Curses Library

- **curses**: a library for screen manipulation

Curses Library

- **curses**: a library for screen manipulation
- terminal independent

Curses Library

- **curses**: a library for screen manipulation
- terminal independent
- **window**: an independent rectangular area of characters displayed on the screen (a WINDOW data structure)

Curses Library

- **curses**: a library for screen manipulation
- terminal independent
- **window**: an independent rectangular area of characters displayed on the screen (a WINDOW data structure)
- **standard screen**: (stdscr) fills entire virtual terminal
  Holds image of terminal screen (screen image)
  One character in an array for every character shown on the screen
  Initially, the screen is filled with blanks.
  Most calls to the curses library work with *stdscr*.

Curses Library

- **curses**: a library for screen manipulation
- terminal independent
- **window**: an independent rectangular area of characters displayed on the screen (a WINDOW data structure)
- **standard screen**: (stdscr) fills entire virtual terminal
  Holds image of terminal screen (screen image)
  One character in an array for every character shown on the screen
  Initially, the screen is filled with blanks.
  Most calls to the curses library work with *stdscr*.
- Include `#include <curses.h>`

Curses Library

- **curses**: a library for screen manipulation
- terminal independent
- **window**: an independent rectangular area of characters displayed on the screen (a WINDOW data structure)
- **standard screen**: (stdscr) fills entire virtual terminal
  Holds image of terminal screen (screen image)
  One character in an array for every character shown on the screen
  Initially, the screen is filled with blanks.
  Most calls to the curses library work with *stdscr*.
- Include `#include <curses.h>`
- Link to `-lncurses`

Curses: Features

- Curses supports both a tiled-window paradigm and overlapping windows

Curses: Features

- Curses supports both a tiled-window paradigm and overlapping windows
- Curses is a text-only display tool

Curses: Features

- Curses supports both a tiled-window paradigm and overlapping windows
- Curses is a text-only display tool
- Move the cursor to any point on the screen

Curses: Features

- Curses supports both a tiled-window paradigm and overlapping windows
- Curses is a text-only display tool
- Move the cursor to any point on the screen
- Insert text anywhere on the screen

Curses: Features

- Curses supports both a tiled-window paradigm and overlapping windows
- Curses is a text-only display tool
- Move the cursor to any point on the screen
- Insert text anywhere on the screen
- Manage each window independently

Curses: Features

- Curses supports both a tiled-window paradigm and overlapping windows
- Curses is a text-only display tool
- Move the cursor to any point on the screen
- Insert text anywhere on the screen
- Manage each window independently
- Draw boxes around windows using character(s) of your choice

Curses: The Current Screen

- Curses does not know what is currently showing on the terminal display; instead, it keeps an image in an array called the *curscr* (current screen).

Curses: The Current Screen

- Curses does not know what is currently showing on the terminal display; instead, it keeps an image in an array called the *curscr* (current screen).
- *stdscr* and *curscr* are both created by initscr(); they cover the entire terminal

Curses: The Current Screen

- Curses does not know what is currently showing on the terminal display; instead, it keeps an image in an array called the *curscr* (current screen).
- *stdscr* and *curscr* are both created by initscr(); they cover the entire terminal
- Curses only does an update of the terminal display when refresh() is called

Curses: The Current Screen

- Curses does not know what is currently showing on the terminal display; instead, it keeps an image in an array called the *curscr* (current screen).
- *stdscr* and *curscr* are both created by initscr(); they cover the entire terminal
- Curses only does an update of the terminal display when refresh() is called
- Refresh attempts to minimize the number of characters issued to the terminal

Curses: The Current Screen

- Curses does not know what is currently showing on the terminal display; instead, it keeps an image in an array called the *curscr* (current screen).
- *stdscr* and *curscr* are both created by initscr(); they cover the entire terminal
- Curses only does an update of the terminal display when refresh() is called
- Refresh attempts to minimize the number of characters issued to the terminal
- It does so by only sending characters that have changed since the last refresh(). Curses compares *stdscr* and *curscr* to determine what characters to send.

Curses: The Current Screen

- Curses does not know what is currently showing on the terminal display; instead, it keeps an image in an array called the *curscr* (current screen).
- *stdscr* and *curscr* are both created by initscr(); they cover the entire terminal
- Curses only does an update of the terminal display when refresh() is called
- Refresh attempts to minimize the number of characters issued to the terminal
- It does so by only sending characters that have changed since the last refresh(). Curses compares *stdscr* and *curscr* to determine what characters to send.
- Curses functions write to another SCREEN, *stdscr*

Curses: The Current Screen

- Curses does not know what is currently showing on the terminal display; instead, it keeps an image in an array called the *curscr* (current screen).
- *stdscr* and *curscr* are both created by initscr(); they cover the entire terminal
- Curses only does an update of the terminal display when refresh() is called
- Refresh attempts to minimize the number of characters issued to the terminal
- It does so by only sending characters that have changed since the last refresh(). Curses compares *stdscr* and *curscr* to determine what characters to send.
- Curses functions write to another SCREEN, *stdscr*
- refresh() compares *stdscr* to *curscr*, only sending characters that have been modified

Curses: The Current Screen

- Curses does not know what is currently showing on the terminal display; instead, it keeps an image in an array called the *curscr* (current screen).
- *stdscr* and *curscr* are both created by initscr(); they cover the entire terminal
- Curses only does an update of the terminal display when refresh() is called
- Refresh attempts to minimize the number of characters issued to the terminal
- It does so by only sending characters that have changed since the last refresh(). Curses compares *stdscr* and *curscr* to determine what characters to send.
- Curses functions write to another SCREEN, *stdscr*
- refresh() compares *stdscr* to *curscr*, only sending characters that have been modified
- Consequently: don't mix the usual output routines (printf(), puts(), etc) with curses output routines; then curscr would no longer correspond with what the terminal is showing!

Curses: Initialization and Termination

```
#include <curses.h>
WINDOW *initscr(void);
int     endwin(void);
bool    isendwin(void);
SCREEN  *newterm(char *type, FILE *outfd, FILE *infd);
SCREEN  *set_term(SCREEN *new);
void    delscreen(SCREEN* sp);
```

- Use initscr() to initialize the screen *(generating a WINDOW)*. It creates two window handles: stdscr and curscr.

Curses: Initialization and Termination

```
#include <curses.h>
WINDOW *initscr(void);
int     endwin(void);
bool    isendwin(void);
SCREEN *newterm(char *type, FILE *outfd, FILE *infd);
SCREEN *set_term(SCREEN *new);
void    delscreen(SCREEN* sp);
```

- Use initscr() to initialize the screen *(generating a WINDOW)*. It creates two window handles: stdscr and curscr.
- initscr() determines the terminal type and initializes all curses data structures

Curses: Initialization and Termination

```
#include <curses.h>
WINDOW *initscr(void);
int       endwin(void);
bool      isendwin(void);
SCREEN  *newterm(char *type, FILE *outfd, FILE *infd);
SCREEN  *set_term(SCREEN *new);
void      delscreen(SCREEN* sp);
```

- Use initscr() to initialize the screen *(generating a WINDOW)*. It creates two window handles: stdscr and curscr.
- initscr() determines the terminal type and initializes all curses data structures
- A program that controls more than one terminal should use the newterm() routine instead of initscr() for each terminal *(generating a SCREEN)*

Curses: Initialization and Termination

```
#include <curses.h>
WINDOW *initscr(void);
int     endwin(void);
bool    isendwin(void);
SCREEN *newterm(char *type, FILE *outfd, FILE *infd);
SCREEN *set_term(SCREEN *new);
void    delscreen(SCREEN* sp);
```

- Use initscr() to initialize the screen *(generating a WINDOW)*. It creates two window handles: stdscr and curscr.
- initscr() determines the terminal type and initializes all curses data structures
- A program that controls more than one terminal should use the newterm() routine instead of initscr() for each terminal *(generating a SCREEN)*
- A curses-using program should always call endwin() before exiting (restoring tty modes, moving the cursor to the lower left-hand corner, resetting the terminal to a non-visual state)

Curses: Initialization and Termination

```
#include <curses.h>
WINDOW *initscr(void);
int     endwin(void);
bool    isendwin(void);
SCREEN  *newterm(char *type, FILE *outfd, FILE *infd);
SCREEN  *set_term(SCREEN *new);
void    delscreen(SCREEN* sp);
```

- Use initscr() to initialize the screen *(generating a WINDOW)*. It creates two window handles: stdscr and curscr.
- initscr() determines the terminal type and initializes all curses data structures
- A program that controls more than one terminal should use the newterm() routine instead of initscr() for each terminal *(generating a SCREEN)*
- A curses-using program should always call endwin() before exiting (restoring tty modes, moving the cursor to the lower left-hand corner, resetting the terminal to a non-visual state)
- Use isendwin() to determine if endwin() has already been called

Curses: Initialization and Termination

```
#include <curses.h>
WINDOW *initscr(void);
int     endwin(void);
bool    isendwin(void);
SCREEN *newterm(char *type, FILE *outfd, FILE *infd);
SCREEN *set_term(SCREEN *new);
void    delscreen(SCREEN* sp);
```

- Use initscr() to initialize the screen *(generating a WINDOW)*. It creates two window handles: stdscr and curscr.
- initscr() determines the terminal type and initializes all curses data structures
- A program that controls more than one terminal should use the newterm() routine instead of initscr() for each terminal *(generating a SCREEN)*
- A curses-using program should always call endwin() before exiting (restoring tty modes, moving the cursor to the lower left-hand corner, resetting the terminal to a non-visual state)
- Use isendwin() to determine if endwin() has already been called
- Use set_term() to switch between different terminals

Curses: Initialization and Termination

```
#include <curses.h>
WINDOW *initscr(void);
int     endwin(void);
bool    isendwin(void);
SCREEN *newterm(char *type, FILE *outfd, FILE *infd);
SCREEN *set_term(SCREEN *new);
void    delscreen(SCREEN* sp);
```

