Table of Contents

OVERVIEW

languageONE is an extensible system, that is – it is easily extended (most languages are). Because the system is basically a collection of assembler macros, a macro may be written by anyone and used as they code. Done correctly, the macro sits in the language as though it had always been there.

So what are packages ?

Packages are a collection of macros, and their underlying subroutines, written in raw *languageONE* code, that extends *languageONE* and provides the backbone for completing common tasks. This document maps out the two such package, LCURSES.PKG that extends *languageONE* by providing greater control over the keyboard and screen and LMENUS.PKG which assist in building common menus.
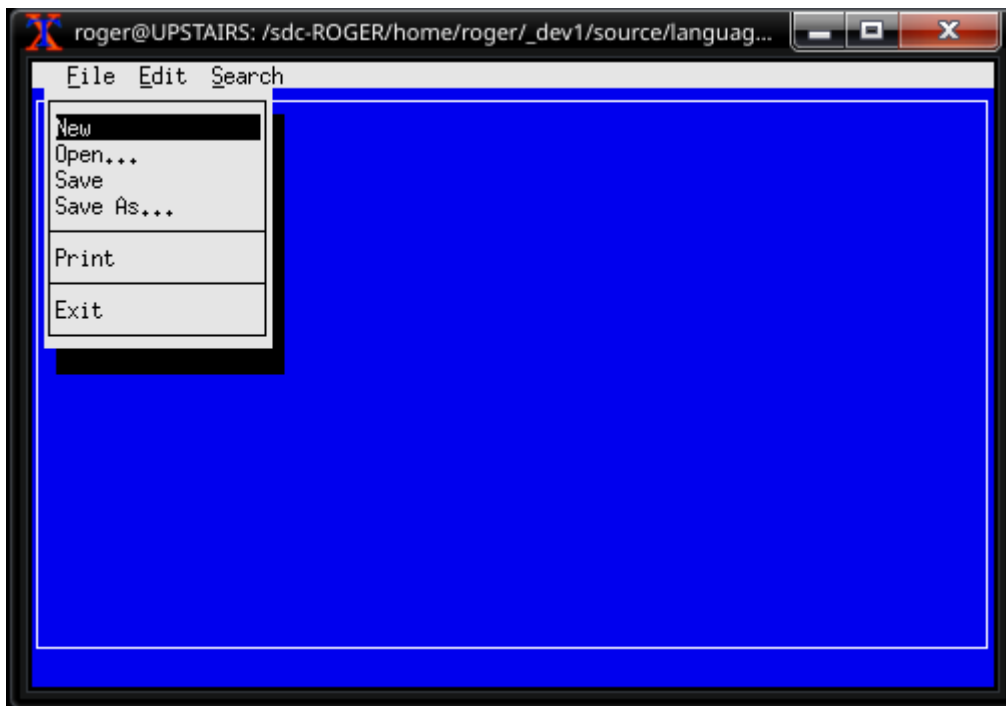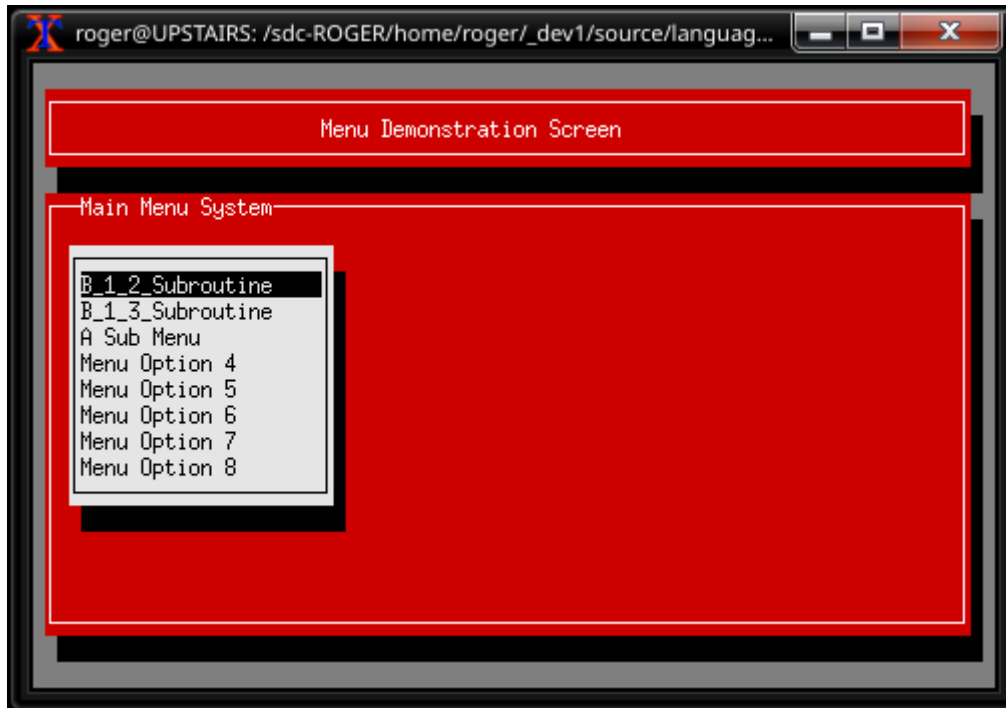
To provide access to a *languageONE* package, you must code an include statement. ie:-

