What a Terminal Can Do

- `printf` only prints strings line by line sequentially in black (or grey or some boring) color
- `scanf` only gets strings one by one
- Think about vi, top and many other tools
  - They use your terminal like a drawing paper
What a Terminal Can Do

- `echo ^[[0;31;40m Red`
- `echo ^[[0;37;40m`
  - `^[' : escape character (CTRL+V and then ESC)`
- `termcap and terminfo`
  - Early libraries that control terminals
Curses Library

- Curses is a pun on the name “cursor optimization”
- A wrapper over working with raw terminal codes
  - Appeared in original SVR4
- Provides highly flexible and efficient API
  - Move cursor
  - Create windows
  - Produce colors
  - play with mouse
  - etc.
Ncurses Library

- New Curses
- Derived from pcurses
  - Developed by Pavel Curtis in 1982
- Released in 1993
- Improved by Eric S. Raymond till 1995
- Includes many new features such as form and menu libraries
- You can easily install ncurses with apt-get in Ubuntu
  - `apt-get install ncurses-dev`
Hello World

#include <ncurses.h>

int main()
{
    initscr(); /* Start curses mode */
   printw("Hello World !!!"); /* Print Hello World */
   refresh(); /* Print it on to the real screen */
   getch(); /* Wait for user input */
    endwin(); /* End curses mode */

    return 0;
}

- gcc <program file> -lncurses
Dissection of Hello World

- `initscr()`
  - initializes the terminal in curses mode
  - clears the screen and presents a blank screen
  - This has to be called first

- `printw()`
  - curses counterpart of printf

- `refresh()`
  - shows any updates made after last refresh call
  - Never forget to call refresh after making some changes
Dissection of Hello World

- endwin()
  - ends the curses mode
  - Call this before you terminate a program
  - Otherwise your terminal might behave strangely
Initialization

- `cbreak()` / `nocbreak()`
  - Disable/enable line-buffering

- `echo()` / `noecho()`
  - Disable / enable echoing of characters typed by the user to the terminal

- `keypad()`
  - Enable function key inputs
# Initialization Example

```c
#include <ncurses.h>

int main()
{
    int ch;

    initscr(); /* Start curses mode */
    raw(); /* Line buffering disabled */
    keypad(stdscr, TRUE); /* We get F1, F2 etc.. */
    noecho(); /* Don't echo() while we do getch */

   printw("Type any character to see it in bold\n");
    ch = getch();
    /* If raw() hadn't been called 
     * we have to press enter before it  
     * gets to the program */
    if(ch == KEY_F(1)) /* Without keypad enabled this will */
        printw("F1 Key pressed"); /* not get to us either */
    /* Without noecho() some ugly escape character might have been printed on screen */
    else
        printw("The pressed key is ");
        attron(A_BOLD);
        printw("%c", ch);
        attroff(A_BOLD);
}
refresh(); /* Print it on to the real screen */
getch(); /* Wait for user input */
endwin(); /* End curses mode */
return 0;
```
A window is an imaginary screen defined by curses system

Default window is \textit{stdscr}

\textit{w}-variant functions of standard curses functions will do the same things but on the given window, not on \textit{stdscr}

- \texttt{wprintw(win, \textquote{Hello World})};
- \texttt{wrefresh(win)}
Creating a Window

- A window is a part of screen that can be manipulated separately from the other windows.
- A window can be created by calling the function `newwin()`.
  - `newwin()` does not create anything on screen.
  - `newwin()` returns the pointer to the new window.
- `delwin()` destroys a window.
WINDOW *create_newwin(int height, int width, int starty, int startx) {
    WINDOW *local_win;

    local_win = newwin(height, width, starty, startx);
    box(local_win, 0, 0); /* 0, 0 gives default characters
                           * for the vertical and horizontal
                           * lines                        */
    wrefresh(local_win); /* Show that box                  */

    return local_win;
}
Window Border Drawing Example

```c
void destroy_win(WINDOW *local_win)
{
    /* box(local_win, ' ', ' '); : This won't produce the desired 
    * result of erasing the window. It will leave it's four corners 
    * and so an ugly remnant of window. 
    */
    wborder(local_win, ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ');
    /* The parameters taken are 
    * 1. win: the window on which to operate 
    * 2. ls: character to be used for the left side of the window 
    * 3. rs: character to be used for the right side of the window 
    * 4. ts: character to be used for the top side of the window 
    * 5. bs: character to be used for the bottom side of the window 
    * 6. tl: character to be used for the top left corner of the window 
    * 7. tr: character to be used for the top right corner of the window 
    * 8. bl: character to be used for the bottom left corner of the window 
    * 9. br: character to be used for the bottom right corner of the window 
    */
    wrefresh(local_win);
    delwin(local_win);
}
```
int main(int argc, char *argv[]) {
    WINDOW *my_win;
    int startx, starty, width, height;
    int ch;

