C99 Utilities Instructions.

.

The utilities on this disk are intended for use by those doing programming in C99. They're not readily usful for any other TI-99/4A programming language. The utilities were developed by Bruce Harrison, written entirely in Assembly language, and are being offered as Public Domain software. Thus they may be copied without limit, placed on BBS systems, in User Group libraries, etc. without compensation to the author..

.

What's on this disk?.

.

The disk contains source code, object files, and "demo" programs. There are six object files which can be used with your C99-derived object file(s). The object files are:.

.

```
ACCNUM/O  (Accept At for numbers and strings).
PUTNUM/O  (Display At for numbers and strings).
PUTNUM2/O (Display At for numbers and strings).
SEGSTR/O  (Similar to XB SEG$ function).
SNNS/O    (Convert number to string and vice versa).
KEYJOY/O  (GetKey, GetJoy, and Button).
```

.

All of these were created at the behest of Vern Jensen, of Pasadena, CA. They were designed to fill some needs he had for routines in his programs. As they turned out pretty well, we decided to offer them to the C99 portion of the TI community..

.

The ACCEPT and DISPLAY operations are unique in that when one specifies a row and column, the routine will interpret these correctly for either the 32 column grahics mode or the 40 column text mode. The C99 programmer thus can forget about what mode the screen is in, and the routine will take care of the math. Of course if you try to put something at column 35 when in the 32 column mode, you won't get the expected result..

.

ACCNUM is designed to accept an integer number in the range -32768 through 32767. The entry field is six characters wide. The "arrow" keys (Function-S and -D) move the cursor back and forth, and Function 1, 2, and 3 behave as in Basic. The user has the option to clear the entry field or not, so default entries can be placed on-screen and accepted. Except for the Function-S, -D, and -1, -2, and -3, no keys other than +, -, and 0-9 will be accepted. Numbers can be accepted anywhere in the field, so that right justified entries will be handled correctly. To invoke ACCNUM, put this in your C99 source:.

.

```
num=accnum(row,col,clr);.
```

.

Row and column are what you'd expect, num must be an integer variable, and clr is either 0 (don't clear field) or non-zero (clear field)..

.

ACCSTR accepts input of a string. There are five parameters in this case, as follows:.

.

```
accstr(row,col,maxlen,clr,buffer);.
OR.
len=accstr(row,col,maxlen,clr,buffer);.
```

.

Here maxlen is the width of the input field in characters, and buffer is a CHAR type array variable. The clr parameter works as described above. The string is reported into buffer as an ASCIZ string, in keeping with C99 practice. In the alternate form, the length of the string (without the terminating 0 byte) gets reported to an integer variable..

.

PUTNUM places an integer number on the screen at a designated row and column. It clears a six character space at that location before placing the number. This routine gives the C99 programmer the choice of either right or left justifying the number in the alloted space. It's invoked by:.

.

```
putnum(row,col,integer,just);.
```

.

The integer can be either an immediate number or the name of an integer variable. If just is 0, the number will be left justified in the space. If non-zero, the number will be right

justified in the six character space beginning at row, col..
.
PUTNUM in the object file PUTNUM2/O works the same except that it doesn't clear the field before placing the number on the screen. The file PUTNUM2/O also includes DISSTR, and that works exactly the same in both versions of PUTNUM..
.
DISSTR displays an ASCIZ string at row, col. Invoke it thus:.
.
```
disstr(row,col,string);.
```
.
String can be either a quoted string or a char array by name. This function does not provide for using the backslash character sequences..
.
SEGSTR works essentially the same as the Basic function SEG$. It will take a selected portion of one string into another string. Since this is to be used with C99, the routine expects the main string to be of ASCIZ type, and produces an ASCIZ output string. Invoke it like this:.
.
```
segstr(mainstr,substr,start,length);.
```
.
Both mainstr and substr must be defined as CHAR arrays. the string at mainstr must be defined, even if it's a null string, with zero at its first position. The routine will take charcters from mainstr starting at start position and running to a maximum of length. If start is beyond the end of mainstr, substr will be a null string. If length extends beyond the end of mainstr, substr will end at the last character of mainstr. As in the Basic case, setting start to 1 will pick up the first character of mainstr, and so on..
.
NUMTST converts an integer number to a string. It also reports the length of the string if desired. Invoke by:.
.
```
numtst(num,string);.
OR.
len=numtst(num,string);.
```
.
The string must point to a char array. Num can be either a number or the name of an integer variable..
.
STTNUM converts a string to an integer number. If the string is non-numeric, the output integer will be zero. The first character must be a numeric symbol, in the set +, -, or 0-9. Invoke like this:.
.
```
num=sttnum(string);.
```
.
String must be in ASCIZ format, in keeping with C99 practice. The resulting number will be correct so long as the string evaluates to a number in the range -32768 thru 32767. Anything after the decimal point will be correctly rounded off..
.
KEYJOY/O.
.
This object file contains three routines. Each of them takes input, but does not wait for input. Rather, these return a status variable that indicates whether input has occurred, so your C99 program can decide what to do with that input..
.
GETKEY scans the keyboard for any key-unit from 0 through 5. It reports the key found, if any, into an integer variable, and reports the status for that key-unit into a return variable which also needs to be of the int type. The status reported is for the selected key-unit only. To use GETKEY, do this:.
.
```
status=GetKey(unit,&&keyvar);.
```
.
The status variable will be 0 if no key is detected on unit. If the status reports 0, the key variable will also be set at 0. If a new key (for this unit) is detected, status will report as 1, and the struck key's value will be placed in keyvar. If the same key is detected as on the last call for this unit, status will report as -1, and the key's value will be placed in variable keyvar..