> %include 'include/LCURSES.PKG'
> %include 'include/LMENUS.PKG'

NOTE:- LMENUS uses LCURSES so if you intend to use LMENUS then LCURSES must be defined alonh with it.

The following screenshots offer an example of the output that is easily attained with the lcurses and the lmenus packages.

A WORD ABOUT TERMINALS

Historically speaking a terminal is a relatively dumb electromechanical device with an input interface (like a keyboard) and an output interface (like a display). It was connected to another device (like a computer) via two logical channels, and all it does is:

- send the keystrokes down the first line
- read from the second line and display them on the screen.

escape sequences were used to control cursor location, color, font styling, and other options on video text terminals. Certain sequences of bytes, most starting with an escape and a bracket character followed by parameters, are embedded into text. The terminal interprets these sequences as commands, rather than text to display verbatim.

Almost all manufacturers of video terminals added vendor-specific escape sequences to perform operations such as placing the cursor at arbitrary positions on the screen. One example is the VT52 terminal, which allowed the cursor to be placed at an x,y location on the screen by sending the `ESC` character, a `Y` character, and then two characters representd by numerical values equal to the x,y location, plus 32 (thus starting at the ASCII space character and avoiding the control characters). The Hazeltine 1500 had a similar feature, invoked using `~`, `DC1` and then the X and Y positions separated with a comma. While the two terminals had identical functionality in this regard, different control sequences had to be used to invoke them.

As these sequences were different for different terminals, elaborate libraries such as termcap ("terminal capabilities") and utilities such as tput had to be created so programs could use the same API to work with any terminal. In addition, many of these terminals required sending numbers (such as row and column) as the binary values of the characters; for some programming languages, and for systems that did not use ASCII internally, it was often difficult to turn a number into the correct character.

The ANSI standard attempted to address these problems by making a command set that all terminals would use and requiring all numeric information to be transmitted as ASCII numbers. These standards were introduced in the 1970s to replace vendor-specific sequences and became widespread in the computer equipment market by the early 1980s. They were used in development, scientific, commercial text-based applications as well as bulletin board systems to offer standardized functionality.

Although hardware text terminals have become increasingly rare in the 21st century, the relevance of the ANSI standard persists because a great majority of terminal emulators and command consoles interpret at least a portion of the ANSI standard.

A WORD ABOUT TERMINAL EMULATION

A terminal emulator is a program that emulates the functionalities of the traditional computer terminals. In simple words, unlike the classic terminal that performed functions using hardware, the terminal emulator executes the same tasks in software.

