Esse é o **CC50.**
O Curso de Harvard, no Brasil 2012/01

Pset 4: 数独

a ser entregue até: 19:00, sex 23/03

Tenha certeza de que seu código é bem comentado
de forma que a funcionalidade seja aparente apenas pela leitura dos comentários.

Objetivos.

- Aprenda a usar ncurses, uma biblioteca para GUIs.
- Criar designs e implementar softwares em escala maior.
- Virar um Mestre de Sudoku.

Leitura recomendada.

- Seções 1 a 13 de http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/ (em inglês) .

Honestidade Acadêmica.

Todo o trabalho feito no sentido do cumprimento das expectativas deste curso deve ser exclusivamente seu, a não ser que a colaboração seja expressamente permitida por escrito pelo instrutor do curso. A colaboração na realização de Psets não é permitida, salvo indicação contrária definida na especificação do Set.

Ver ou copiar o trabalho de outro indivíduo do curso ou retirar material de um livro, site ou outra fonte, mesmo em parte e apresentá-lo como seu próprio constitui desonestidade acadêmica, assim como mostrar ou dar a sua obra, mesmo em parte, a um outro estudante. Da mesma forma é desonestidade acadêmica apresentação dupla: você não poderá submeter o mesmo trabalho ou similar a este curso que você enviou ou vai enviar para outro. Nem poderá fornecer ou tornar as soluções disponíveis para os Psets para os indivíduos que fazem ou poderão fazer este curso no futuro.

Você está convidado a discutir o material do curso com os outros, a fim de melhor compreendê-lo. Você pode até discutir sobre os Psets com os colegas, mas você não pode compartilhar o código. Em outras palavras, você poderá se comunicar com os colegas em Português, mas você não pode comunicar-se em, digamos, C. Em caso de dúvida quanto à adequação de algumas discussões, entre em contato com o instrutor.

Você pode e deve recorrer à Web para obter referências na busca de soluções para os Psets, mas não por soluções definitivas para os problemas. No entanto, deve-se citar (como comentários) a origem de qualquer código ou técnica que você descubra fora do curso.

Todas as formas de desonestidade acadêmica são tratadas com rigor.

Licença.

Notas:

Seu trabalho neste Pset será avaliado em três quesitos principais:

Exatidão. Até que ponto o seu código é consistente com as nossas especificações e livre de bugs?
Design. Até que ponto o seu código é bem escrito*(*escrito claramente, funcionando de forma eficiente, elegante, e / ou lógica)?
Estilo. Até que ponto o seu código é legível (comentado e indentado, com nomes de variáveis apropriadas)?

Antes de começar.

☐ Entre hoje (melhor que não seja quinta à noite!) e o prazo desse Pset, mantenha-se atento para alguma máquina, programa ou website cuja interface de usuário, você sente, foi mal concebida. Não se concentre muito na estética, já que discordâncias sobre o tema beleza sempre existem. Concentre-se mais no trabalho que a interface demanda dos usuários. Isso faz sentido? Será que ainda é possível otimizar para o caso mais comum? É autoexplicativo? Ele requer que você passe por oito etapas apenas para pegar um trem?

Quando chegar a hora de apresentar esse Pset, pedimos que você relate, no arquivo design.txt que veio junto com esse pset, sobre algo que você notou nesse sentido, algum problema de design ou de implementação de algum sistema. Tudo bem se você não tem muito a dizer sobre a máquina, programa ou site onde você encontrou falhas, algumas poucas linhas são mais do que suficiente. Mas não se esqueça de relatar por que o projeto é ruim e como você poderia melhorá-lo.

Depois de começar.

☐ Abra o Terminal e crie um diretório dentro de cc50 chamado pset4. Extraia todos os arquivos de pset4.zip dentro desse diretório .

Se você entrar na pasta sudoku e listar o conteúdo do seu diretório de trabalho atual (lembra-se como?), você deve ver o seguinte. Se não, não hesite em mandar um e-mail para ajuda@cc50.com.br.

```
debug.bin  l33t.bin  Makefile  n00b.bin  sudoku.c  sudoku.h
```

Bem, isso parece divertido!

Brincando com números (de novo!).

☐ De forma parecida com o Jogo dos Quinze, Sudoku é um jogo de lógica que envolve números. Mas é muito mais interessante. Considere o quebra-cabeça abaixo.