    initscr(); /* Start curses mode */
cbreak(); /* Line buffering disabled, Pass on */
    keypad(stdscr, TRUE); /* I need that nifty F1 */

    height = 3;
    width = 10;
    starty = (LINES - height) / 2; /* Calculating for a center placement */
    startx = (COLS - width) / 2; /* of the window */
   printw("Press F1 to exit");
    refresh();
    my_win = create_newwin(height, width, starty, startx);
    while((ch = getch()) != KEY_F(1))
    {
        switch(ch)
        {
            case KEY_LEFT:
                destroy_win(my_win);
                my_win = create_newwin(height, width, starty,--startx);
                break;
            case KEY_RIGHT:
                destroy_win(my_win);
                my_win = create_newwin(height, width, starty,++startx);
                break;
            case KEY_UP:
                destroy_win(my_win);
                my_win = create_newwin(height, width, --starty,startx);
                break;
            case KEY_DOWN:
                destroy_win(my_win);
                my_win = create_newwin(height, width, ++starty,startx);
                break;
        }
    }
    endwin(); /* End curses mode */
    return 0;
}
Output Functions

- `addch()`
  - prints single character with attributes
  - `mvaddch()`, `waddch()` and `mvwaddch()`

- `printw()`
  - prints formatted output similar to `printf`

- `addstr()`
  - prints strings
Simple printw Example

```c
char mesg[]="Just a string";    /* message to be appeared on the screen */
int row, col;                  /* to store the number of rows and */
                                /* the number of columns of the screen */
heitscr();      /* start the curses mode */
getmaxyx(stdscr, row, col);    /* get the number of rows and columns */
mvprintw(row/2, (col - strlen(mesg))/2, "%s", mesg);

/* print the message at the center of the screen */
mvprintw(row-2, 0, "This screen has %d rows and %d columns\n", row, col);
printw("Try resizing your window(if possible) and then run this program again");
refresh();
getch();
endwin();
```
Input Functions

- `getch()`
  - gets a character

- `scanw()`
  - gets formatted input
  - `mvscanw()`, `wscanw()` and `mvwscanw()`

- `getstr()`
  - gets strings
### scanw Example

```c
char msg[]="Enter a string: "; /* message to be appeared on the screen */
char str[80]; /* message to be appeared on the screen */
int row, col; /* to store the number of rows and */
    /* the number of columns of the screen */
initscr(); /* start the curses mode */
getmaxyx(stdscr, row, col); /* get the number of rows and columns */
mvprintw(row/2, (col-strlen(msg))/2, "%s", msg);
/* print the message at the center of the screen */
getstr(str);
mvprintw(LINES - 2, 0, "You Entered: %s", str);
getch();
endwin();
```
Attributes

- `attron()` can enable multiple attributes
  - `attron(A.Reverse | A.Blink);`
- Attribute macros are as follows
  - A.NORMAL
  - A.STANDOUT
    - Best highlighting mode of terminal
  - A.UNDERLINE
  - A.REVERSE
  - A.BLINK
  - A.DIM
  - A.BOLD
  - A.PROTECT
  - A.INVIS
Attributes

- *attron vs attrset*
  - attron switches on only the given attributes
  - attrset resets the current attributes and again sets the given attributes

- *attr_get*
  - gets current attributes and color pair
Colors

- To start using colors, you should first call `start_color()`
- `has_colors()` function returns TRUE if your terminal can provide colors
- A color pair is a pair of foreground and background colors
- Color configuration must be picked from predefined pairs

```c
start_color(); /* Start color */
init_pair(1, COLOR_RED, COLOR_BLACK);
attron(COLOR_PAIR(1));
printw(stdscr, "Voila !!! In color ...");
attroff(COLOR_PAIR(1));
```
Interfacing with Keyboard

- Keyboard input can be obtained via `getch()` function

- `getch()`
  - returns ASCII codes for normal inputs
  - returns special key codes for special keys
  - Special key codes are defined as macros in `curses.h`
  - E.g.) `KEY_LEFT`, `KEY_BACKSPACE` and so on...
Interfacing with Mouse

- A mouse event is treated like a keyboard input
  - KEY_MOUSE

- You must enable mouse events you want to catch beforehand with mousemask()
  - mousemask (mmask_t newmask, mmask_t *oldmask)
  - ALL_MOUSE_EVENTS

- `getmouse()` retrieves information about the event

```c
MEVENT event;
ch = getch();

if(ch == KEY_MOUSE)
if(getmouse(&event) == OK)
    /* Do some thing with the event */
```
Interfacing with Mouse

- Event structure

```c
typedef struct {
    short id;     /* ID to distinguish multiple devices */
    int x, y, z;  /* event coordinates */
    mmask_t bstate; /* button state bits */
} MEVENT;
```
Extra Features

- Menus Library
- Panel Library
- Forms Library
- And so on...
Many parts of this slide was imported from NCURSES Programming Howto originally written by Pradeep Padala