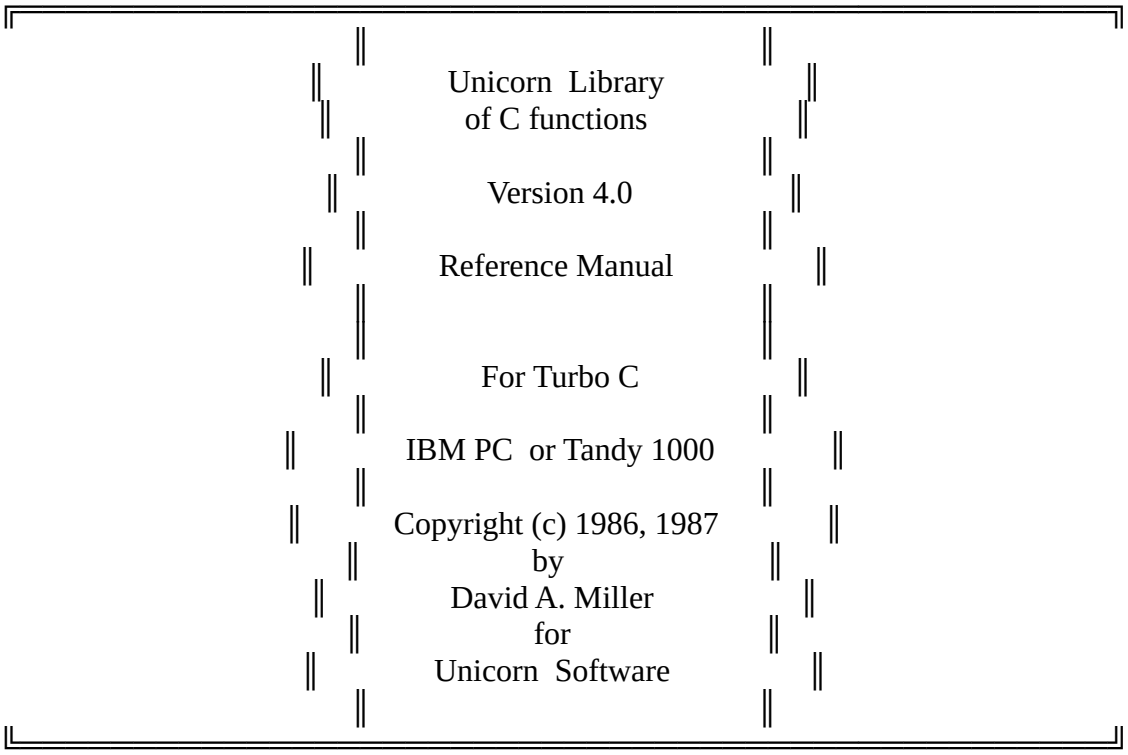

Please note that multiple versions of the library exist. The machine this version is to be used on is listed above. The Tandy 1000 version provides full access to the full 16 color medium resolution mode while the IBM version uses the 4 color medium resolution mode.

Some of the functions are listed for one machine only. Using them on the wrong machine will possibly cause a crash.

If you are using an IBM-PC or wish to use 4 color graphics on a Tandy-1000 you must link the library UCTURBO.LIB. If you wish to use 16 color graphics on a Tandy-1000 be sure to link UCTUR_TD.LIB. These should be specified in your project file.

R E G I S T R A T I O N F O R M

Registration may be accomplished by filling out this page and sending it to the author with a $25.00 donation to cover costs. Help support the idea of freeware. It is a method whereby software authors may make available to the general user various types of software at a minimum cost. It benefits the entire computer community so do your part.

Name: _____

Address: _____

City: _____

State: _____

Zip: _____

Library Version: Version 4.0 P For Turbo C small model

Version Desired (circle 1):

Microsoft C 4.0 Turbo C

I wish a Tandy 1000 version also: yes / no

Computer Type and model: _____

Received this copy of the library from: _____ This will enable us to check distribution and provide updates.

Comments concerning the library and additional functions desired may be listed on the back of the page.

I understand that by submitting this form with the proper donation I will receive a disk with ALL source code for the latest version. I also understand that this allows me to use the library in any commercial or non commercial application I desire without payment of additional fees of any type.

I understand that the source code is proprietary and agree not to give it to third parties although I understand I may give the actual libraries to third parties as long as I include all documentation and copyright notices with or without registering my library and no fee other than a handling fee of no more than $10.00 is charged by a user group or club.

Technical

This library was compiled with the Turbo C compiler and the Microsoft MASM version 4.0. All compilation assumed small memory model.