1			**3**	**4**				**5**
					6			**8**
				5			**6**	**3**
		1	**6**		**5**	**8**		**9**
		3		**7**		**5**		
9		**6**	**2**		**1**	**3**		
6	**8**			**2**				
3			**5**					
2				**1**	**9**			**7**

O objetivo do Sudoku é preencher esta grade 9X9 grade de tal forma que cada coluna, cada linha, e cada uma das nove caixas 3X3 contém cada um dos números de 1 a 9 exatamente uma vez. Um monte de estratégias existem, mas a ideia geral é descobrir iterativamente quais números podem ir para onde.

Por exemplo, vamos nos focar em uma das caixas 3X3 que já tem um monte de números e trabalhar com o velho processo da eliminação. Considere a caixa no meio, destacada a seguir.

1			**3**	**4**				**5**
					6			**8**
				5			**6**	**3**
		1	**6**		**5**	**8**		**9**
		3		**7**		**5**		
9		**6**	**2**		**1**	**3**		
6	**8**			**2**				
3			**5**					
2				**1**	**9**			**7**

Vamos ver, essa caixa já tem um 1 e um 2, mas não um 3. Onde poderíamos colocar um 3? Bem, 3 não pode ir na linha de baixo, já que essa linha já possui um três na caixa da direita. E um 3 não pode ir em nenhum dos lados do 7, já que essa linha já tem um 3 na caixa da direita. Aha! O 3 só pode pertencer à linha de cima dessa caixa, caso no qual só há um lugar para colocá-lo! E assim nós preenchemos esse lugar com um 3, de acordo com o abaixo.

1			**3**	**4**				**5**
					6			**8**
				5			**6**	**3**
		1	**6**	**3**	**5**	**8**		**9**
		3		**7**		**5**		
9		**6**	**2**		**1**	**3**		
6	**8**			**2**				
3			**5**					
2				**1**	**9**			**7**

Vamos tentar outro truque agora. Ao invés de descobrir onde um número pode ir, vamos descobrir onde um número não pode ir! Vamos pensar no 9. Todos os pontos onde 9 não pode ir estão realçados em cinza, ou porque já existe outro número lá ou porque já há um 9 na linha, coluna ou caixa em que esse ponto se encontra, de acordo com o abaixo.

1			3	4				5
					6			8
				5			6	3
		1	6	3	5	8		9
		3		7		5		
9		6	2		1	3		
6	8			2				
3			5					
2				1	9			7

Bem, olhe para isso! Parece que encontramos um lar para um 9 dentro dessa caixa no meio, pois nenhum outro quadrado dentro dessa caixa aceitaria um 9!

1			3	4				5
					6			8
				5			6	3
		1	6	3	5	8		9
		3	9	7		5		
9		6	2		1	3		
6	8			2				
3			5					
2				1	9			7

Depois de repetir este tipo de truque uma quantidade suficiente de vezes, e (assumindo que não há um problema entre o sudoku e a cadeira) vamos acabar com a solução abaixo.

1	6	8	3	4	2	7	9	5
5	3	2	7	9	6	1	4	8
7	9	4	1	5	8	2	6	3
4	7	1	6	3	5	8	2	9
8	2	3	9	7	4	5	1	6
9	5	6	2	8	1	3	7	4
6	8	7	4	2	3	9	5	1
3	1	9	5	6	7	4	8	2
2	4	5	8	1	9	6	3	7

Se ainda não ficou muito claro como o jogo é jogado, não hesite em recorrer a Wikipedia.

```
http://pt.wikipedia.org/wiki/Sudoku
```

E se você se interessa pela matemática e pelos algoritmos por trás do jogo, você também pode achar estes artigos interessantes:

```
http://en.wikipedia.org/wiki/Mathematics_of_Sudoku
http://en.wikipedia.org/wiki/Algorithmics_of_sudoku
```

☐ Apenas por diversão, incluímos um monte de quebra-cabeças (e soluções) no final deste Pset. E se você começar a ver que está viciado, note que provavelmente existe uma implementação do Sudoku para o seu celular!

☐ Da mesma forma que lhe demos aquele código para o Jogo dos Quinze, temos um esqueleto do Sudoku para você. O Pset 3 contou com sequências de escape ANSI para implementar a parte gráfica do Jogo dos Quinze, este Pset apresenta uma biblioteca chamada "ncurses" que irá proporcionar uma interface gráfica mais agradável (Graphical User Interface). Seu programa não será parecido com um Mac OS ou um Windows, mas será bem mais sexy do que o Jogo dos Quinze! Além de fazer ficar muito mais fácil a integração de cores ao seu programa (e até mesmo a integração de caixas de diálogo e menus), a biblioteca ncurses permite que você enxergue a tela do seu Terminal como uma grade de `char`s, onde qualquer um deles pode ser alterado sem afetar os outros. Esse tipo de recurso é perfeito para um jogo como o Sudoku, pois você vai ser capaz de adicionar números ao tabuleiro do jogo sem ter que recriar a tela inteira após cada movimento (como você fez com `printf` para o Jogo dos Quinze).

