[7.45] Display text in multiple fonts, and Pretty-Print expressions

TI Basic programs are usually limited to displaying text in a single font size. However, E.W. (author also of the EQW equation writer) has written a function to display text in 3 font sizes, as well as show expressions in pretty-print on any display screen. Much of the following description is from E.W.'s header in the C source code. You can get this function, called *write()*, from Bhuvanesh Bhatt's web site at http://tiger.towson.edu/~bbhatt1/ti/. The C source code is included.

The zip file includes versions for both the TI-89 (*write.89z*) and the TI-92 Plus (*write.9xz*). Make sure you use the correct version for your calculator. The examples in this tip show screen shots from the TI-92 Plus; screen shots from the TI-89 are similar.

write() supports these features:

- Write a line of text in one of three fonts and four modes.
- Write multiple text lines with the different fonts and modes.
- Write a line of text mixed with pretty-printed expressions.
- Write multiple lines of text mixed with pretty-printed expressions.

write() displays the arguments on the current active screen. It will only write inside the screen client area, that is, it will not write in the toolbar display or the status line area. The active screen may be the home screen, the Program I/O screen, or the Graph screen. Your program is responsible for displaying the screen before calling *write()*. These commands are used to display the different screens:

Home screen:	*DispHome*
Program I/O screen:	*Disp*
Graph screen:	*DispG*

There are three possible syntaxes for *write()*:

Syntax 1:

```
write(x, y, string [,font_size] [,mode])
```

Syntax 2:

```
write(x, y, {string, string, ...} [,font_size] [,mode])
```

Syntax 3:

```
write(x, y, {format_string, "expr",...,format_string,"expr"} [,font_size] [,mode])
```

The first syntax writes a single string. The second syntax writes the strings as multiple lines. The third syntax writes expressions, or lines of text mixed with expressions. The square brackets indicate that the arguments *font_size* and *mode* are optional and need not be included. However, if you include the *mode* argument, you must also specify the *font_size* argument.

x and *y* specify the starting position of the text in pixel coordinates. x is the column coordinate, and y is the row coordinate. The upper left coordinate of the screen has coordinates x = 0 and y = 0. Negative values of *x* and *y* cause the string to be displayed outside of the visible screen area.

string specifies the string to be displayed.

font_size specifies the size of the displayed text:

0 = small

1 = medium (the 'normal' font size; default)
2 = large

mode specifies one of five settings which control the appearance of the displayed text:

0 = white text on black background (over-write mode)
1 = black text on white background (OR mode; default)
2 = black text on white background (XOR mode)
3 = gray text on white background
4 = black text on white background (over-write mode)

In over-write mode, the text and background replace any pixels previously displayed . In OR mode, the text and background pixels are logically OR'd with existing pixels, which means that the pixel at a given location will be 'on' (black) if the previous pixel *or* the text pixel is on. In XOR mode, the resulting pixel is off only if the original background pixel and the text pixel are the same. The over-write mode could be considered an 'opaque' mode, while the OR mode is a 'transparent' mode.

The third syntax and its arguments are described in an example below, after some examples for the first two syntaxes.

Syntax 1 example: Single lines, different font sizes, black and gray text

This example uses the first syntax to display some sample text in three font sizes, with modes 1 and 3.

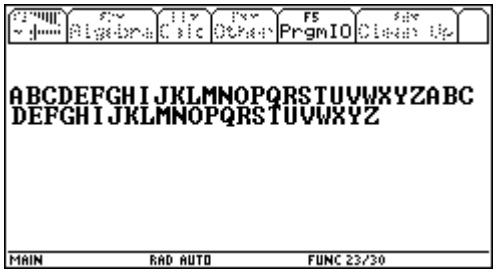
```
clrio                          © Clear Program I/O screen

"ABCDEFGHIJKLMNOPQRSTUVWXYZ"→s

disp                           © Display Program I/O screen
util\write(6Ø,3,s,Ø,1)         © Write lines in mode 1 (black on white)
util\write(4Ø,12,s,1,1)
util\write(1Ø,23,s,2,1)

util\write(6Ø,4Ø,s,Ø,3)        © Write lines in mode 3 (gray)
util\write(4Ø,49,s,1,3)
util\write(1Ø,6Ø,s,2,3)
```

For these examples, the *write()* function is stored in the *util*\ folder, so the function calls use that folder specification.

If the length of the text line exceeds the display width, the text wraps around to the next line, as shown with this example:

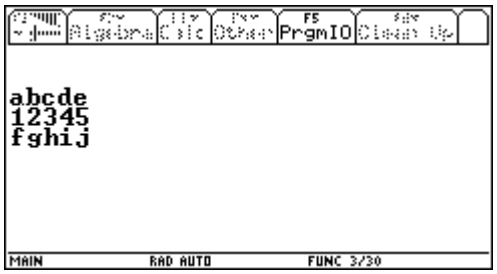
```
clrio
disp
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"→s
```

```
util\write(0,20,s&s,2,1)
```



You may also embed the carriage return character, char(13), in strings. This will force a line break, as shown in this example.

