Practicum 11 - Curses

Assignment Details

Assigned: April 16th, 2014. Due: April 20th, 2014 at midnight.

A complete submission will include the modified code from the practicum.

Practicum

This practicum is set up to introduce using a library in the Unix environment. Specifically, we'll be using the ncurses[1] library – and going over how to do a lot of what will be needed for PA4[2] credit.

For starters, update your source control directory, and in your la11 directory you should see some familiar code…

Before we really get started, make sure everything you have builds and executes properly – and displays a correct (if unexciting) dungeon level.

For this practicum, we're going to call some of the functions in the ncurses library, so the first thing we'll need to do is include the header file. At the top of dlmain.cpp, add:

```
#include <ncurses.h>
```

And then recompile, making sure it's finding the correct header file.

Note that we're only going to use a few functions from the library; more information about the library is available here: http://invisible-island.net/ncurses/ncurses-intro.html

Additionally, a complete reference is here: http://invisible-island.net/ncurses/man/ncurses.3x.html

It should be noted that curses is a C library, not a C++ library. For our purposes this just means that the library consists entirely of functions – it lacks classes. But as C++ is a superset of C[3], we can still use C libraries without any problems.

So, the first thing we'll do is call a ncurses function – `initscr` – to set up the environment before we call more functions.

Go ahead and remove the `dl.dump()`; line in dlmain.cpp and replace it with a call to `initscr();`. Exit out of your preferred editor and run make. You should get something like this:

---

[1] "new curses"; an enhanced form of the venerable curses library. Both support fancier input and output in terminal windows.

[2] Inasmuch as PA4 *exists* as an assignment as opposed to just some bonus points on top of PA3…

[3] This is not *exactly* true, but fairly close – C++ is *effectively* a superset of C, and the differences between the languages that make it not a strict superset are mostly subtle. At the least, it's close enough to true that we can use any C library in C++, but the reverse is not true.

```
g++ --std=c++0x -c dlmain.cp -o dlmain.o
g++ -o dlmain DungeonLevel.o dlmain.o
dlmain.o: In function 'main':
dlmain.cpp:(.text+0x6d): undefined reference to 'initscr'
collect2: ld returned 1 exit status
make: *** [dlmain] Error 1
```

Oh, my. We've got an undefined reference to `initscr`. The first thing we should note here is that the error is showing up at the *linkage* step, **not** the compilation step. Note that the last command executed by make – `g++ -o dlmain DungeonLevel.o dlmain.o` – does not perform any compilation, it just attempts to link the pre-compiled .o files into an executable.

So what does this mean? Compiling dlmain.cpp completed successfully, but linking everything together did not. This means that the compiler knows that `initscr()` exists somewhere (else the compilation step would have failed), but the linker can't find it.

To fix this, we're going to have to modify the makefile. Look at the rule which actually performs linkage. We're going to need to tell that rule – and no others! – that we need to link against the ncurses library. Since ncurses is installed at the system level, we don't have a .o file handy in the current directory, but we can use g++'s -l command line option to tell it to look for a library called ncurses. Update the linkage rule so it is:

```
dlmain: $(OBJECTS) dlmain.o
        g++ -o dlmain $^ -lncurses
```

Invoke make again, and everything should be happy again.

Now, go back into dlmain.cpp, and let's start making some changes.

First, to pair with `initscr()`, add a call to `endwin()` at the end of the program – right before the `return 0` at the end. This cleans up after ncurses and restores normal terminal operation.

So, now that we've both started up and shut down ncurses, let's go ahead and output the generated dungeon level to the screen. To do so, we're going to need to iterate through the DungeonLevel – note that the provided DungeonLevel has the `getWidth()`, `getHeight()`, and `at()` methods which will allow this – and then call ncurses methods to put these values to the screen.