Getting Started

This library will only work with the small memory model. Using it with other memory models will cause highly unpredictable results and probably crash your machine. You must have a graphics adapter to use the graphics commands and if you wish to use mode 9 or any other Tandy only functions you must have a Tandy 1000. Other memory model libraries will soon be available and will be supplied with an update request ($10.00) or free with registration.

Be sure you specify the appropriate library in your project file. For example if you have a file named test as shown:

```
main()
{
        sm(4);
        atsay(10, 5, 3, 1, "This is a test");
        waitkey();
        sm(3);
}
```

and you wish to link it with the library the your project file should look like the below.

```
test.c
ucturbo.lib
```

Doing so will correctly compile the program.

All parameters are integers unless otherwise noted in this document.

EFor the Tandy version all parameters concerning color may be any of the 16 colors. F For the IBM the color set consists of colors 1, 2, and 3 and a background color.

The Author may be contacted at the below address for registration of your copy of the library. This library may be freely copied and distributed as long as a reasonable fee of no more than $8.00 is charged for the service. Registration with the author will provide you with a license to use the library in programs for sale or distribution without royalty payments along with the library commented source code. The Esource code may NOT be distributed without express written permission of the author F.

Future versions will add both graphics and non-graphics functions. Contemplated at present are animation commands, further string and data handling including screen generation and tree functions along with others. The author may be contacted directly concerning the updates and/or suggested additions.

Support this and all freeware programs. It is a way for authors to make programming available to users at a reasonable cost and thus helps both the user and the end user.

David A. Miller
18559 8th Ave. NE
Seattle, WA 98155
(206) 361-0553
please no
collect calls

---

atsay

This function is similar to the dBASE function of the same name. It allows the user to print a string in color at a specified location on the screen. The background color is ignored in graphics modes.

Call: atsay(row, column, foreground_color, back_color, string);

row = row in which to begin string
column = column in which to begin string foreground_color = color in which to print string back_color = background color for string string = string to be printed

Example:
```
main()
{
    atsay(5, 10, 3, 1, "Testing!");
}
```

This will print the string "Testing!" beginning at row 5 column 10 in color number 3 with a background of color 1.

---

backclr

Sets the background color of the graphics screen.

Call: backclr(color);

color = desired background color

Example:
```
main()
    {
        sm(4);      /*  set the graphics mode */
        backclr(14);  /*  sets background color to yellow */ }
```

---

beepsp

Beeps the internal speaker.

Call: beepsp();

Example:

```
main()
{
    beepsp();
}
```

---

box

Plots a rectangular uotline on the graphics screen. The upper left corner and the lower right corner coordinates are used to define the box.

Call: box(ulc, ulr, lrc, lrr, color);

ulc = upper left corner column
ulr = upper left corner row
lrc = lower right corner column
lrr = lower right corner row
color = color to draw box

Example:
```
main()
    {
        sm(4);    /*  Set graphics mode */
```

```
        /* draw a box from 10, 20 to 100, 140 in color 3 */

        box(10, 20, 100, 140, 3);
}
```


---

box1

Paints a one pixel wide border around a filled box. The fill and border colors may be separately specified.

Call: box1(ulc, ulr, lrc, lrr, colorb, colorf);

ulc = upper left corner column
ulr = upper left corner row
lrc = lower right corner column

lrr = lower right corner row
colorb = border color
colorf = fill color

Example:
```
main()
    {
        sm(4);      /* set a graphics mode */

    /*  make a box from 10, 20 to 100,140 with a border in color 3 and filled with color 2
        */

    box110,20,100,140,3,2);
```


box2

Paints a two pixel wide border around a filled box. The fill and border colors may be separately specified.

Call: box2(ulc, ulr, lrc, lrr, colorb, colorf);

ulc = upper left corner column
ulr = upper left corner row
lrc = lower right corner column
lrr = lower right corner row
colorb = border color
colorf = fill color

Example:
```
main()
    {
        sm(4);      /* set a graphics mode */

    /*  make a box from 10, 20 to 100,140 with a border in color 3 and filled with color 2
        */

    box2(10,20,100,140,3,2);
```

---

boxfill

A function to plot a solid filled rectangle on the graphics screen. It draws using the upper left corner and lower right corner coordinates.

Call: boxfill(ulc, ulr, lrc, lrr, color);

ulc = upper left corner column
ulr = upper left corner row
lrc = lower right corner column
lrr = lower right corner row
color = color to paint rectangle

Example:

```
main()
    {
                sm(4);    /*  set a graphics mode  */

        /*  paint a box from 10, 20 to 100, 140 in color 3 */

        boxfill(10,20,100,140,3);
    }
```

---

burble

Creates a rather strange sound on the internal speaker. Has to be heard to be described.

