pset4: Sudoku

Tommy MacWilliam

tmacwilliam@cs50.net

October 2, 2011
Today’s Music

- Ephixa (Zelda Step)
  - Lost Woods
  - Saria’s Song
  - Dragon Roost Island
  - Song of Storms
Today

- Distro code
- ncurses
- moving the cursor
- inputting numbers
- move legality
- inputting blanks
- checking if won
do not try to use the terminal window in gedit
- sudoku needs ALL the pixels!
Sudoku

- goal: every square has a number
  - one of each number in every row
  - one of each number in every column
  - one of each number in every 3x3 block
sudoku.c

- lots, but don’t worry!
- main
  - error checking
  - load board
  - loops to continue to ask user for input
  - handle user input
case-switch

if (c == 'a')
    // do something
else if (c == 'b')
    // do something
else if (c == 'c')
    // do something
case-switch

```c
switch(c) {
    case 'a':
        // do something
        break;
    case 'b':
        // do something
        break;
    case 'c':
        // do something
        break;
}
```
structs allow you to group variables into a single structure

- g is a global struct containing game information
- variables can be of different types!

- g.y, g.x: row and column of cursor

- g.board: 2D array representing board
  - sound familiar?

- g.top, g.left: coordinates of top-left point of board
  (since (0,0) is the top of the terminal window)

- g.number: number of board
sudoku.c

- restart_game()
  - start a new game with the board specified in g.board
- draw_borders()
- draw_grid()
- draw_logo()
- draw_numbers()
sudoku.c

- `show_cursor()`
  - set the position of the cursor based on `g.y` and `g.x`
- `show_banner(char* b)`
  - show the string `b` as a banner
- `hide_banner()`
  - hide the currently-shown banner
Using GDB

- Sudoku takes over the whole terminal, so can’t simply
  gdb ./sudoku
- ./sudoku debug 1 in terminal
- open new terminal tab
- pidof sudoku gives you the unique ID of the process sudoku
- now, gdb ./sudoku #
  - where # is the PID from pidof
Using GDB Like a Hacker

▶ `gdb -tui ./sudoku #`
  ▶ **Text User Interface**
  ▶ displays source above gdb prompt!
ncurses

- library to write GUI (graphic user interface) applications
- still in the terminal, but so much nicer than pset3
ncurses

- pset3: print each row, one at a time
  - once you `printf`, no way of going back
- ncurses: print wherever you want!
  - write a `char` to any `(y, x)` on the screen
Output

- move()
  - move the cursor to the given \((y, x)\) location

- `mvaddch(int y, int x, char c)`
  - move to \((y, x)\), then print \(c\) there
  - don’t forget difference between 1 and ’1’!
Input

- `getch()`
  - get a single character from the user (returns a `char`)
  - `KEY_UP, KEY_DOWN, KEY_LEFT, KEY_RIGHT` represent arrow keys
    - `#define'd` constants by `ncurses`
  - `CTRL('l')` represents Ctrl+l
    - we wrote that one!
pset4: Sudoku
Tommy MacWilliam
Distro Code
ncurses
Cursor
Inputting Numbers
Move Legality
Inputting Blanks
Check if Won
Additional Features

- move cursor
- input number
- move legality
- input blank
- check if won
show_cursor()

- remember, `show_cursor()` moves the cursor based on `g.y` and `g.x`
  - different `g.y` or `g.x`? different cursor location!
  - don’t need to worry about `move` or `mvaddch`
Coordinates

- `g.y` and `g.x` represent the position in the 9x9 board, **NOT** the position on the screen.
Moving

- increment/decrement $g.y$ or $g.x$ based on arrow pressed
  - right now main only handles ‘N’, ‘R’, and CTRL(‘l’), hmmm
  - $(0, 0)$ is top-left, $(8, 8)$ is bottom-right
  - `show_cursor()` takes care of converting position on board to position on screen
- don’t let user move cursor off the board!
TODO

- move cursor
- input number
- move legality
- input blank
- check if won
Inputting Numbers

▶ main also needs to take ’1’ to ’9’ as input
▶ don’t forget about ASCII!
  ▶ ’1’ != 1
  ▶ ’1’ == ’0’ + 1
Updating Board