- Use initscr() to initialize the screen *(generating a WINDOW)*. It creates two window handles: stdscr and curscr.
- initscr() determines the terminal type and initializes all curses data structures
- A program that controls more than one terminal should use the newterm() routine instead of initscr() for each terminal *(generating a SCREEN)*
- A curses-using program should always call endwin() before exiting (restoring tty modes, moving the cursor to the lower left-hand corner, resetting the terminal to a non-visual state)
- Use isendwin() to determine if endwin() has already been called
- Use set_term() to switch between different terminals
- Use delscreen() to free storage associated with SCREEN data structures

Curses: Initialization and Termination

```
#include <curses.h>
WINDOW *initscr(void);
int     endwin(void);
bool    isendwin(void);
SCREEN *newterm(char *type, FILE *outfd, FILE *infd);
SCREEN *set_term(SCREEN *new);
void    delscreen(SCREEN* sp);
```

- Use initscr() to initialize the screen *(generating a WINDOW)*. It creates two window handles: stdscr and curscr.
- initscr() determines the terminal type and initializes all curses data structures
- A program that controls more than one terminal should use the newterm() routine instead of initscr() for each terminal *(generating a SCREEN)*
- A curses-using program should always call endwin() before exiting (restoring tty modes, moving the cursor to the lower left-hand corner, resetting the terminal to a non-visual state)
- Use isendwin() to determine if endwin() has already been called
- Use set_term() to switch between different terminals
- Use delscreen() to free storage associated with SCREEN data structures

See curs01.c, curs02.c

Curses: Typing Initialization, con’t.

int cbreak(void)	int nocbreak(void)	controls if chars are immediately available to pgm
int echo(void)	int noecho(void)	does getch() echo characters as they’re typed?
int raw(void)	int noraw(void)	like cbreak + ctrl chars passed along uninterpreted
int nodelay(WINDOW *,bool)		prevent/enable getch() from blocking

- Normally terminals buffer characters until a newline is typed

Curses: Typing Initialization, con't.

int cbreak(void)	int nocbreak(void)	controls if chars are immediately available to pgm
int echo(void)	int noecho(void)	does getch() echo characters as they're typed?
int raw(void)	int noraw(void)	like cbreak + ctrl chars passed along uninterpreted
int nodelay(WINDOW *,bool)		prevent/enable getch() from blocking

- Normally terminals buffer characters until a newline is typed
- cbreak() disables line buffering and erase/kill character processing, so typed characters are immediately available to the program

Curses: Typing Initialization, con’t.

int cbreak(void)	int nocbreak(void)	controls if chars are immediately available to pgm
int echo(void)	int noecho(void)	does getch() echo characters as they’re typed?
int raw(void)	int noraw(void)	like cbreak + ctrl chars passed along uninterpreted
int nodelay(WINDOW *,bool)		prevent/enable getch() from blocking

- Normally terminals buffer characters until a newline is typed
- cbreak() disables line buffering and erase/kill character processing, so typed characters are immediately available to the program
- echo() controls whether characters obtained by getch() are echo’d as they’re typed

Curses: Typing Initialization, con’t.

int cbreak(void)	int nocbreak(void)	controls if chars are immediately available to pgm
int echo(void)	int noecho(void)	does getch() echo characters as they’re typed?
int raw(void)	int noraw(void)	like cbreak + ctrl chars passed along uninterpreted
int nodelay(WINDOW *,bool)		prevent/enable getch() from blocking

- Normally terminals buffer characters until a newline is typed
- cbreak() disables line buffering and erase/kill character processing, so typed characters are immediately available to the program
- echo() controls whether characters obtained by getch() are echo’d as they’re typed
- use nodelay() to prevent getch() from blocking

Curses: Typing Initialization, con't.

int cbreak(void)	int nocbreak(void)	controls if chars are immediately available to pgm
int echo(void)	int noecho(void)	does getch() echo characters as they're typed?
int raw(void)	int noraw(void)	like cbreak + ctrl chars passed along uninterpreted
int nodelay(WINDOW *,bool)		prevent/enable getch() from blocking

- Normally terminals buffer characters until a newline is typed
- cbreak() disables line buffering and erase/kill character processing, so typed characters are immediately available to the program
- echo() controls whether characters obtained by getch() are echo'd as they're typed
- use nodelay() to prevent getch() from blocking

See curs10.c

Curses: Colors

```
# include <curses.h>
int start_color(void);
bool has_colors(void);
bool can_change_color(void);
```

- Use has_colors() to determine if the terminal in use supports colorization

Curses: Colors

```
# include <curses.h>
int start_color(void);
bool has_colors(void);
bool can_change_color(void);
```

- Use `has_colors()` to determine if the terminal in use supports colorization
- Use `can_change_color()` to determine if your program is allowed to change terminal color(s)

Curses: Colors

```
# include <curses.h>
int start_color(void);
bool has_colors(void);
bool can_change_color(void);
```

- Use has_colors() to determine if the terminal in use supports colorization
- Use can_change_color() to determine if your program is allowed to change terminal color(s)
- Use start_color() to initialize the colorization capability; it initializes eight basic colors (black, red, green, yellow, blue, magenta, cyan, and white)

Curses: Colors

```
# include <curses.h>
int start_color(void);
bool has_colors(void);
bool can_change_color(void);
```

- Use has_colors() to determine if the terminal in use supports colorization
- Use can_change_color() to determine if your program is allowed to change terminal color(s)
- Use start_color() to initialize the colorization capability; it initializes eight basic colors (black, red, green, yellow, blue, magenta, cyan, and white)
- COLORS is the maximum qty of colors the terminal supports

Curses: Colors

```
# include <curses.h>
int start_color(void);
bool has_colors(void);
bool can_change_color(void);
```

- Use has_colors() to determine if the terminal in use supports colorization
- Use can_change_color() to determine if your program is allowed to change terminal color(s)
- Use start_color() to initialize the colorization capability; it initializes eight basic colors (black, red, green, yellow, blue, magenta, cyan, and white)
- COLORS is the maximum qty of colors the terminal supports
- COLOR_PAIRS is the maximum qty of color-pairs (foreground, background) the terminal supports

Curses: Colors

# include <curses.h>
int start_color(void);
bool has_colors(void);
bool can_change_color(void);

- Use has_colors() to determine if the terminal in use supports colorization
- Use can_change_color() to determine if your program is allowed to change terminal color(s)
- Use start_color() to initialize the colorization capability; it initializes eight basic colors (black, red, green, yellow, blue, magenta, cyan, and white)
- COLORS is the maximum qty of colors the terminal supports
- COLOR_PAIRS is the maximum qty of color-pairs (foreground, background) the terminal supports

See curs06.c

Curses: Colors, con’t.

```
int init_pair(short pair, short f, short b);
int init_color(short color, short r, short g, short b);
int color_content(short color, short *r, short *g, short *b);
int pair_content(short pair, short *f, short *b);
```

- Color pairs consist of two integers (foreground, background)

Curses: Colors, con't.

```
int init_pair(short pair, short f, short b);
int init_color(short color, short r, short g, short b);
int color_content(short color, short *r, short *g, short *b);
int pair_content(short pair, short *f, short *b);
```

- Color pairs consist of two integers (foreground, background)
- init_pair(): The value of the first argument f must be between 1 and COLOR_PAIRS-1

Curses: Colors, con't.

```
int init_pair(short pair, short f, short b);
int init_color(short color, short r, short g, short b);
int color_content(short color, short *r, short *g, short *b);
int pair_content(short pair, short *f, short *b);
```

- Color pairs consist of two integers (foreground, background)
- init_pair(): The value of the first argument f must be between 1 and COLOR_PAIRS-1
- init_pair(): The value of the second argument b must be between 0 and COLORS

Curses: Colors, con't.

```
int init_pair(short pair, short f, short b);
int init_color(short color, short r, short g, short b);
int color_content(short color, short *r, short *g, short *b);
int pair_content(short pair, short *f, short *b);
```

- Color pairs consist of two integers (foreground, background)
- init_pair(): The value of the first argument f must be between 1 and COLOR_PAIRS-1
- init_pair(): The value of the second argument b must be between 0 and COLORS
- init_pair(): If a color-pair has already been initialized and used, the screen is refreshed and all usages of that color-pair are changed

Curses: Colors, con't.

int init_pair(short pair, short f, short b);
int init_color(short color, short r, short g, short b);
int color_content(short color, short *r, short *g, short *b);
int pair_content(short pair, short *f, short *b);

- Color pairs consist of two integers (foreground, background)
- init_pair(): The value of the first argument f must be between 1 and COLOR_PAIRS-1
- init_pair(): The value of the second argument b must be between 0 and COLORS
- init_pair(): If a color-pair has already been initialized and used, the screen is refreshed and all usages of that color-pair are changed
- init_color(): Allows one to specify a new color using the red-green-blue model with values between 0-1000

Curses: Colors, con’t.

```
int init_pair(short pair, short f, short b);
int init_color(short color, short r, short g, short b);
int color_content(short color, short *r, short *g, short *b);
int pair_content(short pair, short *f, short *b);
```

- Color pairs consist of two integers (foreground, background)
- init_pair(): The value of the first argument f must be between 1 and COLOR_PAIRS-1
- init_pair(): The value of the second argument b must be between 0 and COLORS
- init_pair(): If a color-pair has already been initialized and used, the screen is refreshed and all usages of that color-pair are changed
- init_color(): Allows one to specify a new color using the red-green-blue model with values between 0-1000
- init_color(): All usages of the color are immediately changed on the screen

Curses: Colors, con't.

```
int init_pair(short pair, short f, short b);
int init_color(short color, short r, short g, short b);
int color_content(short color, short *r, short *g, short *b);
int pair_content(short pair, short *f, short *b);
```

- Color pairs consist of two integers (foreground, background)
- init_pair(): The value of the first argument f must be between 1 and COLOR_PAIRS-1
- init_pair(): The value of the second argument b must be between 0 and COLORS
- init_pair(): If a color-pair has already been initialized and used, the screen is refreshed and all usages of that color-pair are changed
- init_color(): Allows one to specify a new color using the red-green-blue model with values between 0-1000
- init_color(): All usages of the color are immediately changed on the screen
- color_content(): allow programmer to query what red-green-blue values are currently assigned to color