Call: burble(1/pitch, timer);

Where:

pitch = the basic pitch desired
timer = a value sent to the timer port

Example:

```
main()
{
        burble(100, 300);
}
```

---

button

Returns the joystick button status. The return is held in 4 bits as follows. bit 0 - right button #1 bit 1 - right button #2 bit 2 - left button #1 bit 3 - left button #2

Call:

```
button();
```

Example:

```
main()
{
    int a;
    a = button();
}
```

The button status will be held in a.

---

circle

Plots a circle on the graphics screen.

Call: circle(xc, yc, radius, aspectn, aspectd, color);

xc = column coordinate of the center
yc = row coordinate of the center
aspectn = numerator of the aspect ratio
aspectd = denominator of the aspect ratio
color = color of the circle

Example:

```
main()
    {
        sm(4);   /* set a grphics mode */

        /* draw a circle of radius 20 centered at 50, 60 in color 2                    */

        circle(50, 60, 20, 1, 1, 2);
    }
```

cls

Clears the screen and returns the current video mode in the low byte and the number of screen columns in the high byte of an integer.

Call: cls();

Example:

```
main()
{
    cls();
}
```

CM_INCH

Converts inches to centimeters. This is defined as a macro in UC.H.

Call: CM_INCH(inches);

Example:

```
main()
{
    printf("There are %lf cm in %lf inches\n", CM_INCH(4.5), 4.5); }
```


---

countch

Returns the number of times a character appears in a given string.

Call: countch(str, ch);

str = string to search
ch = character to search for

Example:

```
main()
    {
        /* find the number of i's in the string "This is a test".  should return 2 */ countch("This
            is a test", 'i'); }
```

---

ctone

This function sounds a tone using the IBM mode of sound. Compatible with IBM-PC and close compatibles.

Call: ctone(freq, duration);

freq = the frequency in HZ for the tone
duration = the duration of the tone in milliseconds

Example:

```
main()
    {
        ctone(1000, 500);  /* sound a 500 msec tone of 1000 HZ */ }
```

DEG_RAD

Converts angles in degrees to the equivalent radian value. Both the angle and the return are doubles. Defined as a macro in UC.H.

Call: DEG_RAD(angle);

angle = angle in degrees

Example:
```
main()
    {
        a = DEG_RAD(56.78);
    }
```

---

delay

Delays in multiples of 1 millisecond.

Call: delay(count);

count = the number of milliseconds to delay

Example:
```
main()
    {
        delay(1000); /* pause for 1 second */
    }
```

---

drawrow

This function is used in windowing to draw a row of a single character on the screen.

Call: drawrow(row, column, count, ch);

Where:

row = the row to draw in
col = the starting column
count = the numbe of characters to draw
ch = the character to draw

Example:

```
main()
{
    drawrow(10, 10, 50, "-");
}
```

Draws a row of 50 dashes in row 10 starting in column 10.

---

dwarn2

This function issues a warning message in the version of DOS in current use is not at least version 2.00

Call: dwarn2();

Example:

```
main()
{
    dwarn2();
}
```

If DOS 2.00 or higher is not being used it will issue a message and exit the program.

---

dwarn3

This function issues a warning message in the version of DOS in current use is not at least version 3.00

Call: dwarn3();

Example:

```
main()
{
    dwarn3();
}
```

If DOS 3.00 or higher is not being used it will issue a message and exit the program.

This function backspaces the daisy wheel printer 1/120 th of an inch.

Call: dx120bs();

Example:
```
main()
    {
        dx120bs();  /* backspace the printer 1/120 the inch */ }
```

dxback

Select backwards prining. Does not affect cr, tabs or paper movement commands.

Call: dxback();

Example:
```
main()
    {
        dxback(); /* select backwards printing */
    }
```

dxbi

This function sets the bidirectional print mode for the daisy wheel printer.

Call: dxbi();

Example:
```
main()
    {
        dxbi();  /* set bi-directional mode */
    }
```

dxboldst

This sets the daisy wheel printer to begin bold printing.

Call: dxboldst();

Example:
```
main()
    {
        dxboldst();  /* start bold printing */
    }
```


dxbs

This function backspaces the daisy wheel printer 1 full character width.

Call: dxbs();

Example: main()
{
dxbs(); /* back space the daisy wheel 1 character width */ }

---

dxbsend

This ends bold and shadow modes of printing on the daisy wheel printer.

Call: dxbsend();

Example: main()
{
dxbsend(); /* end any bold or shadow printing */
}

dxchwide

Set the character width in 1/120 ths inch. Also know as the HMI (horizontal motion index).

Call: dxchwide(n);

n = the number of 120ths of an inch

Example: main()
{

dxchwide(10); /* set to 10/120 inch or 12 pitch */ }

dxclr1tb

This clears a single tab setting on the daisy wheel printer at the current column.