Para conseguir compilar o jogo, então, você precisa primeiro instalar a biblioteca ncurses. Para isso, simplesmente entre no Terminal e digite

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

Por fim digite sua senha e pronto, você acaba de instalar a biblioteca gráfica ncurses. Se você tiver alguma dúvida ou problema com a instalação, não hesite em postar no forum.

☐ Muito bem, agora começa a verdadeira diversão. Você está prestes a implementar um Sudoku em C.

Uma janela típica de Terminal possui 80 caracteres de largura e 24 caracteres de altura (80×24), e ncurses endereça esses pontos através de coordenadas (y, x), onde (0, 0) refere-se ao canto superior esquerdo da sua janela, (0, 79) refere-se ao canto superior direito, (23, 0) refere-se ao canto inferior esquerdo e (23, 79) refere-se ao canto inferior direito.[1] Mesmo que sua janela tenha dimensões menores ou maiores do que essas, a ideia é a mesma. Quando chegar a hora de preencher um espaço em branco no tabuleiro do Sudoku, você simplesmente atualiza o `char` em alguma coordenada (y, x).

Agora vamos falar sobre aquele esqueleto. Essencialmente, nós implementamos uma estrutura estética para o jogo de modo que você possa se concentrar nas partes mais interessantes: o funcionamento do jogo. Na verdade, nós escrevemos o código (e os comentários), de tal forma que você deve ser capaz de aprender um pouco sobre o ncurses e mais simplesmente por lê-lo. E você verá que nós estruturamos o funcionamento do Sudoku de forma muito parecida com como nós fizemos com o jogo dos Quinze. É em `main` que nós temos um loop grande, esperando o input de algum usuário. E as funções separadas servem para executar todo o resto e responder a esse input.

Você também vai descobrir que demos mais código pronto dessa vez, pois este jogo deve ser mais sofisticado (e divertido) do que o último. Não se desespere, é um pouco mais do que 600 linhas. Mas nós sabemos agora que nada disso é tão complicado. Na verdade, se você olhar para cada uma das funções de forma isolada, provavelmente você encontrará umas bastante simples. O que é interessante é que quando você combina tantos blocos de construção, você obtêm alguns resultados bem convincentes. Vamos dar uma olhada.

Se você ainda não está lá, siga para `~/cc50/pset4/` e execute o comando (cada vez mais familiar) abaixo.

```
make
```

Você deve encontrar um novo executável chamado `sudoku` em seu diretório de trabalho atual. Vá em frente e execute-o digitando o comando abaixo.

```
./sudoku
```

Você ainda não vai ver o nosso esqueleto, mas sim o uso do jogo:

```
Uso: sudoku n00b|l33t [#]
```

[1] De forma irritante, sim, é (y, x) e não (x, y).