Curses: Colors, con't.

```
int init_pair(short pair, short f, short b);
int init_color(short color, short r, short g, short b);
int color_content(short color, short *r, short *g, short *b);
int pair_content(short pair, short *f, short *b);
```

- Color pairs consist of two integers (foreground, background)
- init_pair(): The value of the first argument f must be between 1 and COLOR_PAIRS-1
- init_pair(): The value of the second argument b must be between 0 and COLORS
- init_pair(): If a color-pair has already been initialized and used, the screen is refreshed and all usages of that color-pair are changed
- init_color(): Allows one to specify a new color using the red-green-blue model with values between 0-1000
- init_color(): All usages of the color are immediately changed on the screen
- color_content(): allow programmer to query what red-green-blue values are currently assigned to color
- pair_content(): allow programmer to query which colors pair refers

Curses: Colors, con't.

```
int init_pair(short pair, short f, short b);
int init_color(short color, short r, short g, short b);
int color_content(short color, short *r, short *g, short *b);
int pair_content(short pair, short *f, short *b);
```

- Color pairs consist of two integers (foreground, background)
- init_pair(): The value of the first argument f must be between 1 and COLOR_PAIRS-1
- init_pair(): The value of the second argument b must be between 0 and COLORS
- init_pair(): If a color-pair has already been initialized and used, the screen is refreshed and all usages of that color-pair are changed
- init_color(): Allows one to specify a new color using the red-green-blue model with values between 0-1000
- init_color(): All usages of the color are immediately changed on the screen
- color_content(): allow programmer to query what red-green-blue values are currently assigned to color
- pair_content(): allow programmer to query which colors pair refers
- pre-defined: COLOR_BLACK, COLOR_RED, ..., COLOR_WHITE

Curses: Colors, con’t.

```
int init_pair(short pair, short f, short b);
int init_color(short color, short r, short g, short b);
int color_content(short color, short *r, short *g, short *b);
int pair_content(short pair, short *f, short *b);
```

- Color pairs consist of two integers (foreground, background)
- init_pair(): The value of the first argument f must be between 1 and COLOR_PAIRS-1
- init_pair(): The value of the second argument b must be between 0 and COLORS
- init_pair(): If a color-pair has already been initialized and used, the screen is refreshed and all usages of that color-pair are changed
- init_color(): Allows one to specify a new color using the red-green-blue model with values between 0-1000
- init_color(): All usages of the color are immediately changed on the screen
- color_content(): allow programmer to query what red-green-blue values are currently assigned to color
- pair_content(): allow programmer to query which colors pair refers
- pre-defined: COLOR_BLACK, COLOR_RED, ..., COLOR_WHITE

See curs07.c, curs09.c

Curses: Attributes

```
int attrset(int attrs);
int wattrset(WINDOW *win, int attrs);
int color_set(short color_pair_number, void* opts);
int wcolor_set(WINDOW *win, short color_pair_number, void* opts);
```

- These routines set the attribute of subsequent characters.

Curses: Attributes

```
int attrset(int attrs);
int wattrset(WINDOW *win, int attrs);
int color_set(short color_pair_number, void* opts);
int wcolor_set(WINDOW *win, short color_pair_number, void* opts);
```

- These routines set the attribute of subsequent characters.
- Attributes are properties of the characters, and will move with the character during scrolling, insertion, and deletion

Curses: Attributes

int attrset(int attrs);
int wattrset(WINDOW *win, int attrs);
int color_set(short color_pair_number, void* opts);
int wcolor_set(WINDOW *win, short color_pair_number, void* opts);

- These routines set the attribute of subsequent characters.
- Attributes are properties of the characters, and will move with the character during scrolling, insertion, and deletion
- attrset(), wattrset(): sets the current attributes of the given window to attrs

Curses: Attributes

```
int attrset(int attrs);
int wattrset(WINDOW *win, int attrs);
int color_set(short color_pair_number, void* opts);
int wcolor_set(WINDOW *win, short color_pair_number, void* opts);
```

- These routines set the attribute of subsequent characters.
- Attributes are properties of the characters, and will move with the character during scrolling, insertion, and deletion
- attrset(), wattrset(): sets the current attributes of the given window to attrs

A_NORMAL	Normal display (no highlight)

Curses: Attributes

```
int attrset(int attrs);
int wattrset(WINDOW *win, int attrs);
int color_set(short color_pair_number, void* opts);
int wcolor_set(WINDOW *win, short color_pair_number, void* opts);
```

- These routines set the attribute of subsequent characters.
- Attributes are properties of the characters, and will move with the character during scrolling, insertion, and deletion
- attrset(), wattrset(): sets the current attributes of the given window to attrs

A_NORMAL	Normal display (no highlight)
A_STANDOUT	Best highlighting mode of the terminal.

Curses: Attributes

```
int attrset(int attrs);
int wattrset(WINDOW *win, int attrs);
int color_set(short color_pair_number, void* opts);
int wcolor_set(WINDOW *win, short color_pair_number, void* opts);
```

- These routines set the attribute of subsequent characters.
- Attributes are properties of the characters, and will move with the character during scrolling, insertion, and deletion
- attrset(), wattrset(): sets the current attributes of the given window to attrs

A_NORMAL	Normal display (no highlight)
A_STANDOUT	Best highlighting mode of the terminal.
A_UNDERLINE	Underlining

Curses: Attributes

```
int attrset(int attrs);
int wattrset(WINDOW *win, int attrs);
int color_set(short color_pair_number, void* opts);
int wcolor_set(WINDOW *win, short color_pair_number, void* opts);
```

- These routines set the attribute of subsequent characters.
- Attributes are properties of the characters, and will move with the character during scrolling, insertion, and deletion
- attrset(), wattrset(): sets the current attributes of the given window to attrs

A_NORMAL	Normal display (no highlight)
A_STANDOUT	Best highlighting mode of the terminal.
A_UNDERLINE	Underlining
A_REVERSE	Reverse video

Curses: Attributes

```
int attrset(int attrs);
int wattrset(WINDOW *win, int attrs);
int color_set(short color_pair_number, void* opts);
int wcolor_set(WINDOW *win, short color_pair_number, void* opts);
```

- These routines set the attribute of subsequent characters.
- Attributes are properties of the characters, and will move with the character during scrolling, insertion, and deletion
- attrset(), wattrset(): sets the current attributes of the given window to attrs

A_NORMAL	Normal display (no highlight)
A_STANDOUT	Best highlighting mode of the terminal.
A_UNDERLINE	Underlining
A_REVERSE	Reverse video
A_BLINK	Blinking

Curses: Attributes

```
int attrset(int attrs);
int wattrset(WINDOW *win, int attrs);
int color_set(short color_pair_number, void* opts);
int wcolor_set(WINDOW *win, short color_pair_number, void* opts);
```

- These routines set the attribute of subsequent characters.
- Attributes are properties of the characters, and will move with the character during scrolling, insertion, and deletion
- attrset(), wattrset(): sets the current attributes of the given window to attrs

A_NORMAL	Normal display (no highlight)
A_STANDOUT	Best highlighting mode of the terminal.
A_UNDERLINE	Underlining
A_REVERSE	Reverse video
A_BLINK	Blinking
A_DIM	Half bright

Curses: Attributes

```
int attrset(int attrs);
int wattrset(WINDOW *win, int attrs);
int color_set(short color_pair_number, void* opts);
int wcolor_set(WINDOW *win, short color_pair_number, void* opts);
```

- These routines set the attribute of subsequent characters.
- Attributes are properties of the characters, and will move with the character during scrolling, insertion, and deletion
- attrset(), wattrset(): sets the current attributes of the given window to attrs

A_NORMAL	Normal display (no highlight)
A_STANDOUT	Best highlighting mode of the terminal.
A_UNDERLINE	Underlining
A_REVERSE	Reverse video
A_BLINK	Blinking
A_DIM	Half bright
A_BOLD	Extra bright or bold

Curses: Attributes

int attrset(int attrs);
int wattrset(WINDOW *win, int attrs);
int color_set(short color_pair_number, void* opts);
int wcolor_set(WINDOW *win, short color_pair_number, void* opts);

- These routines set the attribute of subsequent characters.
- Attributes are properties of the characters, and will move with the character during scrolling, insertion, and deletion
- attrset(), wattrset(): sets the current attributes of the given window to attrs

A_NORMAL	Normal display (no highlight)
A_STANDOUT	Best highlighting mode of the terminal.
A_UNDERLINE	Underlining
A_REVERSE	Reverse video
A_BLINK	Blinking
A_DIM	Half bright
A_BOLD	Extra bright or bold
A_PROTECT	Protected mode

Curses: Attributes

```
int attrset(int attrs);
int wattrset(WINDOW *win, int attrs);
int color_set(short color_pair_number, void* opts);
int wcolor_set(WINDOW *win, short color_pair_number, void* opts);
```

- These routines set the attribute of subsequent characters.
- Attributes are properties of the characters, and will move with the character during scrolling, insertion, and deletion
- attrset(), wattrset(): sets the current attributes of the given window to attrs

A_NORMAL	Normal display (no highlight)
A_STANDOUT	Best highlighting mode of the terminal.
A_UNDERLINE	Underlining
A_REVERSE	Reverse video
A_BLINK	Blinking
A_DIM	Half bright
A_BOLD	Extra bright or bold
A_PROTECT	Protected mode
A_INVIS	Invisible or blank mode

Curses: Attributes

```
int attrset(int attrs);
int wattrset(WINDOW *win, int attrs);
int color_set(short color_pair_number, void* opts);
int wcolor_set(WINDOW *win, short color_pair_number, void* opts);
```

- These routines set the attribute of subsequent characters.
- Attributes are properties of the characters, and will move with the character during scrolling, insertion, and deletion
- attrset(), wattrset(): sets the current attributes of the given window to attrs

A_NORMAL	Normal display (no highlight)
A_STANDOUT	Best highlighting mode of the terminal.
A_UNDERLINE	Underlining
A_REVERSE	Reverse video
A_BLINK	Blinking
A_DIM	Half bright
A_BOLD	Extra bright or bold
A_PROTECT	Protected mode
A_INVIS	Invisible or blank mode
A_ALTCHARSET	Alternate character set

