**NCURSES LIBRARY FUNCTIONS AND MACROS**

What is ncurses ? when we login to our terminal, we don't have the desktop interface to our system. Instead, we have a dump terminal with bash running and waiting for our commands. How does then VI our editor works then ? How does it lets us create text files ? Thankfully, we have ncurses library. To create text based applications, we use ncurses library functions to perform cursor and screen manipulations. Without curses library calls, developers have to handle display of characters, cursor positions, creation of menus, events and the program will be very messy and difficult to maintain. Ncurses provides variety of function calls and macros for the developer not to be bothered with the complicated terminals sizes and characteristics, but focus on the application logic. Curses functions such as move the cursor, display a character or string, highlight or underline characters, handle key presses, create menus and windows and other functions empower the developers provide a rich presentation to the user. You may think, oh no yet another thing to learn. Fortunately, ncurses rescues us by supporting C type syntax writing curses applications is much easier.

We will introduce basic library calls with which you can create any simple applications.

Here is the simple program that demonstrates these functions - Example 1 :

|  |  |
| --- | --- |
| Program : initialization and refresh | |
| #include <curses.h>  int main ( void )  {  initscr ( ) ;  addch ( 'A' ) ;  refresh ( ) ;  sleep ( 3 ) ;  endwin ( ) ;  return 0;  } | We include the header file curses.h  Then in the main function, we call the initscr ( ) . in Line# 6, we call addch ( 'A' ) . This is a macro call that will display the letter X at the top left corner of the window.  To display the letter A, we have to call refresh ( ) function, otherwise the letter will be saved in the memory, but not displayed. If you want to update the display, refresh has to be called.  Then, we pause the program for 3 seconds, followed by endwin ( ) that forces curses library to release all data structures and memory before exiting. |
| gcc ex1.c –lcurses –o ex1 | if you are curious about the macro expansions, you can use the preprocessor option –E |

**initscr ( )** – library call is normally the first curses routine to call when initializing a program. The initscr ( ) code determines the terminal type and initializes all curses data structures. initscr also calls to refresh to clear the screen. If errors occur, initscr writes an appropriate error message to standard error and exits; otherwise, a pointer of type WINDOW is returned , named stdscr.

**What is stdscr pointer?** – You have heard of pointers, yes it is a pointer of type WINDOW . The ncurses library permits manipulation of data structures, called windows, which can be thought of as two-dimensional arrays of characters representing all or part of a CRT screen. A default window called stdscr is returned when you call initscr ( ) . This pointer is used by subsequent calls to control the screen.

**refresh ( )** - After using routines to manipulate a window, refresh ( ) is called to get actual output to the terminal, as other routines merely manipulate data structures. The refresh routine uses stdscr as the default window. Without calling refresh ( ) , your screen won't be updated.

**endwin ( )** - And finally don't forget to end the curses mode and endwin() must be called before your program exits. Otherwise your terminal might behave strangely after the program quits. endwin() frees the memory taken by curses sub-system and its data structures and puts the terminal in normal mode.

**How do you describe the screen size and layout ?** Your screen is a window with set number of rows and set of columns. Usually, a position on the screen is represented by the coordinate as (Y, X) where Y is the rows and X is the column. Unlike the regular graph, Y is the row and X is the column. All rows and columns start with zero. The coordinate ( 0, 0 ) is at the top left corner. Location ( 0, 1 ) is the first row but second column. Location ( 0, 2 ) is first row, second column. Location ( 2, 0 ) is the third row from the top going downward, and first column.

**How do I compile curses program ?** To compile, you have to use the option –lcurses. If you have a program file say, test.c , then you compile as :

ex: gcc test.c –lcurses

**How do I display a character ?** addch ( chtype ch ) - puts the character *ch* into the given window at its current window position, which is then advanced. They are analogous to putchar in our C language. If the advance is at the right margin, the cursor automatically wraps to the beginning of the next line. At the bottom of the current scrolling region, if scrollok is enabled, the scrolling region is scrolled up one line. This return the integer ERR upon failure and OK on success.

**What is chtype ?** Is it different from char ? Yes. The characters in a window are actually of type chtype, (character and attribute data) so that other information about the character may also be stored with each character. Attributes such as BOLD, underline, reverse background. We will discuss little later though we only skim over the subject.

**How do I display a string ?** addchstr ( const chtype \*str ) – This copy *chstr* into the window image structure at and after the current cursor position. The window cursor is *not* advanced, and do not perform any kind of checking (such as for the newline, backspace, or carriage return characters), do not advance the current cursor position, and truncates the string if it crosses the right margin, rather than wrapping it around to the new line.

|  |  |
| --- | --- |
| program to print strings using addstr function | |
| initscr ( ) ;  addstr ( "Hello " ) ;  addch ( ' ' ) ;  addstr ( "World " ) ;    refresh ( ) ;  sleep ( 3 ) ;  endwin ( ) ; |  |
|  |  |

QUIZ: Name atleast three differences between addch and addchstr ?

