To more easily display characters on the screen, we are going to use the ncurses C library. This allows us to write a character to a specific X, Y location on screen, which will be useful in drawing the dungeon grid. The sections below detail how to install and compile with this library based on your OS.

Installing – Linux:

On Linux, ncurses can be installed using `apt-get` with the following command:

```
sudo apt-get install libncurses5-dev libncursesw5-dev
```

Installing – Mac:

On Mac, ncurses can be installed using `brew` with the following command:

```
brew install ncurses
```

Compiling – Linux and Mac:

Both Linux and Mac simply require add the library using `-lncurses`. For example:

```
g++ -g -lncurses example.cpp -o example
```

Installing – Windows:

Unfortunately, ncurses does not exist on Windows, though an alternative exists called PDCurses. However, the authors do not ship binaries, so it is necessary to compile it yourself. To make this easier on the students, the files `pdcurses.lib` and `pdcurses.dll` are included. This is compiled for x64 for consistency with the build instructions for Xerces. Note some versions of MingW64 install i686 which would not work with this file.

Since it is difficult to use Xerces on Windows outside of Visual Studio, these instructions describe how to install PDCurses for Visual Studio. If you managed to use Make on Windows with Xerces and want to use Make for PDCurses as well, ask on Piazza and I can help set that up, though the directions are pretty similar.

Compiling PDCurses (optional)

The `.lib` and `.dll` files should work for most cases. However, if you want to learn about compiling external libraries or for some reason the file does not work, the steps below tell you how to compile PDCurses:

- Download PDCurses from this link. Extract the contents of the zip file into a folder.
- Back in the extracted folder, there is a `README.md` file. It directs us into the `wincon` folder for compiling on Windows.
- According to `wincon/README.md`, we should be able to simply run in the folder to build the library.
  - To run `nmake`, first type “x64 Native Tools Command Prompt for VS” into the start menu search bar. A command prompt should show.
  - If the native tools command prompt is missing, ensure you have the Visual Studio build tools properly installed
- In the command prompt, navigate to the `wincon` folder from where you extracted PDCurses.
- Run the following command:
  - `nmake -f Makefile.vc DLL=Y platform=x64`
  - The options are as follow:
    - `-f Makefile.vc`: build using the makefile designed for nmake, as the default `Makefile` is for the Linux version of make
    - `DLL=Y`: Express that we want to build using a dll. This was done for consistency with Xerces, leaving off this option will build just a library that will be included in the executable.
    - `platform=x64`: Specifies we are building for x64, for consistency with Xerces again.
- You should have a `pdcurses.lib` and `pdcurses.dll`

Compiling – Windows:

The easiest way to compile on Windows is by creating a Visual Studio project, though it is also possible using an NMake project.

- Create a new empty project, or navigate to an existing project.
- Make sure your project has at least one C++ source file, so its recognized as a C++ project.
- Open your project in Windows Explorer. Create two folders inside: `lib` and `include`
    - The names of the folders are not important, as long as they are consistent with later steps.
    - Within the `include` folder, paste all three `.h` files from the zip file. These will be used on compilation to include the new functions.
        - If you choose to compile PDCurses yourself, its better to use the three header files from the zip you downloaded.
    - Within the lib folder, paste `pdcurses.lib`. This will be used during linking to include the extra functions.
    - Within your project's run directory (by default, same folder with your source files), paste `pdcurses.dll`. This will be used at runtime to include the library code.
- Back in Visual Studio, right click your project and click properties
- On the left sidebar under "Configuration Properties":
    - Click "C/C++" > "General"
        - Add "`.\include`" under "Additional Include Directories".
        - If you have another folder there (such as Xerces), separate them with semicolons.
    - Click "Linker" > "General"
        - Add "`.\lib`" under "Additional Library Directories".
        - If you have another folder there (such as Xerces), separate them with semicolons.
    - Click "Linker" > "Input"
        - Add "`legacy_stdio_definitions.lib`" and "`pdcurses.lib`" under "Additional Library Directories".
        - Separate these, along with any other libraries (such as Xerces) with semicolons.
- In the top left, click "Configuration Manager"
    - Switch "Active Solution Platform" to "x64"
    - Switch "Platform" under the project to "x64"

You should now be able to build the project and run using the local debugger.