Examples of Linux terminal emulators are xterm, konsole and gnome-terminal.

and NCURSES

ncurses (new curses) is a programming library providing an application programming interface that allows the programmer to write text based user interfaces in a terminal-independent manner. It is a toolkit for developing application software that runs under a terminal emulator.

and finally LCURSES

LCURSES maybe thought of as "*languageONE* curses" or perhaps "*light* curses". It does not try to be ncurses but tries to provide enough functionality so that a fully functioning text based interface can be easily produced within a terminal emulator. It defines the most common escape codes as mnemonics and allows their transmission and receipt via the standard *languageONE* display and accept statement. *LanguageONE* has previously provided this function but the LCURSES.PKG has been developed to give far greater control than was previously possible. It also highlights the use of *languageONE* as a macro language and how the language itself may be extended by building on itself.

Some Assumptions

Lcurses does not try to determine the type of terminal any specific emulator mimics. It assumes ANSI standard or VT52/VT100 (or combination). It has been tested with xterm, konsole and gnome-terminal and relies on the ability of these emulators to be configured to return specific escape sequence for any given key.

STDIO.LIB is the *languageONE* module that provides terminal access and it is there you will find the escape codes that are being used. As *languageONE* is delivered with all the original source code it is a simple task to modify the generic escape codes currently being used to something that may be more appropriate. Having said that, it is a more desireable path to simply configure the emulator to return that which *languageONE* expects.

One Last Thing

Originally a terminal was connected to the computer through serial cables plugged into a Universal Asynchronous Receiver and Transmitter (UART). A UART driver reads from the hardware device and applies the line discipline. The line discipline is in charge of converting special characters (like end of line, backspaces), *and echoing what has been received back to the teletype*, so that the user can see what it has been typed. It is also responsible to buffer the characters. When enter is pressed, the buffered data is passed to the foreground process for the session associated with the TTY. The whole stack as defined above is called a TTY device.

IMPORTANT:- *echoing what has been received back to the teletype*

It is undesireable to echo escape sequences as this will destroy any screen that we are trying to build. This should be turned off by the use of the following command

```
LINUX:/$  stty -echoctl
```

To make this permenant it should be coded within your .bashrc

EXAMPLES

languageONE/src/examples/**V2.13.ONE** is an example program used for the development of the lcurses and lmenus packages and can be used as a guide in developing *languageONE* programs that use the lcurses and lmenus facilities.

In addition languageONE/src/examples/**0.MENU-TEMPLATE.ONE** is a template program that can be used to build menu driven applications.

LCURSES

The LCURSES.PKG manages line drawing characters and an enhanced AcceptAt macro.

Horizontal Line

lcurses.hline (startRow,StartColumn,EndColumn)

This macro will generate a horizontal line at the specified row, starting at startingColumn and finishing at EndingColumn. The desired foreground and background colours need to established prior to this call.

Vertical Line

lcurses.vline (startCol,StartRow,EndRow)

This macro will generate a vertical line at the specified column, starting at startingRow and finishing at EndingRow. The desired foreground and background colours need to established prior to this call.

Box

Lcurses.box (startRow,startColumn,endRow,endColumn,Shadow)

A box will be drawn using the supplied parameters. Shadow should be set to c_TRUE if a shadow is required or c_FALSE if a show is not required. The desired foreground and background colours need to established prior to this call.

AcceptAt

lcurses.AcceptAt (Row,Col,FieldName)

AcceptAt enhances the Accept.At macro by acknowledge a number of function keys and returning a value by way of RETURN_CODE.

