Curses-based classes

The **CursesWindow** class is a repackaging of standard curses library features into a class. It relies on **curses.h**.

The supplied **curses.h** is a fairly conservative declaration of curses library features, and does not include features like ``screen'' or X-window support. It is, for the most part, an adaptation, rather than an improvement of C-based **curses.h** files. The only substantive changes are the declarations of many functions as inline functions rather than macros, which was done solely to allow overloading.

The **CursesWindow** class encapsulates curses window functions within a class. Only those functions that control windows are included: Terminal control functions and macros like **cbreak** are not part of the class. All **CursesWindows** member functions have names identical to the corresponding curses library functions, except that the ``w'' prefix is generally dropped. Descriptions of these functions may be found in your local curses library documentation.

A **CursesWindow** may be declared via

CursesWindow w(WINDOW* win) attaches w to the existing WINDOW* win. This is constructor is normally used only in the following special case.

CursesWindow w(stdscr) attaches w to the default curses library standard screen window.

CursesWindow w(int lines, int cols, int begin_y, int begin_x) attaches to an allocated curses window with the indicated size and screen position.

CursesWindow sub(CursesWindow& w,int l,int c,int by,int bx,char ar='a') attaches to a subwindow of w created via the curses `subwin' command. If ar is sent as `r', the origin (by, bx) is relative to the parent window, else it is absolute.

The class maintains a static counter that is used in order to automatically call the curses library **initscr** and **endscr** functions at the proper times. These need not, and should not be called ``manually''.

CursesWindows maintain a tree of their subwindows. Upon destruction of a **CursesWindow**, all of their subwindows are also invalidated if they had not previously been destroyed.

It is possible to traverse trees of subwindows via the following member functions

CursesWindow* w.parent() returns a pointer to the parent of the subwindow, or 0 if there is none.

CursesWindow* w.child() returns the first child subwindow of the window, or 0 if there is none.

CursesWindow* w.sibling() returns the next sibling of the subwindow, or 0 if there is none.

For example, to call some function **visit** for all subwindows of a window, you could write

```
void traverse(CursesWindow& w)
{
  visit(w);
  if (w.child() != 0) traverse(*w.child);
  if (w.sibling() != 0) traverse(*w.sibling);
}
```