Curses: Attributes

```
int attrset(int attrs);
int wattrset(WINDOW *win, int attrs);
int color_set(short color_pair_number, void* opts);
int wcolor_set(WINDOW *win, short color_pair_number, void* opts);
```

- These routines set the attribute of subsequent characters.
- Attributes are properties of the characters, and will move with the character during scrolling, insertion, and deletion
- attrset(), wattrset(): sets the current attributes of the given window to attrs

A_NORMAL	Normal display (no highlight)
A_STANDOUT	Best highlighting mode of the terminal.
A_UNDERLINE	Underlining
A_REVERSE	Reverse video
A_BLINK	Blinking
A_DIM	Half bright
A_BOLD	Extra bright or bold
A_PROTECT	Protected mode
A_INVIS	Invisible or blank mode
A_ALTCHARSET	Alternate character set
COLOR_PAIR(n)	Color-pair number n

Curses: Erasing and Clearing

int erase(void)	int werase(WINDOW *win)	copy blanks to every position in screen/window
int clear(void)	int wclear(WINDOW *win)	like erase(), but also calls clearok()
int clrtobot(void)	int wclrtobot(WINDOW *win)	clears to current line to bottom of screen/window
int clrtoeol(void)	int wclrtoeol(WINDOW *win)	clears to end-of-line
int clearok(WINDOW *win, bool bf)		clear screen at next refresh, but don’t reset window

- All routines return the integer OK on success and ERR on failure

Curses: Erasing and Clearing

int erase(void)	int werase(WINDOW *win)	copy blanks to every position in screen/window
int clear(void)	int wclear(WINDOW *win)	like erase(), but also calls clearok()
int clrtobot(void)	int wclrtobot(WINDOW *win)	clears to current line to bottom of screen/window
int clrtoeol(void)	int wclrtoeol(WINDOW *win)	clears to end-of-line
int clearok(WINDOW *win, bool bf)		clear screen at next refresh, but don’t reset window

- All routines return the integer OK on success and ERR on failure
- These functions may actually be macros

Curses: Erasing and Clearing

int erase(void)	**int werase(WINDOW *win)**	**copy blanks to every position in screen/window**
int clear(void)	**int wclear(WINDOW *win)**	**like erase(), but also calls clearok()**
int clrtobot(void)	**int wclrtobot(WINDOW *win)**	**clears to current line to bottom of screen/window**
int clrtoeol(void)	**int wclrtoeol(WINDOW *win)**	**clears to end-of-line**
int clearok(WINDOW *win, bool bf)		**clear screen at next refresh, but don’t reset window**

- All routines return the integer OK on success and ERR on failure
- These functions may actually be macros
- Most implementations clear the screen after wclear(), even if done only for a subwindow or window

Curses: Erasing and Clearing

int erase(void)	int werase(WINDOW *win)	copy blanks to every position in screen/window
int clear(void)	int wclear(WINDOW *win)	like erase(), but also calls clearok()
int clrtobot(void)	int wclrtobot(WINDOW *win)	clears to current line to bottom of screen/window
int clrtoeol(void)	int wclrtoeol(WINDOW *win)	clears to end-of-line
int clearok(WINDOW *win, bool bf)		clear screen at next refresh, but don’t reset window

- All routines return the integer OK on success and ERR on failure
- These functions may actually be macros
- Most implementations clear the screen after wclear(), even if done only for a subwindow or window
- If you don’t want to clear the screen on the next wrefresh(), use werase() instead

Curses: Erasing and Clearing

int erase(void)	int werase(WINDOW *win)	copy blanks to every position in screen/window
int clear(void)	int wclear(WINDOW *win)	like erase(), but also calls clearok()
int clrtobot(void)	int wclrtobot(WINDOW *win)	clears to current line to bottom of screen/window
int clrtoeol(void)	int wclrtoeol(WINDOW *win)	clears to end-of-line
int clearok(WINDOW *win, bool bf)		clear screen at next refresh, but don’t reset window

- All routines return the integer OK on success and ERR on failure
- These functions may actually be macros
- Most implementations clear the screen after wclear(), even if done only for a subwindow or window
- If you don’t want to clear the screen on the next wrefresh(), use werase() instead
- erase() and clear() are easy to confuse, because there are two ways to clear the screen:

  fill the screen with blanks or use the hardware/driver clear control code.

Curses: Erasing and Clearing

int erase(void)	**int werase(WINDOW *win)**	**copy blanks to every position in screen/window**
int clear(void)	**int wclear(WINDOW *win)**	**like erase(), but also calls clearok()**
int clrtobot(void)	**int wclrtobot(WINDOW *win)**	**clears to current line to bottom of screen/window**
int clrtoeol(void)	**int wclrtoeol(WINDOW *win)**	**clears to end-of-line**
int clearok(WINDOW *win, bool bf)		**clear screen at next refresh, but don’t reset window**

- All routines return the integer OK on success and ERR on failure
- These functions may actually be macros
- Most implementations clear the screen after wclear(), even if done only for a subwindow or window
- If you don’t want to clear the screen on the next wrefresh(), use werase() instead
- erase() and clear() are easy to confuse, because there are two ways to clear the screen:

  fill the screen with blanks or use the hardware/driver clear control code.

  erase() fill screen with blanks

Curses: Erasing and Clearing

int erase(void)	int werase(WINDOW *win)	copy blanks to every position in screen/window
int clear(void)	int wclear(WINDOW *win)	like erase(), but also calls clearok()
int clrtobot(void)	int wclrtobot(WINDOW *win)	clears to current line to bottom of screen/window
int clrtoeol(void)	int wclrtoeol(WINDOW *win)	clears to end-of-line
int clearok(WINDOW *win, bool bf)		clear screen at next refresh, but don't reset window

- All routines return the integer OK on success and ERR on failure
- These functions may actually be macros
- Most implementations clear the screen after wclear(), even if done only for a subwindow or window
- If you don't want to clear the screen on the next wrefresh(), use werase() instead
- erase() and clear() are easy to confuse, because there are two ways to clear the screen:

  fill the screen with blanks or use the hardware/driver clear control code.

erase()	fill screen with blanks
clearok()	sets _clear to the given bf

Curses: Erasing and Clearing

int erase(void)	**int werase(WINDOW *win)**	**copy blanks to every position in screen/window**
int clear(void)	**int wclear(WINDOW *win)**	**like erase(), but also calls clearok()**
int clrtobot(void)	**int wclrtobot(WINDOW *win)**	**clears to current line to bottom of screen/window**
int clrtoeol(void)	**int wclrtoeol(WINDOW *win)**	**clears to end-of-line**
int clearok(WINDOW *win, bool bf)		**clear screen at next refresh, but don't reset window**

- All routines return the integer OK on success and ERR on failure
- These functions may actually be macros
- Most implementations clear the screen after wclear(), even if done only for a subwindow or window
- If you don't want to clear the screen on the next wrefresh(), use werase() instead
- erase() and clear() are easy to confuse, because there are two ways to clear the screen:

  fill the screen with blanks or use the hardware/driver clear control code.

erase()	fill screen with blanks
clearok()	sets _clear to the given bf
clear()	calls erase() and clearok() with bf=1

Curses: Erasing and Clearing

int erase(void)	int werase(WINDOW *win)	copy blanks to every position in screen/window
int clear(void)	int wclear(WINDOW *win)	like erase(), but also calls clearok()
int clrtobot(void)	int wclrtobot(WINDOW *win)	clears to current line to bottom of screen/window
int clrtoeol(void)	int wclrtoeol(WINDOW *win)	clears to end-of-line
int clearok(WINDOW *win, bool bf)		clear screen at next refresh, but don’t reset window

- All routines return the integer OK on success and ERR on failure
- These functions may actually be macros
- Most implementations clear the screen after wclear(), even if done only for a subwindow or window
- If you don’t want to clear the screen on the next wrefresh(), use werase() instead
- erase() and clear() are easy to confuse, because there are two ways to clear the screen:

  fill the screen with blanks or use the hardware/driver clear control code.

erase()	fill screen with blanks
clearok()	sets _clear to the given bf
clear()	calls erase() and clearok() with bf=1 (ie. uses hardware clear control code)

Curses: Output Options

int clearok(WINDOW *win, bool bf);
int idlok(WINDOW *win, bool bf);
void idcok(WINDOW *win, bool bf);
void immedok(WINDOW *win, bool bf);

- clearok(): with bf=TRUE, the next call to wrefresh() will clear the screen completely, redrawing the entire screen.

Curses: Output Options

```
int clearok(WINDOW *win, bool bf);
int idlok(WINDOW *win, bool bf);
void idcok(WINDOW *win, bool bf);
void immedok(WINDOW *win, bool bf);
```

- clearok(): with bf=TRUE, the next call to wrefresh() will clear the screen completely, redrawing the entire screen.
- idlok(): with bf=TRUE, assumes that the hardware supports insert/delete and will use it.

  With bf=FALSE, line insertion/deletion is unavailable, and so curses will redraw any changed lines

Curses: Output Options

```
int clearok(WINDOW *win, bool bf);
int idlok(WINDOW *win, bool bf);
void idcok(WINDOW *win, bool bf);
void immedok(WINDOW *win, bool bf);
```

- clearok(): with bf=TRUE, the next call to wrefresh() will clear the screen completely, redrawing the entire screen.
- idlok(): with bf=TRUE, assumes that the hardware supports insert/delete and will use it.

  With bf=FALSE, line insertion/deletion is unavailable, and so curses will redraw any changed lines
- idcok(): with bf=TRUE curses will re-enable consideration of using hardware character-oriented insert/delete. With bf=FALSE curses will not consider using hardware character-oriented insert/delete

Curses: Output Options

int clearok(WINDOW *win, bool bf);
int idlok(WINDOW *win, bool bf);
void idcok(WINDOW *win, bool bf);
void immedok(WINDOW *win, bool bf);

- clearok(): with bf=TRUE, the next call to wrefresh() will clear the screen completely, redrawing the entire screen.
- idlok(): with bf=TRUE, assumes that the hardware supports insert/delete and will use it.