.

GETJOY checks the status and values of the joystick inputs for joystick 1 or 2. As in the case of GetKey, the status is unique to the selected unit. Call it like this:.

.

```
status=GetJoy(unit,&&xvar,&&yvar);.
```

.

Status will report as 0 and both xvar and yvar will be zero if the stick (unit) is in the neutral position. Status will report -1 if this stick is in the same non-neutral position as it was on the last call for this unit. Status will be 1 if the stick is in a different non-neutral position from the last call for this unit. GetJoy can be used only with unit numbers 1 or 2. Any other key unit in GetJoy will result in a zero status and nothing reported to the x and y variables..

.

BUTTON checks to see whether either fire button (or the Q or Y keys on the keyboard) is being pressed. Call it like this:.

.

```
status=button() (no parameters).
```

.

Status will report as 0 if neither fire button is pressed, 1 if a new button press has occurred, and -1 if a button is still being held since the last call to button. This does not report which fire button is down. (you can use GetKey with unit 1 or 2 to determine which joystick fire button is down, by looking for k=18 after the call to GetKey.).

.

ABOUT THOSE AMPERSANDS.

.

If you used our PRINTINST program to print these instructions, there will be double ampersands in the examples of how to use GetKey and GetJoy, above. You should only use a single ampersand in your source file. We put two in because otherwise the TI-Writer or Funelweb Formatter would see the ampersand as a signal to underline what came after it. Since many people will use one of those Formatters to print these instructions, we've put in double ampersands so that one will get printed by the formatter. In C99 code, the ampersand means that the reference supplied to our Assembly routine is the address of the variable rather than its value..

.

Words of Caution.

.

The C99 programmer must take care to provide sufficient room in his CHAR array variables. The array must have one byte more than the greatest string length to be allowed. If, for example, a string space is reserved like this:.

.

```
char mainstr[81].
```

.

No more than 80 characters can be placed in that array, because there must be room for a 0 byte to delimit the string at its end. The routine ACCSTR has no way of finding out what length has been reserved, so it's up to the C99 programmer to watch what he puts in the maxlen parameter..

.

We said earlier that the routines which have row, col parameters will work correctly with either 32 or 40 character screen widths. You can, however, generate your own problem with this if you should, for example, put something at column 35 when you're in 32 column mode. The routine would work, but your item would appear at column 2 of the next row on the screen. You, the programmer, have to remember little things like what mode you've placed the computer into. In the default C99 case, where the screen is 40 characters wide, that won't be a problem..

.

All of the row, col routines check for the end of the screen image table, using a limit appropriate for the width in use. In cases of input fields, such as ACCNUM and ACCSTR, the limit is checked for the end of the input field, and if the field would extend beyond the screen, the routine exits without doing anything. In the case of DISSTR, if the starting point is within the screen image, the routine will place characters from the string on screen until it reaches the end of the string or the end of the screen, whichever comes first..

.

The Demo Programs.

.

There are two demo programs on this disk, called NUM and TEST. The original C source for NUM is NUM/C, the compiled assembly source is NUM/S, the object file is NUM/O, and the Option-5 E/A Program file is just plain NUM. TEST is available only as an E/A Option 5 program file. It demonstrates use of GetKey, GetJoy, and Button. When you're finished with TEST, exit from it by pressing Function-= (QUIT). The demo program NUM exercises each and every one of the other routines..

.

First, it does ACCNUM, with a prompt at the top of the screen. When you've entered a number there, the program will put that number just below as a left-justified number. Below that, it will use PUTNUM again, but with its right-justify option turned on. Next the program will double-convert the number you just entered. It will convert to a string and then convert that string back to a number, then put the result on the screen with PUTNUM. Next, it will take a SEG of the first three characters in the string form of this number, and display that segment using DISSTR..

.

Now the program prompts for a string with a field 36 characters wide. Type in anything you like. When you've pressed <ENTER>, the program will take a segment of that string, starting at the 5th character and extending 7 characters from there. It will then display the resulting segment just below the full string. This will work even if you left the entry field blank, but of course nothing will appear on the screen where the segment would go. Last, the program reports the length of the main string on screen..

.

The program ends with an exit(0), so you'll see the familiar exit/rerun prompt near the bottom of the screen, and that will work as usual..

.

The program TEST simply serves to demonstrate the use of the KEYJOY routines. On the left side of the screen is the display of scanning key-unit 0. In the middle are the results of scanning the joysticks, with joystick one near the top of the screen and joystick 2 below that. On the right side is the results of calling BUTTON. There is a built-in delay in this program so that the screen can show you status changing from 1 to -1 if a key is held down. To exit, press Q or q on the keyboard..

.

We hope you'll find these little gems useful in your C99 programs. If you need further help, please don't hesitate to ask:.

.

Bruce Harrison.
5705 40th Place.
Hyattsville, MD 20781.
U.S.A..
Phone (301) 277-3467

.....