**How do I display a character at some location ?** mvaddch ( int y, int x, chtype ch ) is another curses library call. It takes y as the row number, x as the column number and the character ch. it uses the stdscr as the default WINDOW pointer.

**How do I display a string at some location ?** mvaddch ( int y, int x, const chtype \*str) ; is another curses library call. It takes y as the row number, x as the column number and the string str as pointer. it uses the stdscr as the default WINDOW pointer.

|  |  |
| --- | --- |
| Program using mv prefixed curses library calls. |  |
| initscr ( ) ;  mvaddstr (5,0, "Hello" );  mvaddch ( 5,5,' ');  mvaddstr ( 5,6, "World" );  refresh ( );  sleep ( 3 );  endwin ( ) ; | We print Hello in row#=5 and column#=0  Then, we print a space  then we print World appropriately. |
|  |  |

**How do I know the screen or terminal size ?** GetMaxYX ( ) macro provides the screen size information. The syntax is getmaxyx ( stdscr , y, x ) where y and x are integers. The macro call populates the y and x values. We demonstrate this with another simple program

|  |  |
| --- | --- |
| Program using getmaxyx which uses stdscr ( ) and printw | |
|  | Here we introduce another curses library calls.  printw which is much similar to our printf function in C. |
|  |  |

**How to get characters typed by the user ?** The ncurses library provides several functions which let an application change the way input from the terminal is handled. The following functions read a single character from the terminal. But there are several subtle facts to consider. For example if you don't use the function cbreak( ) (described below), curses will not read your input characters contiguously but will begin read them only after a new line or an EOF is encountered. In order to avoid this, the cbreak() function must used so that characters are immediately available to your program. Another widely used function is noecho(). As the name suggests, when this function is set (used), the characters that are keyed in by the user will not show up on the screen.

**cbreak ( )** - Normally, the tty driver buffers typed characters until a newline or carriage return is typed. The cbreak routine disables line buffering and erase/kill character-processing making characters typed by the user immediately available to the program.

**nocbreak ( )** routine returns the terminal to normal (cooked) mode from cbreak ( ) mode.

**getch ( )** routine read a character from the window. In delay mode (ie nocbreak ( ) mode), the program waits until the system passes text through to the program. Depending on the setting of cbreak, this is after one character (cbreak mode), or after the first newline (nocbreak mode)

|  |  |
| --- | --- |
| Program using getch ( ) and nocbreak ( ) | using noecho ( ) , typed character won't be visible |
| nocbreak ( );  int ch = getch ( ) ;  if ( ch == 'q' )  addstr ( "quitting" );  else  addstr ( "keep typing " ); | nocbreak ( );  noecho ( ) ;  int ch = getch ( ) ;  if ( ch == 'q' )  addstr ( "quitting" );  else  addstr ( "keep typing " ); |
|  |  |

|  |  |
| --- | --- |
| Program using move and cbreak ( ) where characters are readily available to the program | |
| cbreak ( );  move (5,5);  int ch = getch ( ) ;  if ( ch == 'q' )  addstr ( "quitting" );  else  addstr ( "keep typing " ); | cbreak ( );    int ch = mvgetch ( ) ;  if ( ch == 'q' )  addstr ( "quitting" );  else  addstr ( "keep typing " ); |
|  | Notes: program combines move and getch |

**scanw ( )** is also another function, much similar to scanf function.

**getstr ( char \*str )** - The function getstr is equivalent to a series of calls to getch, until a newline or carriage return is received (the terminating character is not included in the returned string). The resulting value is placed in the area pointed to by the character pointer str.

**int mvgetstr(int y, int x, char \*str)** is another function to read a string from a location ( y , x )

Why do some library calls are prefixed with mv w : Many curses routines have two or more versions. The routines prefixed with w require a window argument. This window argument can created with a call to the newterm function or using the default stdscr pointer. Those without a prefix generally use the default stdscr.

The routines prefixed with mv require a y and x coordinate to move to before performing the appropriate action. The mv routines imply a call to move before the call to the other routine. The coordinate y always refers to the row (of the window), and x always refers to the column.