  With bf=FALSE, line insertion/deletion is unavailable, and so curses will redraw any changed lines
- idcok(): with bf=TRUE curses will re-enable consideration of using hardware character-oriented insert/delete. With bf=FALSE curses will not consider using hardware character-oriented insert/delete
- immedok(): if true, any change to the window image (ie. waddch(), wclrtobot(), wscrl(), etc) automatically causes wrefresh().

Curses: Scrolling and Return Key

```
int leaveok(WINDOW *win, bool bf);
int setscrreg(int top, int bot);
int wsetscrreg(WINDOW *win, int top, int bot);
int scrollok(WINDOW *win, bool bf);
int nl(void);
int nonl(void);
```

- leavok(): normally the cursor is left at the window's cursor; if true, then the cursor will be left wherever

Curses: Scrolling and Return Key

```
int leaveok(WINDOW *win, bool bf);
int setscrreg(int top, int bot);
int wsetscrreg(WINDOW *win, int top, int bot);
int scrollok(WINDOW *win, bool bf);
int nl(void);
int nonl(void);
```

- leavok(): normally the cursor is left at the window’s cursor; if true, then the cursor will be left wherever
- setscrreg() and wsetscrreg(): allows one to set a screen/window region to be scrollable. You’ll want to have idlok() true, too.

Curses: Scrolling and Return Key

```
int leaveok(WINDOW *win, bool bf);
int setscrreg(int top, int bot);
int wsetscrreg(WINDOW *win, int top, int bot);
int scrollok(WINDOW *win, bool bf);
int nl(void);
int nonl(void);
```

- leavok(): normally the cursor is left at the window's cursor; if true, then the cursor will be left wherever
- setscrreg() and wsetscrreg(): allows one to set a screen/window region to be scrollable. You'll want to have idlok() true, too.
- scrollok(): when the cursor of a window is moved off the edge of a window or scrolling region: bf=TRUE: screen/window will scroll (you'll want idlok() true) bf=FALSE: cursor will be left on last line

Curses: Scrolling and Return Key

```
int leaveok(WINDOW *win, bool bf);
int setscrreg(int top, int bot);
int wsetscrreg(WINDOW *win, int top, int bot);
int scrollok(WINDOW *win, bool bf);
int nl(void);
int nonl(void);
```

- leavok(): normally the cursor is left at the window's cursor; if true, then the cursor will be left wherever
- setscrreg() and wsetscrreg(): allows one to set a screen/window region to be scrollable. You'll want to have idlok() true, too.
- scrollok(): when the cursor of a window is moved off the edge of a window or scrolling region: bf=TRUE: screen/window will scroll (you'll want idlok() true) bf=FALSE: cursor will be left on last line
- nl() and nonl(): control whether the terminal will translate the return key into newline + linefeed. nonl() provides faster cursor motion.

Curses: Refreshing and Updating

```
int refresh(void);
int wrefresh(WINDOW *win);
int wnoutrefresh(WINDOW *win);
int doupdate(void);
int redrawwin(WINDOW *win);
int wredrawln(WINDOW *win, int beg_line, int num_lines);
```

- refresh(), wrefresh(), wnoutrefresh(), and doupdate() must be called to get the display to change

Curses: Refreshing and Updating

```
int refresh(void);
int wrefresh(WINDOW *win);
int wnoutrefresh(WINDOW *win);
int doupdate(void);
int redrawwin(WINDOW *win);
int wredrawln(WINDOW *win, int beg_line, int num_lines);
```

- refresh(), wrefresh(), wnoutrefresh(), and doupdate() must be called to get the display to change
- Other routines merely manipulate *stdscr*

Curses: Refreshing and Updating

```
int refresh(void);
int wrefresh(WINDOW *win);
int wnoutrefresh(WINDOW *win);
int doupdate(void);
int redrawwin(WINDOW *win);
int wredrawln(WINDOW *win, int beg_line, int num_lines);
```

- refresh(), wrefresh(), wnoutrefresh(), and doupdate() must be called to get the display to change
- Other routines merely manipulate *stdscr*
- wnoutrefresh() and doupdate() allow multiple efficient updating

Curses: Refreshing and Updating

```
int refresh(void);
int wrefresh(WINDOW *win);
int wnoutrefresh(WINDOW *win);
int doupdate(void);
int redrawwin(WINDOW *win);
int wredrawln(WINDOW *win, int beg_line, int num_lines);
```

- refresh(), wrefresh(), wnoutrefresh(), and doupdate() must be called to get the display to change
- Other routines merely manipulate *stdscr*
- wnoutrefresh() and doupdate() allow multiple efficient updating
- wnoutrefresh() copies the named window to the virtual screen and then calls doupdate()

Curses: Refreshing and Updating

```
int refresh(void);
int wrefresh(WINDOW *win);
int wnoutrefresh(WINDOW *win);
int doupdate(void);
int redrawwin(WINDOW *win);
int wredrawln(WINDOW *win, int beg_line, int num_lines);
```

- refresh(), wrefresh(), wnoutrefresh(), and doupdate() must be called to get the display to change
- Other routines merely manipulate *stdscr*
- wnoutrefresh() and doupdate() allow multiple efficient updating
- wnoutrefresh() copies the named window to the virtual screen and then calls doupdate()
- doupdate() compares the virtual screen to the physical screen

Curses: Refreshing and Updating

```
int refresh(void);
int wrefresh(WINDOW *win);
int wnoutrefresh(WINDOW *win);
int doupdate(void);
int redrawwin(WINDOW *win);
int wredrawln(WINDOW *win, int beg_line, int num_lines);
```

- refresh(), wrefresh(), wnoutrefresh(), and doupdate() must be called to get the display to change
- Other routines merely manipulate *stdscr*
- wnoutrefresh() and doupdate() allow multiple efficient updating
- wnoutrefresh() copies the named window to the virtual screen and then calls doupdate()
- doupdate() compares the virtual screen to the physical screen
- One may call wnoutrefresh() for multiple windows, and then call doupdate() once

Curses: Refreshing and Updating

```
int refresh(void);
int wrefresh(WINDOW *win);
int wnoutrefresh(WINDOW *win);
int doupdate(void);
int redrawwin(WINDOW *win);
int wredrawln(WINDOW *win, int beg_line, int num_lines);
```

- refresh(), wrefresh(), wnoutrefresh(), and doupdate() must be called to get the display to change
- Other routines merely manipulate *stdscr*
- wnoutrefresh() and doupdate() allow multiple efficient updating
- wnoutrefresh() copies the named window to the virtual screen and then calls doupdate()
- doupdate() compares the virtual screen to the physical screen
- One may call wnoutrefresh() for multiple windows, and then call doupdate() once
- wredrawln() indicates that lines from beg_line to beg_line+num_lines-1 are corrupt and should be thrown away prior to writing over them.

Curses: Refreshing and Updating

```
int refresh(void);
int wrefresh(WINDOW *win);
int wnoutrefresh(WINDOW *win);
int doupdate(void);
int redrawwin(WINDOW *win);
int wredrawln(WINDOW *win, int beg_line, int num_lines);
```

- refresh(), wrefresh(), wnoutrefresh(), and doupdate() must be called to get the display to change
- Other routines merely manipulate *stdscr*
- wnoutrefresh() and doupdate() allow multiple efficient updating
- wnoutrefresh() copies the named window to the virtual screen and then calls doupdate()
- doupdate() compares the virtual screen to the physical screen
- One may call wnoutrefresh() for multiple windows, and then call doupdate() once
- wredrawln() indicates that lines from beg_line to beg_line+num_lines-1 are corrupt and should be thrown away prior to writing over them.
- redrawwin() touches the entire window

Curses: Typical Window Updating Procedure

- Initialize the WINDOW to represent some screen real-estate

Curses: Typical Window Updating Procedure

- Initialize the WINDOW to represent some screen real-estate
- Insert and move characters, change colors, in the character array

Curses: Typical Window Updating Procedure

- Initialize the WINDOW to represent some screen real-estate
- Insert and move characters, change colors, in the character array
- Call refresh() to push changes in stdscr to curscr and on to the display

Curses: Typical Window Updating Procedure

- Initialize the WINDOW to represent some screen real-estate
- Insert and move characters, change colors, in the character array
- Call refresh() to push changes in stdscr to curscr and on to the display
- Make more changes to the window

Curses: Typical Window Updating Procedure

- Initialize the WINDOW to represent some screen real-estate
- Insert and move characters, change colors, in the character array
- Call refresh() to push changes in stdscr to curscr and on to the display
- Make more changes to the window
- Refresh again, repeat as required

Curses: WINDOWs

```
struct {
    short_cury,_curx;   pos'n of logical cursor in window
    short_maxy,_maxx;   height and width of window
    short_begy,_begx;   pos'n of upper left corner rltv to screen
    short_flags;        see next slide
    bool _clear;        set by clearok(),clear() if screen should be cleared
                        by the terminal clear ctrl code, but only if the
                        window occupies the entire screen.
    bool _leave;        (leaveok()) cursor left at last chgd char
    bool _scroll;       (scrollok()) enables logical scrolling
    char **_y;          pointer to char array holding screen image
    short*_firstch;     an array of indices to leftmost changed char in each line
    short*_lastch;      an array of indices to rightmost changed char in each line
}
```

Curses: WINDOW’s _flags

Hex	*Attribute*	*Description*
1	_SUBWIN	window is a subwindow

Curses: WINDOW's _flags

Hex	*Attribute*	*Description*
1	_SUBWIN	window is a subwindow
2	_ENDLINE	right end of each line is at edge of screen

Curses: WINDOW's _flags

Hex	*Attribute*	*Description*
1	_SUBWIN	window is a subwindow
2	_ENDLINE	right end of each line is at edge of screen
4	_FULLWIN	window fills whole screen

Curses: WINDOW’s _flags

Hex	*Attribute*	*Description*
1	_SUBWIN	window is a subwindow
2	_ENDLINE	right end of each line is at edge of screen
4	_FULLWIN	window fills whole screen
8	_SCROLLWIN	last character of the window is on the lower right edge of screen. The terminal should scroll if a character is put there