So, take the generated dungeon level – `dl` in the provided code – iterate through it, and display the tiles in the level one at a time by first moving the cursor to the appropriate position(via `move()`), and then write a character out (via `addch()`). This loop should look something like[4]:

[4] Note that the PA3 requirement for having your DungeonLevel be able to generate something in a similar format exists solely to make this easier when we get to PA4.

```
for( size_t x = 0; x < dl.getWidth(); x++ )
{
      for( size_t y = 0; y < dl.getHeight(); y++ )
      {
            move(y,x);
            addch(dl.at(x,y));
      }
}
```

Note that `move()` accepts it coördinates y first!

So, implement that output loop, compile, run, and… nothing.

First thing to note here – when you call ncurses's functions, they all just update data structures internal to ncurses. To actually draw something to the screen, we need to call `refresh()` after finishing all of the drawing – that's the method that actually does the drawing[5].

Build, run, and… nothing.

We've got one more step to show output – note that when we call `endwin()` at the end of the program, it restores the normal terminal output. This also has the side effect of clearing out anything we wrote to the screen while using ncurses, so we need to add something to the program to delay shutting down. Enter the `getch()` function. This function reads a single character from the keyboard, but unlike doing so through the methods we've used thus far in CS215 and CS216, it doesn't wait for the user to hit enter[6] – so we get a character directly from the keyboard.

So, adding a call to `getch()` right before `endwin()`, and we should display a dungeon level, then wait for a key to be pressed, then exit. Build, run and… hey, it works this time.

Now, one more trick and we're done for today – just to demonstrate updating the whole screen dynamically, we're going to modify the program to perform a loop: instead of just drawing the one generated DungeonLevel, instead we're going to draw a level[7], get a key press from the user, and then – if that key is not q[8] – do it again.

We'll just add more ncurses function at this point – `clear()` – which clears the screen; call this before drawing each dungeon level.

---

[5] There is actually some variation in versions of ncurses with regard to this; but, still, one should call `refresh()`.

[6] Which, it should be pointed out, there is no universally portable way to do, so instead of having something like this available in the C++ STL, we must use a different library.

[7] Remember that you'll need to seed your random number generator *outside* of the loop.

[8] `while(getch() != 'q')` might just do the job here, as `getch()` gives us the character code of the key read from the keyboard

Once all is said and done, you should have an understanding of how to draw the dungeon level each turn of the game in PA3/PA4 without having to generate an unending stream of text[9].

A brief ncurses reference

`WINDOW * initscr()` – Initializes the ncurses environment and prepares writing to the console. Return value can be ignored if you're only using one window (like what we'll be doing in this and the next practicum…)

`int clear()` – Clears the screen. Note that like the rest of the output functions, `refresh()` must be called before the screen is actually updated. Returns an error code; on success, it will be `OK`.

`int noecho()` – Instructs ncurses to not echo any characters input from the console back to the console. Might be useful to call right after `initscr()`. Returns an error code; on success, it will be `OK`.

`int getch()` – Gets a single character from the console *without* requiring the user to hit enter.

`int refresh()` – Updates the screen with what has been drawn (note: you must call this function to actually display something to the screen). Returns an error code; on success, it will be `OK`.

`int move(int y, int x)` – Moves the cursor to the specified position on the screen. The top left character is 0,0; the positive x axis goes right, and the positive y axis goes down. Note that the y coördinate is specified first. Returns an error code; on success, it will be `OK`.

`int addch(char c)` – Draws the specified character at the current cursor position. Returns an error code; on success, it will be `OK`.

`int endwin()` – Cleans up after ncurses and restores the terminal to its normal state; this should always be called at the end of the program. Returns an error code; on success, it will be `OK`.

`int addstr(const char * str)` – Writes the specified string (C-style, zero terminated – if you have a string stored in a C++ string, you'll need to pass its `c_str()` return, not the string itself to this function!).

Note that many, many more functions are listed at http://invisible-island.net/ncurses/man/ncurses.3x.html.

[9] Note that while PA4 is going to require this, PA3 does not – we're just getting ready for PA4 a bit ahead of time, as we won't have a lot of time for it.