes the field specified by field_number from the line.

```
INSERT field_number count
```

Inserts count number of blank spaces before the field_number specified.

```
MOVE field_number1 field_number2
```

Moves field_number1 from where it is currently located to before field_number2.

```
COPY field_number1 field_number2
```

Puts a copy field_number1 before field_number2. The original field_number1 is not removed as with the MOVE command.

```
FILL field_number1 fill_text
```

Takes fill_text (which may contain embedded spaces) and puts it into field_number1. If the text is longer than the field, the text is truncated. If the text is shorter than the field, it is padded with spaces.

A Warning

If you use commands to move a field or delete a field or just about any of the text manipulation commands (except FILL), the fields may not be where you expect them. The field definitions are not changed when fields are moved, so you have to keep track of where things went. Be aware of this fact since or you may be in for some unpleasant surprises. Other approaches to writing this sort of program can remove that problem but have other problems.

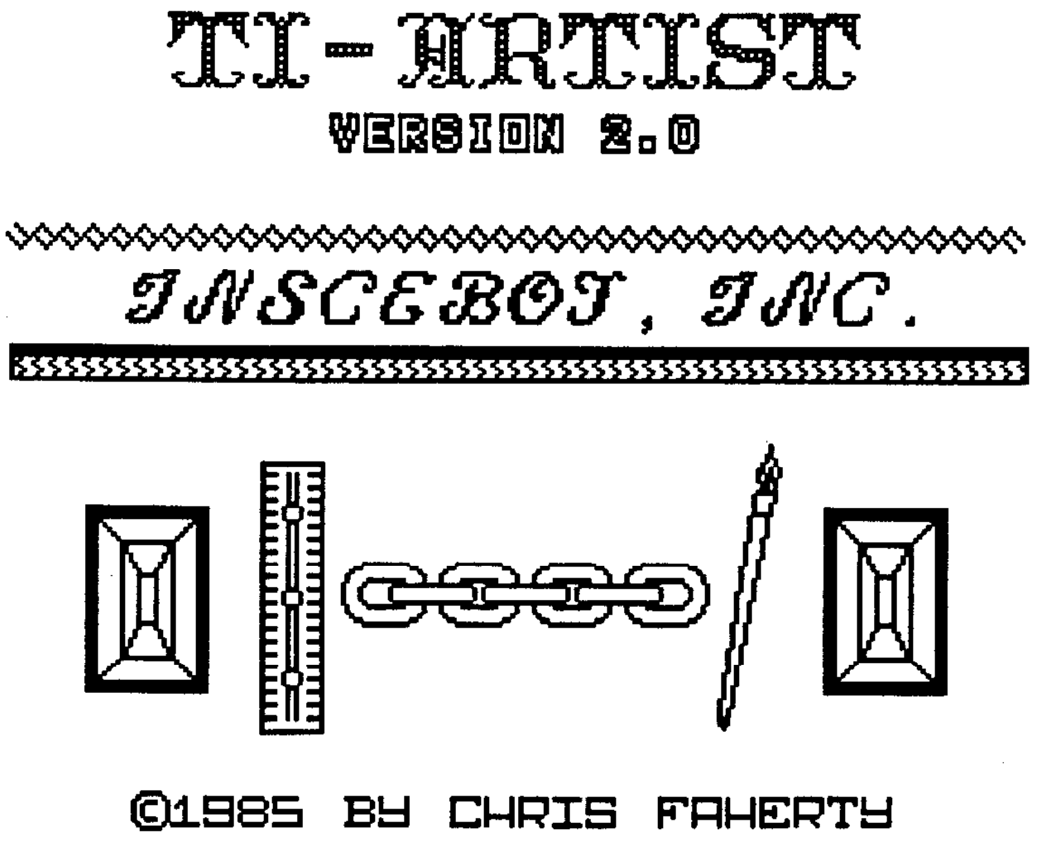
The Program

```
5 CALL CLEAR :: PRINT "COLUMN MANIPULATIONS": :"BY
J. PETER HODDIE":"AT THE REQUEST OF":"Wm. CORSON
WYMAN": :
10 DIM F(40,2),PROC$(100)
50 COM$="DELETEINSERTMOVE  COPY  FILL  FIELD "
60 GOSUB 2000
61 PRINT "PARSING STARTUP DATA...": :
65 FOR LOOP=0 TO P_START-1 :: C$=PROC$(LOOP)::
GOSUB 100 :: NEXT LOOP
```

```
66 PRINT "PROCESSING FILE...": :
70 OPEN #1:I$,INPUT :: OPEN #2:O$,OUTPUT
80 INPUT #1:L$ :: FOR LOOP=P_START TO P_END ::
C$=PROC$(LOOP):: GOSUB 100 :: NEXT LOOP :: PRINT
#2:L$
85 IF EOF(1)THEN 90 ELSE 80
90 CLOSE #1 :: CLOSE #2 :: PRINT "FINISHED..." ::
END
100 ! COMMAND PROCESSOR
105 Z=POS(C$," ",1):: IF Z=0 THEN RETURN
110 Z$=SEG$(C$,1,Z-1):: Z$=SEG$(Z$&"      ",1,6)::
C$=SEG$(C$,Z+1,255)
120 Z=POS(COM$,Z$,1):: IF Z=0 THEN RETURN ELSE ON
(Z+5)/6 GOTO 600,700,800,900,1
000,1100,1200,1300,1400
400 PF=POS(C$," ",1):: IF PF=0 AND LEN(C$)=0 THEN
PRINT "ERROR IN INPUT" :: STOP
405 IF PF=0 THEN PF=LEN(C$)+1
410 Z$=SEG$(C$,1,PF):: C$=SEG$(C$,PF+1,255)
420 RETURN
600 ! DELETE FIELD
610 GOSUB 400 :: Z=VAL(Z$)
620 L$=SEG$(L$,1,F(Z,1)-
1)&SEG$(L$,F(Z,1)+F(Z,2),255)
630 RETURN
700 ! INSERT FIELD
710 GOSUB 400 :: Z=VAL(Z$):: GOSUB 400 ::
Z1=VAL(Z$)
720 L$=SEG$(L$,1,F(Z,1)-1)&RPT$("
",Z1)&SEG$(L$,F(Z,1),255)
730 RETURN
800 ! MOVE FIELD BEFORE FIELD
810 GOSUB 400 :: Z=VAL(Z$):: GOSUB 400 ::
Z1=VAL(Z$)
820 F$=SEG$(L$,F(Z,1),F(Z,2))! COPY ORIGINAL FIELD
830 L$=SEG$(L$,1,F(Z,1)-
1)&SEG$(L$,F(Z,1)+F(Z,2),255)! DELETE ORIGINAL
FIELD
840 L$=SEG$(L$,1,F(Z1,1)-
1)&F$&SEG$(L$,F(Z1,1),255)! MOVE THE FIELD...
850 RETURN
900 ! COPY FIELD BEFORE FIELD
910 GOSUB 400 :: Z=VAL(Z$):: GOSUB 400 ::
Z1=VAL(Z$)
920 L$=SEG$(L$,1,F(Z1,1)-
1)&SEG$(L$,F(Z,1),F(Z,2))&SEG$(L$,F(Z1,1),255)
930 RETURN
1000 ! FILL FIELD WITH...
1010 GOSUB 400 :: Z=VAL(Z$)
1020 IF LEN(C$)<F(Z,2)THEN C$=C$&RPT$(" ",F(Z,2)-
LEN(C$))
1025 IF LEN(C$)>F(Z,2)THEN C$=SEG$(C$,1,F(Z,2))
1050 L$=SEG$(L$,1,F(Z,1)-
1)&C$&SEG$(L$,F(Z,1)+F(Z,2),255)
1060 RETURN
1100 ! DEFINE FIELD NUMBER, START,LENGTH
1110 GOSUB 400 :: Z=VAL(Z$):: GOSUB 400 ::
F(Z,1)=VAL(Z$):: GOSUB 400 :: F(Z,2)=VAL(Z$)::
RETURN
2000 ! LOAD PROCEDURE FILE
2010 INPUT "NAME OF PROCEDURE FILE:      ":Z$
2015 OPEN #1:Z$,INPUT
2016 PRINT :"LOADING PROCEDURE FILE": :
2020 Z=0
2021 P_START=0
2025 INPUT #1:Z$
2030 IF SEG$(Z$,1,5)="BEGIN" THEN P_START=Z ::
GOTO 2025
2035 IF SEG$(Z$,1,3)="END" THEN 2050
2036 IF SEG$(Z$,1,5)="INPUT" THEN
I$=SEG$(Z$,POS(Z$," ",1)+1,255)
2037 IF SEG$(Z$,1,6)="OUTPUT" THEN
O$=SEG$(Z$,POS(Z$," ",1)+1,255)
2040 PROC$(Z)=Z$
2042 Z=Z+1
2045 GOTO 2025
2050 CLOSE #1 :: P_END=Z-1
2070 IF I$="" THEN INPUT "INPUT FILE: ":I$ :: IF
I$="" THEN 2070
2080 IF O$="" THEN INPUT "OUTPUT FILE: ":O$ :: IF
O$="" THEN 2080
2090 RETURN
```