Here is the table of function that require the WINDOW argument

|  |  |
| --- | --- |
| Macros or Functions taking WINDOW pointer argument | |
| waddch ( WINDOW \*win, chtype ch ) ; | mvwaddch  (WINDOW \*win, int y, int x, chtype ch ) ; |
|  |  |
| waddchstr (WINDOW \*win, const chtype \*str ) ; | mvwaddch  (WINDOW \*win, int y, int x, const chtype \*str) ; |
|  |  |

**How to set attributes to the characters ?** We can display a character as BOLD , UNDERLINE, BLINK, REVERSE background video, DIM. There are constants for each of these as shown in the table. We can set an attribute to a specific character or to a group of statements

|  |  |
| --- | --- |
| Attribute | constant |
| Normal Display | A\_NORMAL |
| underline | A\_UNDERLINE |
| blink | A\_BLINK |
| bold | A\_BOLD |
| reverse | A\_REVERSE |

|  |  |
| --- | --- |
| Example of attribute to a specific character vs using attron to statements | |
| addch ( 'X' );  addch ( 'Y' | A\_UNDERLINE );  addch ( 'Z' );  Here we BIT ORed with a specific character | attron ( A\_UNDERLINE ) ;  addch ( 'X' );  addch ( 'Y' );  addch ( 'Z' );  attroff ( A\_UNDERLINE ) ; |
|  |  |

You can bitwise OR them together to set a particular attribute using attron and turn off using attroff calls.

**How do I interact with the keyboard ?** The character typed by the user can be compared with function keys too. You need to enable the window to accept these keys by calling the function keypad ( ) . Here is the table

|  |  |  |  |
| --- | --- | --- | --- |
| Handle FUNCTION keys and arrow keys. call keypad ( ) first. | | | |
| KEY\_UP | KEY\_DOWN | KEY\_LEFT | KEY\_RIGHT |
| KEY\_DL | KEY\_BACKSPACE | KEY\_ENTER |  |
| NOTE: Every operating system delivers they keys differently. Some of you might encounter different experience with these. | | | |

|  |  |
| --- | --- |
| Simple application that handles up, down, left or right arrows | |
| initscr ( ) ;  keypad ( stdscr, TRUE);  noecho ( ) ;  cbreak;  int ch = getch ( ) ;  while ( ch != 'q' ) {  clear( );  wmove (stdscr, 0,0);  switch ( ch ) {  case KEY\_UP :  printw ( "you pressed UP KEY ");  break;  case KEY\_DOWN :  printw ( "you pressed DOWN KEY ");  break;  case KEY\_LEFT :  printw ( "you pressed LEFT KEY ");  break;  case KEY\_RIGHT :  printw ( "you pressed RIGHT KEY ");  break;  default :  printw ( "you pressed %c ", ch);  break;  }  ch = getch ( ) ;  }  refresh ( );  sleep ( 3 );  endwin ( ) ;  return 0 ; |  |
|  |  |

|  |  |
| --- | --- |
| Simple application that prompts the user to enter the correct answer: | |
| nocbreak( ) ;  mvprintw ( 0,0, "What is 2+3:");  move ( 0, 13);  int ch = getch ( ) ;  if ( ch == '5')  mvaddstr ( 1,0, "Good Job" );  else  mvaddstr ( 1,0, "Oh no... " );  refresh ( ) ;  sleep ( 3 );  endwin ( ) ; |  |

|  |  |
| --- | --- |
| Program to check if the password requirements are satisfied | |
| int main ( void )  {  unsigned char symbols = 0, digits=0, uppercase=0, lowercase=0;  initscr ( ) ;  noecho ( ) ;  cbreak;  wmove (stdscr, 0,0);  clear( );  int ch = getch ( ) ;  while ( ch != '\n' ) {  if ( ch == '@' || ch == '#' || ch == '$' || ch == '%' ||  ch == '&' || ch == '\*' || ch == ')' || ch == '(' ||  ch == '+' || ch == '-' )  symbols = 1;  else  if ( ch >= 65 && ch <= 90 )  uppercase = 1;  else  if ( ch >= 97 && ch <= 122 )  lowercase = 1;  else  if ( ch >= 49 && ch <= 57 )  digits = 1;  ch = getch ( ) ;  }  displayError (symbols, lowercase, digits, uppercase ) ;  refresh( );  sleep ( 8 );  endwin ( ) ;  return 0 ;  } | void displayError ( unsigned char s,  unsigned char l, unsigned char d,  unsigned char u )  {  clear ( ) ;  move ( 1, 0 ) ;  if ( d == 0 )  printw ( "You did not enter digits in your password") ;  move ( 2, 0 ) ;  if ( u == 0 )  printw ( "You did not enter uppercase letter in your password");  move ( 3, 0 ) ;  if ( l == 0 )  printw ( "You did not enter lowercase letter in your password");  move ( 4, 0 ) ;  if ( s == 0 )  printw ( "You did not enter symbols letter in your password");  } |
|  |  |