Call: dxclr1tb();

Example: main()

```
{
    dxclr1tb(); /* clears a tab at current column */
}
```

Clears all horizontal tabs that were set by the dxsetht command.

Call: dxclrtab();

Example:
```
main()
    {
        dxclrtab(); /* clear ALL horizontal tabs */
    }
```

dxcr

Sends a carriage return to the printer. May or may not start a new line depending on switch settings.

Call: dxcr();

Example:
```
main()
    {
        dxcr();  /* send a carriage return */
    }
```

dxdefwid

Select the default character width or HMI.

Call: dxdefwid();

Example:
```
main()
    {
        dxdefwid(); /* select the default width according to switch settings. */ }
```

dxeject

Eject a single sheet of paper without loading a new sheet when using the single sheet feeder option.

Call: dxeject();

Example:
```
main()
    {
        dxeject();  /* eject page without loding new paper */ }
```

dxff

Send a form feed character to the daisy wheel printer.

Call: dxff();

Example: main()
{
dxff(); /* send form feed to daisy wheel printer */ }

dxfor

Selects forward printing. Used to cancel the backwards printing selected by dxback.

Call: dxfor();

Example:
```
main()
    {
        dxback();  /* select backwards printing */
        dxfor();   /* cancel backwards printing */
    }
```

dxhlf

Advances the page 1/2 of the current line spacing. Used with dxrhlf to produce sub/superscripts.

Call: dxhlf();

Example: main()

```
{
    dxhlf();  /* move half line feed for a subscript */ dxrhlf(); /* move back to previous line */ }
```


dxht

Sends a horizontal tab (HT) character to the daisy wheel printer. Tab stops must be set with dxsetht.

Call: dxht();

Example: main()
{
dxht(); /* move to next tab stop set by dxsetht */ }

dxinit

Initializes daisy wheel printer to normal power on settings.

Call: dxinit();

Example:
```
main()
    {
        dxinit(); /* cancel all previous settings and return to default power on settings */ }
```

dxlf

Advances the paper one current line spacing. Remains in current column.

Call: dxlf();

Example:
```
main()
    {
        dxlf(); /* advance paper one line */
    }
```

dxlinsp

Sets the line spacing for line feed characters to (n-1)/48 inch. The value of n can range from 1 to 126. This is sometimes refered to as vertical motion index (VMI).

Call: dxlinsp();

Example:
```
main()
    {
        dxlinsp(6); /* set line spacing to 6/48 inch or 8 lines/in */
    }
```

dxlmarg

Set the left margin at the current column.

Call: dxlmarg();

Example: main()
{
dxlmarg(); /* set the left margin to current location */ }

dxpglen

Sets the page length to n lines. This is stored as an absolute length and does not change if you change line spacing. This also sets the top of form to current paper location.

Call: dxpglen(n);

n = number of lines on one page

Example:
```
main()
    {
        dxpglen(66); /* set page to 66 lines, normal 11" paper */ }
```

dxrhlf

Roll the paper down one half current line spacing. See dxhlf for further uses and details.

Call: dxrhlf();

Example:
```
main()
    {
        dxhlf();  /* move half line feed for a subscript */ dxrhlf(); /* move back to previous line */ }
```

dxrlf

Moves paper down one line which in effect moves the print position up one line.

Call: dxrlf();

Example:
```
main()
    {
        dxrlf(); /* move print position up one line spacing */ }
```

dxsetht

Sets a horizontal tab at current position. Up to 160 HT's may be set.

Call: dxsetht();

Example:
```
main()
    {
        dxsetht();  /* set a tab at current column */
    }
```

dxshast

Select shadow print. This is canceled by dxbsend or a carriage return.

Call: dxshast();

Example:
```
main()
    {
        dxshast();   /* turn on shadow printing */
    }
```

dxundend

Cancels underlining set by dxundlst.

Call: dxundend();

Example:
```
main()
    {
        dxundlst();  /* start underline */
                                        lputch('a'); /* send an a to the printer and underline it */
        dxundend();  /* stop the underlining */
    }
```

dxunldst

Starts underlining. All characters including spaces are underlined except spaces before the first printable character or after the last printable character.

Call: dxundlst();

Example:
```
main()
    {
        dxundlst();  /* start underline */
                                    lputch('a'); /* send an a to the printer and underline it */
        dxundend();  /* stop the underlining */
    }
```

dxuni

Selects uni-directional printing. Used to print in one direction only which improves alignment when printing vertical columns.

Call: dxuni();

Example:
```
main()
    {
        dxuni();  /* select unidirectional printing */
    }
```