- just like pset3, \( g.board[i][j] \) contains the number at row \( i \), column \( j \)
  - cursor should always be at \( (g.y, g.x) \)
  - changing \( g.board[i][j] \) changes the number there

- various \texttt{draw_} functions redraw the board based on \( g \)
  - still don’t need to \texttt{mvaddch} yourself!
  - don’t need to redraw everything if only numbers on the board changed!
Changing the Board

- don’t allow user to change numbers that came with the board
  - else Sudoku would be pretty easy
- when game is started, need to remember which numbers were already placed
  - array1 = array2 won’t work :(
- before changing any space, check if that space can be changed (e.g. number was originally blank)
Combining Cases

```c
switch (c) {
    case 'a':
    case 'b':
    case 'c':
        // do something
        break;
    default:
        // do something
        break;
}
```
Design

▶ factor out as much code as possible
  ▶ to try write reusable functions, then actually reuse them!
▶ write your changes in functions, then have existing code call those functions
  ▶ much easier than heavily changing existing functions
pset4: Sudoku

Todo:

- move cursor
- input number
- move legality
- input blank
- check if won
Legality

- after changing a number, need to check legality
  - if move is illegal, tell user via `show_banner()`
- banner does NOT need to persist
  - if I make an illegal move, then a legal move, no more banner
Definition

3 rules for move to be legal:

- number doesn’t already exist in row
- number doesn’t already exist in column
- number doesn’t already exist in 3x3 block
Row and Column

- user just inputted number into `g.board[g.y][g.x]`
  - need to check `g.board[g.y][j]` for `0 \leq j \leq 8`
  - need to check `g.board[i][g.x]` for `0 \leq i \leq 8`
- if number is already found, move is illegal
  - check for **illegal** moves, not **wrong** moves
  - checking for wrong moves is much harder (Hacker Edition!)
Row and Column

Sudoku by John Harvard

playing debug #1

[N]ew Game  [R]estart Game  [Q]uit Game
3x3 Block

- board divided into contiguous 3x3 blocks
3x3 Block

- need to check within defined block, not necessarily 3 columns right and 3 rows down from cursor
- given some \((y, x)\), determine coordinates of top-left of block
  - sounds like a job for division and friends!
  - then, check 3 columns right and 3 columns down
  - only need to check one 3x3 block, not every single one!
TODO

- move cursor
- input number
- move legality
- input blank
- check if won
Blanks

- user must be able to delete number via KEY_BACKSPACE, KEY_DC, ., or 0
  - doing the same thing on multiple cases again? combine them!
- according to draw_numbers(), blank represented by 0 in g.board
Blanks

- can’t delete numbers that came with the board!
  - good thing we wrote a **reusable** function for that!
- is inputting a blank always a legal move?
- can the game be won if I removed a number?
pset4: Sudoku

Tommy MacWilliam

Distro Code
ncurses
Cursor
Inputting Numbers
Move Legality
Inputting Blanks
Check if Won
Additional Features

TODO

- move cursor
- input number
- move legality
- input blank
- check if won
**Won**

- game is won if:
  - every square is filled in
  - every row contains every number
  - every column contains every number
  - every 3x3 block contains every number

- need to check if game is won whenever user makes a move
Won

- only need to look at every row/column once
  - check if every number is found?
  - check if any number is found twice?
- using an array to keep track of what we’ve seen sounds helpful
Won

- need to check every 3x3 square once
  - good thing we already wrote a **reusable** function for that!
- do we need to separately check if every blank is filled?
Design
TODO

pset4: Sudoku

Tommy MacWilliam

Distro Code
ncurses

Cursor
Inputting Numbers
Move Legality
Inputting Blanks
Check if Won

Additional Features

▶ move cursor
▶ input number
▶ move legality
▶ input blank
▶ check if won
Additional Features

▶ don’t forget to implement an additional feature!
  ▶ turn all numbers green when won
  ▶ turn column/row red until mistake is corrected
  ▶ enable cursor wrapping
  ▶ use different color for numbers that came with board
  ▶ keep track of amount of time
  ▶ allow user to undo with U or Ctrl-z