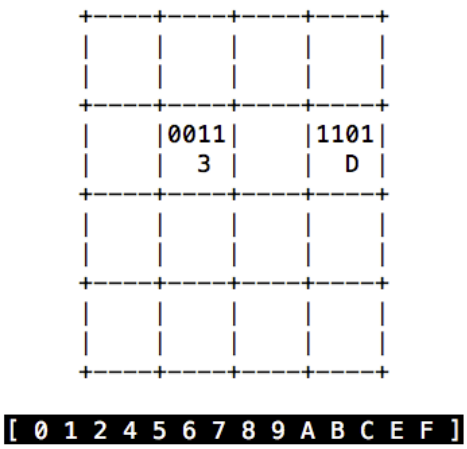
Computer Science 237
Assignment 2
Due next week, *before lab*

The purpose of this week's lab is to finish off a zero-sum game called *Quarto*. I have written most of the code to play this game visually (using a terminal-friendly system called *curses*). What remains is for you to detect a winning board state. Once this `win` method is written correctly, you should be able to play the game and win. You may play the game against another person, or the computer.



A Quarto board, amid play.

Quarto is a game that centers around 16 distinguishable playing pieces, each of which has 4 features. In the commercial version of the game, the pieces have color (dark or light), height (short or tall), outer shape (round or square), and inner shape (solid or hollow). Each physical piece is different from every other piece in at least one feature. As computer scientists, we might imagine that the pieces are the 16 4-bit unsigned integers between 0 and 15, and that each of these integers differs from the others in one or more bit positions.

The play of the game alternates between two players. A turn consists of placing a piece on a 4×4 board in an empty square. Once placed, the piece is never moved. The goal is to be the player who first gets a row, column, or diagonal of pieces that *share* some characteristic. For example, you may finish a diagonal with a piece that has the same 4's bit as the other three. The important principle to remember (and to get over) is that no one "owns" any board pieces; pieces you play can work for you this turn, and against you the next. (Aside: is it possible for a game to be winnerless?)

The application. The game has been written to reinforce your emacs skills. At any time you can give up (`C-G`) or get help (`C-h`). When you're placing pieces, you can move the "cursor" forward (`C-f`), backward (`C-b`), up (`C-p`), or down (`C-n`) to get to a free square.

You can pick up the distribution of this lab from the course kit space as `quarto.tar.gz`:

```
cd ~/cs237
tar xvfz /usr/cs-local/share/cs237/kits/quarto.tar.gz
```

This creates a directory, `quarto`, containing the source and this handout. To build the application, type:

```
make quarto
```

When you make changes to the source, you type `make` again. When you're done, type:

```
make clean
```

removing crust created by development and debugging. To get rid of the executable, returning the directory to a pristine condition, type:

```
make realclean
```

The Task. You are to write one procedure:

```
extern int win(board *b);
```

The `win` method returns true if four pieces appear in a winning position, and false otherwise. At the moment, `win` always returns zero, so it is only possible to place all the pieces on the board, and draw.

Ideally, your implementation of this procedure will scan across the board comparing groups of 4 pieces looking for common digits. If a row, column, or diagonal is found that meets this requirement, you should highlight four pieces that prove a win, and return a true value. Of course, if there is no winning configuration, you should simply return false.

Internally, the pieces are stored in an array that represents the board. If the board position has the value `EMPTY`, that location is not a valid piece, it's empty. If the board has a non-`EMPTY` value, it is a piece and the low 4 bits contain the piece's digits. Other bits in that integer may or may not record internal state that is not important to this task.

You will find the following methods of help:

```
extern int get(board *b, int row, int col);
```

This method returns the piece stored at the indicated board position ($0 \leq row, col < 4$). The piece is encoded as a small integer. If the board square is empty, the value is `EMPTY`. If the board square is a valid, the low 4 bits are the small integer representing the piece.

When you identify four squares that are proof of a win, you should call

```
extern void highlight(board *b, int row, int col);
```

This helps players (who may be surprised) understand how the win occurred.

There are many other methods. I would not suggest you 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 must convert.

The Finish. Turn in your `win.c` using

```
turnin -c 237 win.c
```

before next week's lab.