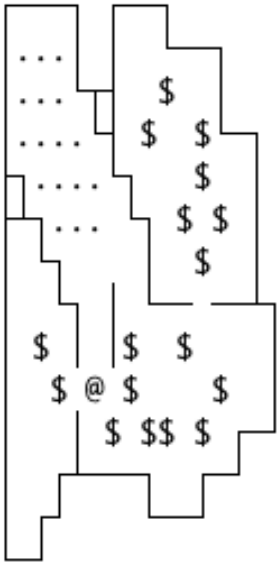
Computer Science 237
Assignment 2
Due next week, before lab.

The purpose of this week's lab is to finish off a simple puzzle, called Sokoban. I have written most of the code to play this simple solitare game; what remains is the detection of a solved puzzle. Once this method is written correctly, you should be able to solve each of the 90 levels in turn. This handout describes the task.

Sokoban ('warehouse worker') is a puzzle developed by Hiroyuki Imabayashi, one of many excellent Japanese puzzle designers to appear in the last few decades. The puzzle shows an overhead view of a warehouse including a worker (represented by an `@`), a number of boxes (each, a `$`), and a number of storage locations (located at dots `.`):



Sokoban level 45, at the start.

The worker's job is to move about the warehouse (left, right, up, down) and push the boxes into the storage locations. The worker can only push one crate at a time. He cannot pull them. If boxes get pushed against other objects (walls or boxes) it can become impossible to move them further.

The puzzle has been written to be played in the traditional way: on a terminal using the symbols shown above. When the worker or a box is in the the storage area, their symbols change to `+` and `*`, respectively. To reenforce your emacs skills, you'll find the worker can be moved about using the traditional emacs cursor movement functions for line-up (`C-p`), line-down (`C-n`), left (`C-b`) and right (`C-f`). Undo is, as it is in emacs, `C-_`. To give up, type `C-g`.

You can pick up the distribution of this lab from the course web page as `sokoban.tar.gz`. This file is 'tarred' and 'zipped'. To unzip and untar this package with

```
tar xvfz sokoban.tar.gz
```

The result is a directory, `sokoban`, that contains the source, this lab handout, and a number of data files.

To build the program, you simply type:

```
make sokoban
```

Whenever you make changes to any of the source files, you simply type the make command again. When you are finished, you may want to

```
make clean
```

which gets rid of all the cruft that is left behind during the development and debugging processes. It leaves the executable alone.

If you want to get rid of the executable as well, you can type:

```
make realclean
```

and the directory is left in a pristine condition.

All I require is that you write one procedure,

```
extern int win(level *l);
```

At the moment, `win` always returns zero, so it is not possible to move on to the next level.

Ideally, this procedure scans across the entire puzzle, retrieves the representations of each of the box locations and checks to make sure that each is stowed away. If all the boxes are in storage, this function highlights the boxes and returns a nonzero value (the value 1 seems ideal). If there is a box still not put away, the function simply returns zero. Once the procedure works correctly, `sokoban` displays a winning message and loads the next level.

Internally, the level is a rectangular array of integers, each of which potentially has one or more of the following bits set:

`WALL`. This location is part of a wall. Walls surround the worker.

`BOX`. This location contains a box that may have to be moved. In some levels, boxes are already in storage locations outside the warehouse.

`STORE`. This is a storage location. There may or may not be an object at this location.

`WORKER`. The worker is at this location.

`SPACE`. This is open space, and may be inside or outside the warehouse.

`HILITE`. This area will be drawn highlighted. You can cause this to be set by calling the `highlight` function, below.

For internal record keeping, other bits may be set as well. They should not have any impact on the interpretation of the contents of the location.

You will find the following methods of help:

- `int height(level *l)`. Returns the height of the warehouse. The rows of the warehouse are numbered starting at zero.
- `int width(level *l)`. Returns the width of the warehouse. The columns of the warehouse are numbered starting at zero.
- `char get(level *l, int r, int c)`. Returns the bits that represent the location at row r, column c of the warehouse. One or more of the bits described above may be set, and they will be set in a consistent manner. Other unrelated bits may also be set.
- `void highlight(level *l, int r, int c)`. This sets the location at row `r` and column `c` to emphasized. This be used to focus the attention on locations where boxes are found in storage and only when a win is certain.

There are many other methods. I would suggest you not make use of them in this assignment. The reasoning is simple: next week we will be converting your `win` method to assembly code and you will want to limit the amount of code you convert.

Otherwise, you are free to edit the `sokoban` code or add new levels.

Please: place your code in a file called `win.c` and turn it in using `turnin -c 237 win.c` before next week's lab.