Curses: WINDOW’s _flags

Hex	*Attribute*	*Description*
1	_SUBWIN	window is a subwindow
2	_ENDLINE	right end of each line is at edge of screen
4	_FULLWIN	window fills whole screen
8	_SCROLLWIN	last character of the window is on the lower right edge of screen. The terminal should scroll if a character is put there
10	_FLUSH	reserved

Curses: WINDOW’s _flags

Hex	*Attribute*	*Description*
1	_SUBWIN	window is a subwindow
2	_ENDLINE	right end of each line is at edge of screen
4	_FULLWIN	window fills whole screen
8	_SCROLLWIN	last character of the window is on the lower right edge of screen. The terminal should scroll if a character is put there
10	_FLUSH	reserved
20	_FULLINE	each line of window reaches across the width of the screen

Curses: WINDOW’s _flags

Hex	*Attribute*	*Description*
1	_SUBWIN	window is a subwindow
2	_ENDLINE	right end of each line is at edge of screen
4	_FULLWIN	window fills whole screen
8	_SCROLLWIN	last character of the window is on the lower right edge of screen. The terminal should scroll if a character is put there
10	_FLUSH	reserved
20	_FULLINE	each line of window reaches across the width of the screen
40	_STANDOUT	inserted chars should be displayed in standout mode

Curses: WINDOW’s _flags

Hex	*Attribute*	*Description*
1	_SUBWIN	window is a subwindow
2	_ENDLINE	right end of each line is at edge of screen
4	_FULLWIN	window fills whole screen
8	_SCROLLWIN	last character of the window is on the lower right edge of screen. The terminal should scroll if a character is put there
10	_FLUSH	reserved
20	_FULLINE	each line of window reaches across the width of the screen
40	_STANDOUT	inserted chars should be displayed in standout mode
80	_INSL	if a line has been inserted

Curses: WINDOW's _flags

Hex	*Attribute*	*Description*
1	_SUBWIN	window is a subwindow
2	_ENDLINE	right end of each line is at edge of screen
4	_FULLWIN	window fills whole screen
8	_SCROLLWIN	last character of the window is on the lower right edge of screen. The terminal should scroll if a character is put there
10	_FLUSH	reserved
20	_FULLINE	each line of window reaches across the width of the screen
40	_STANDOUT	inserted chars should be displayed in standout mode
80	_INSL	if a line has been inserted
100	_DELL	if a line has been deleted

Curses: Creating and Removing Windows

```
WINDOW *newwin(int nlines, int ncols, int begin_y, int begin_x);
int delwin(WINDOW *win);
WINDOW *subwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);
```

- new windows: are independent windows

Curses: Creating and Removing Windows

```
WINDOW *newwin(int nlines, int ncols, int begin_y, int begin_x);
int delwin(WINDOW *win);
WINDOW *subwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);
```

- new windows: are independent windows
- subwindows: share screen image array of parent window

Curses: Creating and Removing Windows

WINDOW *newwin(int nlines, int ncols, int begin_y, int begin_x);
int delwin(WINDOW *win);
WINDOW *subwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);

- new windows: are independent windows
- subwindows: share screen image array of parent window
- newwin(): creates a new window. It may be as large as the screen or as small as 1x1 character.

  The window's image on the screen is initialized to blanks.

  Check for NULL return because the internal allocation may fail!

Curses: Creating and Removing Windows

WINDOW *newwin(int nlines, int ncols, int begin_y, int begin_x);
int delwin(WINDOW *win);
WINDOW *subwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);

- new windows: are independent windows
- subwindows: share screen image array of parent window
- newwin(): creates a new window. It may be as large as the screen or as small as 1x1 character.

  The window's image on the screen is initialized to blanks.

  Check for NULL return because the internal allocation may fail!
- subwin(): creates a subwindow, with orig as the parent window.

  begin_y,begin_x are relative to the screen orig (not the window!)

  A subwindow does not retain a pointer to its parent window (or vice versa)

  Don't delete a parent window and continue using a subwindow!

  Subwindows may themselves have subwindows.

Curses: Creating and Removing Windows

```
WINDOW *newwin(int nlines, int ncols, int begin_y, int begin_x);
int delwin(WINDOW *win);
WINDOW *subwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);
```

- new windows: are independent windows
- subwindows: share screen image array of parent window
- newwin(): creates a new window. It may be as large as the screen or as small as 1x1 character.

  The window's image on the screen is initialized to blanks.

  Check for NULL return because the internal allocation may fail!
- subwin(): creates a subwindow, with orig as the parent window.

  begin_y,begin_x are relative to the screen orig (not the window!)

  A subwindow does not retain a pointer to its parent window (or vice versa)

  Don't delete a parent window and continue using a subwindow!

  Subwindows may themselves have subwindows.
- delwin(): deletes the specified window.

  Won't free the screen array unless its not a subwindow.

  Does not automatically delete any subwindows the window may have!

Curses: Creating and Removing Windows

WINDOW *newwin(int nlines, int ncols, int begin_y, int begin_x);
int delwin(WINDOW *win);
WINDOW *subwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);

- new windows: are independent windows
- subwindows: share screen image array of parent window
- newwin(): creates a new window. It may be as large as the screen or as small as 1x1 character.

  The window's image on the screen is initialized to blanks.

  Check for NULL return because the internal allocation may fail!
- subwin(): creates a subwindow, with orig as the parent window.

  begin_y,begin_x are relative to the screen orig (not the window!)

  A subwindow does not retain a pointer to its parent window (or vice versa)

  Don't delete a parent window and continue using a subwindow!

  Subwindows may themselves have subwindows.
- delwin(): deletes the specified window.

  Won't free the screen array unless its not a subwindow.

  Does not automatically delete any subwindows the window may have!

See curs04.c

Curses: Creating and Removing Windows

```
WINDOW *derwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);
int touchwin(WINDOW *win);
int touchline(WINDOW *win, int start, int count);
int untouchwin(WINDOW *win);
```

- derwin(): is just like subwin(), except that begin_y,begin_x are window-relative (not screen relative)

Curses: Creating and Removing Windows

WINDOW *derwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);
int touchwin(WINDOW *win);
int touchline(WINDOW *win, int start, int count);
int untouchwin(WINDOW *win);

- derwin(): is just like subwin(), except that begin_y,begin_x are window-relative (not screen relative)
- subwin() and derwin(): creates and returns a pointer to a WINDOW with the specified qty of nlines lines and ncol columns.

Curses: Creating and Removing Windows

```
WINDOW *derwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);
int touchwin(WINDOW *win);
int touchline(WINDOW *win, int start, int count);
int untouchwin(WINDOW *win);
```

- derwin(): is just like subwin(), except that begin_y,begin_x are window-relative (not screen relative)
- subwin() and derwin(): creates and returns a pointer to a WINDOW with the specified qty of nlines lines and ncol columns.
- Changes to a subwindow writes to the containing window's character array.

Curses: Creating and Removing Windows

WINDOW *derwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);
int touchwin(WINDOW *win);
int touchline(WINDOW *win, int start, int count);
int untouchwin(WINDOW *win);

- derwin(): is just like subwin(), except that begin_y,begin_x are window-relative (not screen relative)
- subwin() and derwin(): creates and returns a pointer to a WINDOW with the specified qty of nlines lines and ncol columns.
- Changes to a subwindow writes to the containing window’s character array.
- Call touchwin() or touchline() on orig before calling wrefresh on the subwindow

Curses: Creating and Removing Windows

```
WINDOW *derwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);
int touchwin(WINDOW *win);
int touchline(WINDOW *win, int start, int count);
int untouchwin(WINDOW *win);
```

- derwin(): is just like subwin(), except that begin_y,begin_x are window-relative (not screen relative)
- subwin() and derwin(): creates and returns a pointer to a WINDOW with the specified qty of nlines lines and ncol columns.
- Changes to a subwindow writes to the containing window's character array.
- Call touchwin() or touchline() on orig before calling wrefresh on the subwindow
- touchwin() and touchline() throw away optimization information about which parts of win have been touched by flagging the entire window as needing redrawing.

Curses: Creating and Removing Windows

```
WINDOW *derwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);
int touchwin(WINDOW *win);
int touchline(WINDOW *win, int start, int count);
int untouchwin(WINDOW *win);
```

- derwin(): is just like subwin(), except that begin_y,begin_x are window-relative (not screen relative)
- subwin() and derwin(): creates and returns a pointer to a WINDOW with the specified qty of nlines lines and ncol columns.
- Changes to a subwindow writes to the containing window's character array.
- Call touchwin() or touchline() on orig before calling wrefresh on the subwindow
- touchwin() and touchline() throw away optimization information about which parts of win have been touched by flagging the entire window as needing redrawing.
- untouchwin() flags a window as pristine (all lines untouched since the last call to wrefresh())

Curses: Manipulating Windows

```
int mvwin(WINDOW *win, int y, int x);
int mvderwin(WINDOW *win, int par_y, int par_x);
WINDOW *dupwin(WINDOW *win);
int overlay(const WINDOW *srcwin, WINDOW *dstwin);
int overwrite(const WINDOW *srcwin, WINDOW *dstwin);
int copywin(const WINDOW *srcwin, WINDOW *dstwin, int sminrow, int smincol, int
dminrow, int dmincol, int dmaxrow, int dmaxcol, int overlay);
```

- mvwin(): move win to y,x in screen coordinates

Curses: Manipulating Windows

```
int mvwin(WINDOW *win, int y, int x);
int mvderwin(WINDOW *win, int par_y, int par_x);
WINDOW *dupwin(WINDOW *win);
int overlay(const WINDOW *srcwin, WINDOW *dstwin);
int overwrite(const WINDOW *srcwin, WINDOW *dstwin);
int copywin(const WINDOW *srcwin, WINDOW *dstwin, int sminrow, int smincol, int
dminrow, int dmincol, int dmaxrow, int dmaxcol, int overlay);
```

- mvwin(): move win to y,x in screen coordinates
- mvderwin(): move a derived WINDOW, win, to y,x in window coordinates

Curses: Manipulating Windows

```
int mvwin(WINDOW *win, int y, int x);
int mvderwin(WINDOW *win, int par_y, int par_x);
WINDOW *dupwin(WINDOW *win);
int overlay(const WINDOW *srcwin, WINDOW *dstwin);
int overwrite(const WINDOW *srcwin, WINDOW *dstwin);
int copywin(const WINDOW *srcwin, WINDOW *dstwin, int sminrow, int smincol, int
dminrow, int dmincol, int dmaxrow, int dmaxcol, int overlay);
```

- mvwin(): move win to y,x in screen coordinates
- mvderwin(): move a derived WINDOW, win, to y,x in window coordinates
- dupwin(): creates an exact duplicate of the WINDOW win

Curses: Manipulating Windows

```
int mvwin(WINDOW *win, int y, int x);
int mvderwin(WINDOW *win, int par_y, int par_x);
WINDOW *dupwin(WINDOW *win);
int overlay(const WINDOW *srcwin, WINDOW *dstwin);
int overwrite(const WINDOW *srcwin, WINDOW *dstwin);
int copywin(const WINDOW *srcwin, WINDOW *dstwin, int sminrow, int smincol, int
dminrow, int dmincol, int dmaxrow, int dmaxcol, int overlay);
```

- mvwin(): move win to y,x in screen coordinates
- mvderwin(): move a derived WINDOW, win, to y,x in window coordinates
- dupwin(): creates an exact duplicate of the WINDOW win
- overlay(): overlays srcwin atop dstwin, excluding blanks.