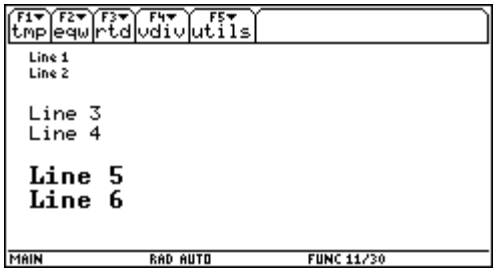
```
clrio
disp
char(13)→cr
"abcde"&cr&"12345"&cr&"fghij"→s
util\write(0,20,s,2,1)
```



Syntax 2 example: Multiple lines

The next example uses the second syntax to display multiple lines with one call to *write()*,

```
clrio
disp
util\write(10,3,{"Line 1","Line 2"},0,1)
util\write(10,30,{"Line 3","Line 4"},1,1)
util\write(10,60,{"Line 5","Line 6"},2,1)
```



Syntax 3 example: Pretty-print expressions

As mentioned above, this syntax is used to display expressions:

```
write(x, y, {format_string, "expr",...,format_string,"expr"} [,font_size] [,mode])
```

format_string is string which contains one or more control characters. The control character char(174) will be replaced by the next expression *expr* in the list. char(174) is the circle-R character "®". This syntax allows the display of multiple expressions on one display line, and mixing text strings and expressions on the same line. In its simplest form, this syntax displays a single expression:

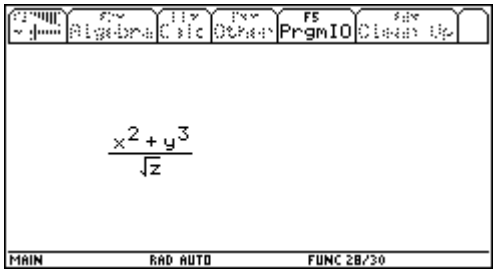
```
char(174)→c
util\write(5Ø,4Ø,{c,"(x^2+y^3)/√(z)"})
```

The control code is saved as the variable *c* for convenience, but this is not necessary. If you are coding directly on the calculator, you can insert the "®" character with [CHAR], [3], [N]. If you are coding in GraphLink, you cannot create the "®" character, and must instead use char(174).

expr is any valid expression, which is passed as a string.

font_size is ignored with this syntax. Expressions are always displayed with font size 1, the standard font. The *mode* argument is used, however. In mode 0 (white text on black background), only the text characters are displayed, and graphic elements such as divisor bars and root symbols may disappear. In mode 3 (gray characters), the expression may be completely illegible.

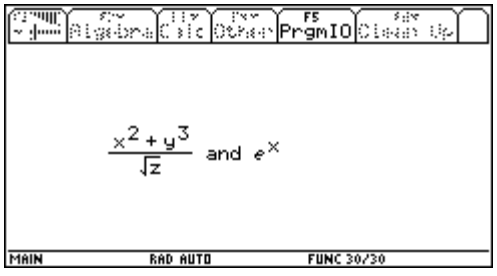
This code segment results in the following display:



Note that the expression is passed as a string, and the default values for *font_size* and *mode* are used.

Up to ten control characters may be used in a single *format_string*. This example shows a *format_string* which contains both control characters and text:

```
char(174)→c
util\write(5Ø,4Ø,{c&" and "&c,"(x^2+y^3)/√(z)","e^(x)"})
```

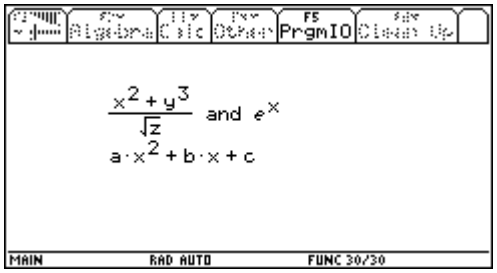


In this case *format_string* is built by concatenating the control characters with the text using the "&" operator, with the expression *c&" and "&c*, which results in the string "® and ®". Since there are two

control characters, the list must include two expressions. In general, the number of expressions must equal the number of control characters.

This syntax for *write()* can also be used to display pretty-print expressions on multiple lines. Each *format_string* starts a new display line, as this example shows:

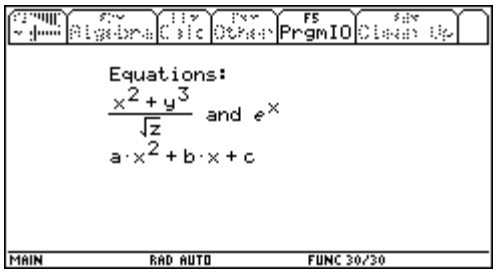
```
char(174)→c
util\write(5Ø,2Ø,{c&" and "&c,"(x^2+y^3)/√(z)","e^(x)",c,"a*x^2+b*x+c"})
```



Note that the second *format_string*, which is just *c*, causes the third expression to be displayed on a new line.