Print TI-Artist Pictures on a Laser Printer!!!!
By J. Peter Hoddie

As noted earlier in this newsletter, this newsletter is produced using an Apple Macintosh and a LaserWriter printer. What many of you may not know is that the LaserWriter is the real magic in the combination. It provides for 300 dots per inch of resolution on output. It also contains a magical language called PostScript which allows for pages to be created with great detail and accuracy. I always thought it would be nice to get high quality laser printed copies of TI documents and pictures. At a recent computer show I picked up the PostScript Language Reference Manual and wasted some time reading it along with some other material on PostScript (Don Lancaster in Computer Shopper is quite good on the subject). In any case, with much help from Adobe's manuals (the creators of PostScript), I managed to figure out how to print TI-Artist pictures on the LaserWriter. I wrote a program on the TI which creates a PostScript program from the TI-Artist picture. Then I send this program to the LaserWriter using a Macintosh. In theory it could be done directly, but alas there is no TI at the BCS office. You could upload the file to a service bureau and get a printout for something like 50 cents a page though. Anyhow, I worked all this out tonight (the night before the meeting) and though some of you might find it interesting. If the program ever becomes user friendly enough I'll release it somehow. Below is a sample of what is possible (taken from a familiar source)....


TI-ARTIST
VERSION 2.0
INSCEBOT, INC.
©1985 BY CHRIS FAHERTY