Curses: Manipulating Windows

```
int mvwin(WINDOW *win, int y, int x);
int mvderwin(WINDOW *win, int par_y, int par_x);
WINDOW *dupwin(WINDOW *win);
int overlay(const WINDOW *srcwin, WINDOW *dstwin);
int overwrite(const WINDOW *srcwin, WINDOW *dstwin);
int copywin(const WINDOW *srcwin, WINDOW *dstwin, int sminrow, int smincol, int
dminrow, int dmincol, int dmaxrow, int dmaxcol, int overlay);
```

- mvwin(): move win to y,x in screen coordinates
- mvderwin(): move a derived WINDOW, win, to y,x in window coordinates
- dupwin(): creates an exact duplicate of the WINDOW win
- overlay(): overlays srcwin atop dstwin, excluding blanks.
- overwrite(): just like overlay(), except blanks are also copied.

Curses: Manipulating Windows

```
int mvwin(WINDOW *win, int y, int x);
int mvderwin(WINDOW *win, int par_y, int par_x);
WINDOW *dupwin(WINDOW *win);
int overlay(const WINDOW *srcwin, WINDOW *dstwin);
int overwrite(const WINDOW *srcwin, WINDOW *dstwin);
int copywin(const WINDOW *srcwin, WINDOW *dstwin, int sminrow, int smincol, int
dminrow, int dmincol, int dmaxrow, int dmaxcol, int overlay);
```

- mvwin(): move win to y,x in screen coordinates
- mvderwin(): move a derived WINDOW, win, to y,x in window coordinates
- dupwin(): creates an exact duplicate of the WINDOW win
- overlay(): overlays srcwin atop dstwin, excluding blanks.
- overwrite(): just like overlay(), except blanks are also copied.
- copywin(): use this routine to copy a subset of srcwin onto dstwin, with overlay controlling whether blanks are copied (=0) or not (=1)

Curses: Getting Characters

```
int getch(void);
int wgetch(WINDOW *win);
int mvgetch(int y, int x);
int mvwgetch(WINDOW *win, int y, int x);
int ungetch(int ch);
int getstr(char *str);
int wgetstr(WINDOW *win, char *str);
```

- getch(): checks the boolean _echoit (set by echo(), unset by noecho()) to determine if the entered character should be echo'd. If true, addch() will be used to add the character at the logical cursor from the screen.

Curses: Getting Characters

```
int getch(void);
int wgetch(WINDOW *win);
int mvgetch(int y, int x);
int mvwgetch(WINDOW *win, int y, int x);
int ungetch(int ch);
int getstr(char *str);
int wgetstr(WINDOW *win, char *str);
```

- getch(): checks the boolean _echoit (set by echo(), unset by noecho()) to determine if the entered character should be echo'd. If true, addch() will be used to add the character at the logical cursor from the screen.
- wgetch(): gets a character from win

Curses: Getting Characters

```
int getch(void);
int wgetch(WINDOW *win);
int mvgetch(int y, int x);
int mvwgetch(WINDOW *win, int y, int x);
int ungetch(int ch);
int getstr(char *str);
int wgetstr(WINDOW *win, char *str);
```

- getch(): checks the boolean _echoit (set by echo(), unset by noecho()) to determine if the entered character should be echo'd. If true, addch() will be used to add the character at the logical cursor from the screen.
- wgetch(): gets a character from win
- mvgetc(): gets a character from win, but at screen-relative position y,x.

Curses: Getting Characters

```
int getch(void);
int wgetch(WINDOW *win);
int mvgetch(int y, int x);
int mvwgetch(WINDOW *win, int y, int x);
int ungetch(int ch);
int getstr(char *str);
int wgetstr(WINDOW *win, char *str);
```

- getch(): checks the boolean _echoit (set by echo(), unset by noecho()) to determine if the entered character should be echo'd. If true, addch() will be used to add the character at the logical cursor from the screen.
- wgetch(): gets a character from win
- mvgetc(): gets a character from win, but at screen-relative position y,x.
- ungetch(): pushes ch back onto the input queue (there is but one input queue for all windows)

Curses: Getting Characters

```
int getch(void);
int wgetch(WINDOW *win);
int mvgetch(int y, int x);
int mvwgetch(WINDOW *win, int y, int x);
int ungetch(int ch);
int getstr(char *str);
int wgetstr(WINDOW *win, char *str);
```

- getch(): checks the boolean _echoit (set by echo(), unset by noecho()) to determine if the entered character should be echo'd. If true, addch() will be used to add the character at the logical cursor from the screen.
- wgetch(): gets a character from win
- mvgetc(): gets a character from win, but at screen-relative position y,x.
- ungetch(): pushes ch back onto the input queue (there is but one input queue for all windows)
- getstr(): getch() is repeatedly called until a <cr> or a <eol> is encountered.

Curses: Getting Characters

```
int getch(void);
int wgetch(WINDOW *win);
int mvgetch(int y, int x);
int mvwgetch(WINDOW *win, int y, int x);
int ungetch(int ch);
int getstr(char *str);
int wgetstr(WINDOW *win, char *str);
```

- getch(): checks the boolean _echoit (set by echo(), unset by noecho()) to determine if the entered character should be echo'd. If true, addch() will be used to add the character at the logical cursor from the screen.
- wgetch(): gets a character from win
- mvgetc(): gets a character from win, but at screen-relative position y,x.
- ungetch(): pushes ch back onto the input queue (there is but one input queue for all windows)
- getstr(): getch() is repeatedly called until a <cr> or a <eol> is encountered.
- wgetstr(): like getstr(), except characters obtained from win

Curses: Formatted Input

```
int scanw(char *fmt, ...);
int wscanw(WINDOW *win, char *fmt, ...);
int mvscanw(int y, int x, char *fmt, ...);
int mvwscanw(WINDOW *win, int y, int x, char *fmt, ...);
int vw_scanw(WINDOW *win, char *fmt, va_list varglist);
```

- scanw(), wscanw(), mvscanw(): akin to scanf(); returns the quantity of format coded fields read in

Curses: Formatted Input

```
int scanw(char *fmt, ...);
int wscanw(WINDOW *win, char *fmt, ...);
int mvscanw(int y, int x, char *fmt, ...);
int mvwscanw(WINDOW *win, int y, int x, char *fmt, ...);
int vw_scanw(WINDOW *win, char *fmt, va_list varglist);
```

- scanw(), wscanw(), mvscanw(): akin to scanf(); returns the quantity of format coded fields read in
- vw_scanw(): similar to vscanf(), using a variable argument list.

  varglist is a va_list (a pointer to a list of variables)

  (use #include <stdarg.h> with this function)

Curses: Character Output

```
int addch(const chtype ch);
int waddch(WINDOW *win, const chtype ch);
int mvaddch(int y, int x, const chtype ch);
int mvwaddch(WINDOW *win, int y, int x, const chtype ch);
int addstr(const char *str);
int waddstr(WINDOW *win, const char *str);
int mvaddstr(int y, int x, const char *str);
int mvwaddstr(WINDOW *win, int y, int x, const char *str);
```

- addch(), waddch(), mvaddch(), mvwaddch(): add a character to stdscr

Curses: Character Output

```
int addch(const chtype ch);
int waddch(WINDOW *win, const chtype ch);
int mvaddch(int y, int x, const chtype ch);
int mvwaddch(WINDOW *win, int y, int x, const chtype ch);
int addstr(const char *str);
int waddstr(WINDOW *win, const char *str);
int mvaddstr(int y, int x, const char *str);
int mvwaddstr(WINDOW *win, int y, int x, const char *str);
```

- addch(), waddch(), mvaddch(), mvwaddch(): add a character to stdscr
- addstr(), waddstr(), mvaddstr(), mvwaddstr(): add a string to stdscr by calling addch() multiple times

Curses: Borders and Boxes

int border(chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int wborder(WINDOW *win, chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int box(WINDOW *win, chtype verch, chtype horch);

- border(), wborder(), and box() draw a box around the edges of a window.