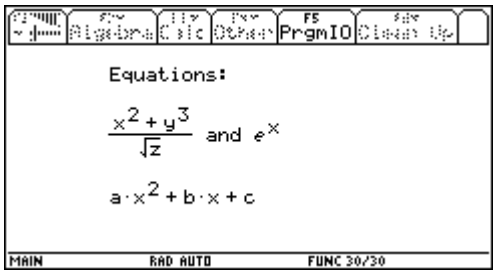
Finally, this syntax can also be used to display text on some lines, and expressions on others. This example shows the text "Equations:" on the line before the expressions:

```
char(174)→c
util\write(5Ø,1Ø,{"Equations:",c&" and "&c,"(x^2+y^3)/√(z)","e^(x)",c,"a*x^2+b*x+c"})
```



You can use empty strings ("") to insert blank lines:

```
char(174)→c
util\write(5Ø,1Ø,{"Equations:","",c&" and "&c,"(x^2+y^3)/√(z)","e^(x)","",c,"a*x^2+b*x+c"})
```



C source code

```
// C Source File
// Created 5/22/2001; 10:07:44 PM
// Author: E.W.


#define OPTIMIZE_ROM_CALLS    // Use ROM Call Optimization

//#define SAVE_SCREEN           // Save/Restore LCD Contents

//#include <tigcclib.h>         // Include All Header Files

//short _ti89;                  // Produce .89Z File
//short _ti92plus;              // Produce .9XZ File


#include <nostub.h>
#include <estack.h>
#include <wingraph.h>
#include <error.h>
#include <args.h>
#include <string.h>
#include <graph.h>

//#include <tigcclib.h>

#define WMAIN ((WINDOW*)(*((long*)(*((long *)(*((long*)0xC8)))))));

int _ti89,_ti92plus;
short GetInt(unsigned char **exp)
{
short value,err;
if((err=estack_to_short(*exp, &value))!=1) {
if(err==0)
ER_throwVar(830);
ER_throwVar(40);
}
*exp=next_expression_index(*exp);
return value;
}
void _main(){
  unsigned char *ap,tag,*ap1,*ap2,*ap3,*expr[10];
  short
f=1,c=-1,x,x1,attr,y,y1,NumArguments,bot,top,width,sindex,index,wi[10],tmax,bmax,i;
  char *s,*si[11],*s1;
  int error;
  ERROR_FRAME err_frame;
  WINDOW *w=WMAIN;

//  clrscr();

  InitArgPtr(ap1);
  NumArguments=remaining_element_count(ap1);
  if(NumArguments<3)
    ER_throwVar (930);
  if(NumArguments>5)
   ER_throwVar (940);
  x=GetInt(&ap1);
  y=GetInt(&ap1);
  ap=ap1;
  ap1=next_expression_index(ap1);
  if(NumArguments>3)
   f=GetInt(&ap1);
  if(NumArguments>4)
   c=GetInt(&ap1);
  if(f>2||c>4)
   ER_throwVar (40);
  if(f<0) f=1;
  tag=*ap;
  ap1=top_estack;
  if(tag==LIST_TAG) ap--;
  w->Flags|=WF_TTY ;
```

```
  if (!(error=ER_catch(err_frame))) {
if(c>=0)
attr=WinAttr(w, c);
if(f!=1)
WinFont (w, f);
while(*ap!=END_TAG) {
if(*ap!=STR_TAG) ER_throwVar (130);
s=*si=GetStrnArg(ap);
index=0;
sindex=1;
tmax=f+2;
bmax=f+4;
x1=x;
while((s=strchr(s,174))!=NULL&&index<=10) {
if(tag!=LIST_TAG) ER_throwVar (90);
if(*ap==END_TAG) ER_throwVar (930);
if(*ap!=STR_TAG) ER_throwVar (130);
*s++=0;
si[sindex++]=s;
if(*ap==STR_TAG) {
push_parse_text(GetStrnArg(ap));
}
Parms2D(expr[index]=Parse2DExpr (top_estack, 0), &wi[index],&bot,&top);
top--;
bot++;
if(bot>bmax) bmax=bot;
if(top>tmax) tmax=top;
index++;
}
if(y>w->Client.xy.y1-w->Client.xy.y0) break;
y=y+tmax+1;
y1=y+bmax+1;
if(y1<0) continue;
for(i=0;i<sindex;i++) {
WinStrXY(w,x1, y-f-2,si[i]);
x1=x1+DrawStrWidth(si[i], f);
if(index--) {
Print2DExpr(expr[i],w,x1,y);
x1+=wi[i];
if(f!=1)
WinFont (w, f);
}
}
if(tag!=LIST_TAG) break;
top_estack=ap1;
y=y1;
}
ER_success();
  }
  w->Flags&=~WF_TTY ;
  if(f!=1)
   WinFont (w, 1);
  if(c>=0)
   WinAttr (w, attr);
  if(error) ER_throwVar (error);

//  ngetchx();
}
```