The following constants have been defined:-

    RETURN,FUNCTIONKEY1,FUNCTIONKEY2,FUNCTIONKEY3,FUNCTIONKEY4,FUNCTIONKEY5,
    FUNCTIONKEY6,FUNCTIONKEY7,FUNCTIONKEY8,FUNCTIONKEY9,FUNCTIONKEY10,
    FUNCTIONKEY11,FUNCTIONKEY12,ALT,ARROWUP,ARROWDOWN,ARROWRIGHT,ARROWLEFT,
    END,HOME,INSERT,ENDOFFIELD,ESCAPE,BACKSPACE,NULL

Additionally, STDIO now includes 2 more escape code driven variables. They are:-
        c_HideCursor
        c_ShowCursor

LMENUS

The LMENUS.PKG will draw and manage menus based on the data supplied to it. Note that it uses the lcurses.pkg which must be included in your program if you want to use it. It functions by accessing two tables that should be defined by the calling program. They are:-

The All Menus Table

which defines all the menus that the calling program wishes to manipulate. The index into the table identifies the table while the table record defines the geometry of the table. The following record must be coded within the calling programs files section.

```
begin.record c_AM_RecordLen,AM_Record
    insertnumber    AM_StartRow, 00,'99'          ; Starting row
    insertnumber    AM_StartCol, 00,'99'          ; Starting column
    insertnumber    AM_EndRow,   00,'99'          ; Ending Row
    insertnumber    AM_EndCol,   00,'99'          ; Ending Column
end.record AM_Record
insertnumber AM_Idx,             00,'99'          ; the index returned by
LMENUS.PKG
```

The Menus Table

which defines the individual menu items. It is a 2 dimensional table with the 1st dimension defined by AM_Idx sourced from the all menus table. The following record must be coded within the calling programs files section.

```
begin.record c_M_RecordLen,M_Record
    insertword      M_Type,      01,''"          ; the menu entry type
    insertnumber    M_Target,    00,'99999999'   ; the target
    insertword      M_Entry,     55,''"          ; and the text
end.record M_Record
insertnumber M_Idx,              00,'99'          ; the index returned by
LMENUS.PKG
```

NOTE: that c_AM_RecordLen and c_M_RecordLen are predefined constants made availabale by lcueses, while constants c_AM_NoOfRecords and c_M_NoOfRecords used by the lmenus program (and your own) must be established preceeding the include statement. Define them as:-

```
%define c_AM_NoOfRecords        nn
%define c_M_NoOfRecords         nn
```

As with all tables in *languageONE* there must be an entry in the tables section:-

```
inserttable AM_Table,c_AM_RecordLen*8              ; Allows for 8 menus
inserttable M_Table,c_M_RecordLen*8*16             ; Allows for 8 menus with 16 entries
```

and they must be bound in the instructions section:-

```
tables.bind AM_Table,AM_Record,8
tables.bind M_Table,M_Record,8,16
```

Menu Items

lmenus.MenuItems (AM_Idx,M_Idx,M_Type,M_Target,M_Entry)

The Menus record may be populated and loaded manually but the lmenus.MenuItems has been defined to make this a little easier.

M_TYPE

S:	A Subroutine in this (or a linked) program
M:	A sub Menu
X:	An external program
:	Space is end of options and must be defined to mark the end of the menu

M_TARGET

M_Type = S:	Subroutine name
M_Type = M:	Menu No
M_Type = X:	0

M_ENTRY

The text that appears on the menu

NOTE:- That if this table is loaded manually, the target must be loaded using the following fragments of assembler code:-

- If the Target is a Menu, the target must be loaded with it's number
        mov qword[M_Target],Menu Number
- while if the target is a subroutine the address must be loaded as
        mov rax,SubRoutine Name
        mov qword[M_Target],rax

Menu

lmenus.Menu (AM_Idx,c_TRUE/c_FALSE)

The menu macro will pass handling of the menu over to lmenus to manage. This routine simply handles the ARROWUP and ARROWDOWN keys while returning the RETURN and ESCAPE keys along with the ARROWLEFT and ARROWRIGHT keys in RETURN_CODE.

When the RETURN key has been pressed AM_Idx and M_Idx will have been set and the table record has been read.

When the second parameter is set to c_TRUE lmenus will wait for a user response. When the second parameter is set to c_FALSE, the menu will be displayed and lmenus will return control to the calling program.

Call

lmenus.Call (M_Target)

This macro should be used (in place of the normal $Call) when a sub routine name has been returned the lmenus.Menu.