Curses: Borders and Boxes

```
int border(chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int wborder(WINDOW *win, chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int box(WINDOW *win, chtype verch, chtype horch);
```

- `border()`, `wborder()`, and `box()` draw a box around the edges of a window.
- `box()` is shorthand for `wborder(win,verch,verch,horch,horch,0,0,0,0)`

Curses: Borders and Boxes

```
int border(chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int wborder(WINDOW *win, chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int box(WINDOW *win, chtype verch, chtype horch);
```

- `border()`, `wborder()`, and `box()` draw a box around the edges of a window.
- `box()` is shorthand for `wborder(win,verch,verch,horch,horch,0,0,0,0)`
- `border()`, `wborder()`: specify the box characters with:

Curses: Borders and Boxes

int border(chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int wborder(WINDOW *win, chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int box(WINDOW *win, chtype verch, chtype horch);

- border(), wborder(), and box() draw a box around the edges of a window.
- box() is shorthand for wborder(win,verch,verch,horch,horch,0,0,0,0)
- border(), wborder(): specify the box characters with:

ls	left side	tl	top left-hand corner

Curses: Borders and Boxes

```
int border(chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int wborder(WINDOW *win, chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int box(WINDOW *win, chtype verch, chtype horch);
```

- `border()`, `wborder()`, and `box()` draw a box around the edges of a window.
- `box()` is shorthand for `wborder(win,verch,verch,horch,horch,0,0,0,0)`
- `border()`, `wborder()`: specify the box characters with:

ls	left side	tl	top left-hand corner
rs	right side	tr	top right-hand corner

Curses: Borders and Boxes

```
int border(chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int wborder(WINDOW *win, chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int box(WINDOW *win, chtype verch, chtype horch);
```

- `border()`, `wborder()`, and `box()` draw a box around the edges of a window.
- `box()` is shorthand for `wborder(win,verch,verch,horch,horch,0,0,0,0)`
- `border()`, `wborder()`: specify the box characters with:

ls	left side	tl	top left-hand corner
rs	right side	tr	top right-hand corner
ts	top side	bl	bottom left-hand corner

Curses: Borders and Boxes

```
int border(chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int wborder(WINDOW *win, chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int box(WINDOW *win, chtype verch, chtype horch);
```

- `border()`, `wborder()`, and `box()` draw a box around the edges of a window.
- `box()` is shorthand for `wborder(win,verch,verch,horch,horch,0,0,0,0)`
- `border()`, `wborder()`: specify the box characters with:

ls	left side	tl	top left-hand corner
rs	right side	tr	top right-hand corner
ts	top side	bl	bottom left-hand corner
bs	bottom side	br	bottom right-hand corner

Curses: Borders and Boxes

```
int border(chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int wborder(WINDOW *win, chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int box(WINDOW *win, chtype verch, chtype horch);
```

- border(), wborder(), and box() draw a box around the edges of a window.
- box() is shorthand for wborder(win,verch,verch,horch,horch,0,0,0,0)
- border(), wborder(): specify the box characters with:

ls	left side	tl	top left-hand corner
rs	right side	tr	top right-hand corner
ts	top side	bl	bottom left-hand corner
bs	bottom side	br	bottom right-hand corner

See curs03.c

Curses: Borders and Boxes, con’t.

- If any of these characters are 0, default characters are used instead:

  (defined in curses.h)

Curses: Borders and Boxes, con’t.

- If any of these characters are 0, default characters are used instead:

  (defined in curses.h)

ls	ACS_VLINE	tl	ACS_ULCORNER

Curses: Borders and Boxes, con't.

- If any of these characters are 0, default characters are used instead:

  (defined in curses.h)

ls	ACS_VLINE	tl	ACS_ULCORNER
rs	ACS_VLINE	tr	ACS_URCORNER

Curses: Borders and Boxes, con't.

- If any of these characters are 0, default characters are used instead:

  (defined in curses.h)

ls	ACS_VLINE	tl	ACS_ULCORNER
rs	ACS_VLINE	tr	ACS_URCORNER
ts	ACS_HLINE	bl	ACS_LLCORNER

Curses: Borders and Boxes, con't.

- If any of these characters are 0, default characters are used instead:

  (defined in curses.h)

ls	ACS_VLINE	tl	ACS_ULCORNER
rs	ACS_VLINE	tr	ACS_URCORNER
ts	ACS_HLINE	bl	ACS_LLCORNER
bs	ACS_HLINE	br	ACS_LRCORNER

Curses: Formatted Character Output

```
int printw(const char *fmt, ...);
int wprintw(WINDOW *win, const char *fmt, ...);
int mvprintw(int y, int x, const char *fmt, ...);
int mvwprintw(WINDOW *win, int y, int x, const char *fmt, ...);
int vwprintw(WINDOW *win, const char *fmt, va_list varglist);
int vw_printw(WINDOW *win, const char *fmt, va_list varglist);
int insertln(void);
int winsertln(WINDOW *win);
```

- *printw*(): These functions provide formatted printf-style output to stdscr

Curses: Formatted Character Output

```
int printw(const char *fmt, ...);
int wprintw(WINDOW *win, const char *fmt, ...);
int mvprintw(int y, int x, const char *fmt, ...);
int mvwprintw(WINDOW *win, int y, int x, const char *fmt, ...);
int vwprintw(WINDOW *win, const char *fmt, va_list varglist);
int vw_printw(WINDOW *win, const char *fmt, va_list varglist);
int insertln(void);
int winsertln(WINDOW *win);
```

- *printw*(): These functions provide formatted printf-style output to stdscr
- *insertln(): These routines insert a blank line above the current line; the bottom line is lost

Curses: Formatted Character Output

```
int printw(const char *fmt, ...);
int wprintw(WINDOW *win, const char *fmt, ...);
int mvprintw(int y, int x, const char *fmt, ...);
int mvwprintw(WINDOW *win, int y, int x, const char *fmt, ...);
int vwprintw(WINDOW *win, const char *fmt, va_list varglist);
int vw_printw(WINDOW *win, const char *fmt, va_list varglist);
int insertln(void);
int winsertln(WINDOW *win);
```

- *printw*(): These functions provide formatted printf-style output to stdscr
- *insertln(): These routines insert a blank line above the current line; the bottom line is lost

See curs05.c

Curses: Erasing and Clearing, con't.

```
int delch(void);
int wdelch(WINDOW *win);
int mvdelch(int y, int x);
int mvwdelch(WINDOW *win, int y, int x);
int deleteln(void);
int wdeleteln(WINDOW *win);
```

- delch(), wdelch(), mvdelch(), mvwdelch(): delete the character under the cursor; characters shift in from the right

Curses: Erasing and Clearing, con’t.

```
int delch(void);
int wdelch(WINDOW *win);
int mvdelch(int y, int x);
int mvwdelch(WINDOW *win, int y, int x);
int deleteln(void);
int wdeleteln(WINDOW *win);
```

- delch(), wdelch(), mvdelch(), mvwdelch(): delete the character under the cursor; characters shift in from the right

- deleteln(), wdeleteln(): these routines delete the line under the cursor; lines below the deleted line are shifted up

Curses: Cursor Location

```
int move(int y, int x);
int wmove(WINDOW *win, int y, int x);
void getyx(WINDOW *win, int y, int x);
void getparyx(WINDOW *win, int y, int x);
void getbegyx(WINDOW *win, int y, int x);
void getmaxyx(WINDOW *win, int y, int x);
```

- move(), wmove(): Move the cursor to window-relative y,x (the upper left-hand corner of the window is at (0,0))

Curses: Cursor Location

```
int move(int y, int x);
int wmove(WINDOW *win, int y, int x);
void getyx(WINDOW *win, int y, int x);
void getparyx(WINDOW *win, int y, int x);
void getbegyx(WINDOW *win, int y, int x);
void getmaxyx(WINDOW *win, int y, int x);
```

- move(), wmove(): Move the cursor to window-relative y,x (the upper left-hand corner of the window is at (0,0))

- getyx(): puts the current cursor position in WINDOW win into y,x (its a macro)

Curses: Cursor Location

```
int move(int y, int x);
int wmove(WINDOW *win, int y, int x);
void getyx(WINDOW *win, int y, int x);
void getparyx(WINDOW *win, int y, int x);
void getbegyx(WINDOW *win, int y, int x);
void getmaxyx(WINDOW *win, int y, int x);
```

- move(), wmove(): Move the cursor to window-relative y,x (the upper left-hand corner of the window is at (0,0))
- getyx(): puts the current cursor position in WINDOW win into y,x (its a macro)
- getparyx(): places beginning coordinates of the subwindow relative to the parent window into y,x. (if you use a window, you'll get -1s)

Curses: Cursor Location

```
int move(int y, int x);
int wmove(WINDOW *win, int y, int x);
void getyx(WINDOW *win, int y, int x);
void getparyx(WINDOW *win, int y, int x);
void getbegyx(WINDOW *win, int y, int x);
void getmaxyx(WINDOW *win, int y, int x);
```

- move(), wmove(): Move the cursor to window-relative y,x (the upper left-hand corner of the window is at (0,0))
- getyx(): puts the current cursor position in WINDOW win into y,x (its a macro)
- getparyx(): places beginning coordinates of the subwindow relative to the parent window into y,x. (if you use a window, you'll get -1s)
- getbegyx(): get the beginning coordinates of the specified win

Curses: Cursor Location

```
int move(int y, int x);
int wmove(WINDOW *win, int y, int x);
void getyx(WINDOW *win, int y, int x);
void getparyx(WINDOW *win, int y, int x);
void getbegyx(WINDOW *win, int y, int x);
void getmaxyx(WINDOW *win, int y, int x);
```

- move(), wmove(): Move the cursor to window-relative y,x (the upper left-hand corner of the window is at (0,0))
- getyx(): puts the current cursor position in WINDOW win into y,x (its a macro)
- getparyx(): places beginning coordinates of the subwindow relative to the parent window into y,x. (if you use a window, you'll get -1s)
- getbegyx(): get the beginning coordinates of the specified win
- getmaxyx(): get the size of the specified win

Curses: Cursor Location

```
int move(int y, int x);
int wmove(WINDOW *win, int y, int x);
void getyx(WINDOW *win, int y, int x);
void getparyx(WINDOW *win, int y, int x);
void getbegyx(WINDOW *win, int y, int x);
void getmaxyx(WINDOW *win, int y, int x);
```

- move(), wmove(): Move the cursor to window-relative y,x (the upper left-hand corner of the window is at (0,0))
- getyx(): puts the current cursor position in WINDOW win into y,x (its a macro)
- getparyx(): places beginning coordinates of the subwindow relative to the parent window into y,x. (if you use a window, you'll get -1s)
- getbegyx(): get the beginning coordinates of the specified win
- getmaxyx(): get the size of the specified win

See curs08.c