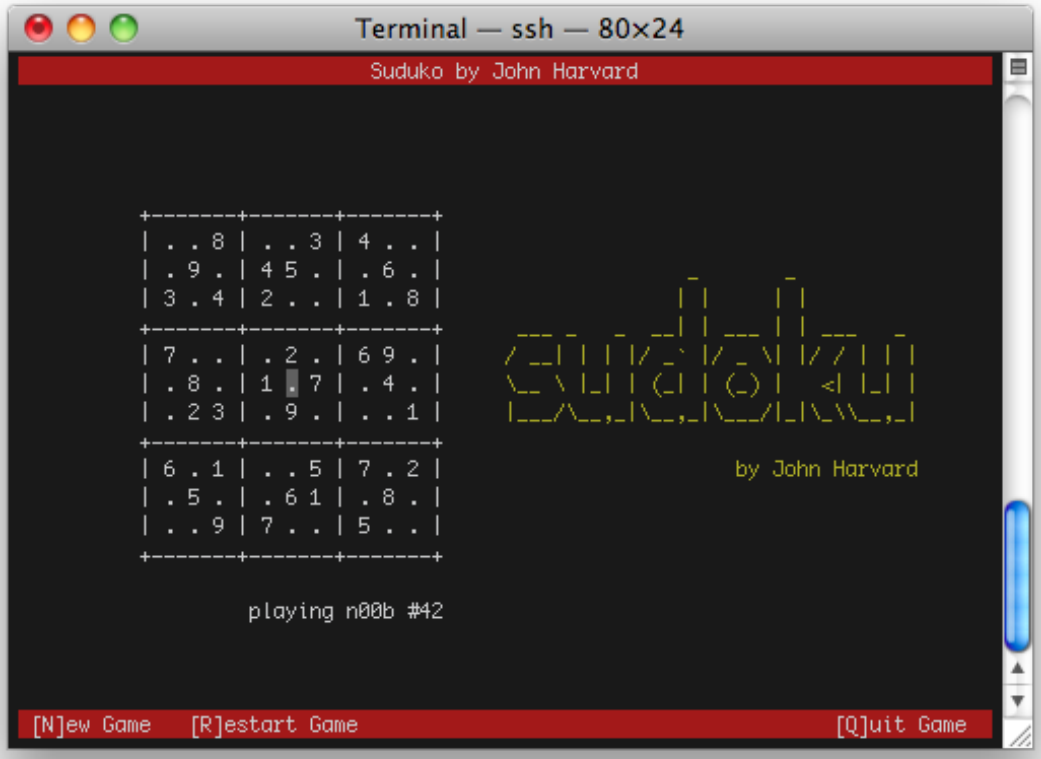
Alem de suportar dois níveis de jogo (n00b[1] e l33t[2]), o nosso esqueleto também vem com 1024 configurações de tabuleiro diferentes para cada nível. Finalmente, se você gostaria de jogar uma configuração n00b pseudoaleatoriamente escolhida, você vai querer executar apenas:[3]

```
./sudoku n00b
```

Como está escrito no menu na parte de baixo do jogo, você pode pressionar **Q** para sair. Agora, se você quiser jogar uma configuração específica (por exemplo n00b #42), talvez uma que você já venceu anteriormente, você pode carregá-la manualmente. Na verdade, vá em frente e execute

```
./sudoku n00b 42
```

para carregar o nosso esqueleto com n00b #42. Você deverá ver uma GUI como essa abaixo.



Observe como, por uma questão de clareza, utilizamos pontos para representar os espaçoes em branco; debaixo do capô, cada um desses espaços em branco está na verdade representado com 0 (um verdadeiro `int`). Então isto tudo é até bem arrumado, mas este esqueleto não tem aquele toque pessoal (para não mencionar o suporte para mover o cursor). As funcionalidades que já

[1] http://en.wikipedia.org/wiki/Newbie
[2] http://en.wikipedia.org/wiki/Leet
[3] n00b tem dois zeros.

estão implementadas são **[N]ew Game**, **[R]estart Game** e **[Q]uit Game**. Vá em frente e aperte **Q** para sair.

Em seguida, abra `sudoku.h` (com Nano ou similar). Você vai encontrar neste arquivo um monte de constantes que são compiladas no programa. Vá em frente e mude, pelo menos, `AUTHOR` para o seu próprio nome. Sinta-se livre para mudar `TITLE` também. Para ver os resultados, salve as alterações e saia. Então reexecute `make` seguido por `sudoku`. Você acaba de roubar o programa para você!

Agora volte a `sudoku.h` e brinque com todas aquelas cores. Acontece que ncurses lida com as cores em pares, em que todos os caracteres têm uma cor de primeiro plano e uma cor de fundo. Por padrão, o primeiro plano dos caracteres é branco e o fundos é preto. Mas é evidente que substituímos os padrões para as bordas e o logotipo do nosso esqueleto. Por enquanto, você vai querer deixar `enum` quieto, mas sinta-se livre para alterar o valor de qualquer constantes cujo nomes começa com `FG_` ou `BG_`. Aqui estão as cores que vem junto com o ncurses:

```
COLOR_BLACK
COLOR_RED
COLOR_GREEN
COLOR_YELLOW
COLOR_BLUE
COLOR_MAGENTA
COLOR_CYAN
COLOR_WHITE
```

Você vai, naturalmente, precisar recompilar o seu jogo para ver as alterações das cores. Não é muito difícil fazer um jogo ficar muito feio, hein?

Ok, agora dê uma rápida olhada no `Makefile`. Ele deve se parecer muito com o do jogo dos Quinze, mas tome nota que nós incluímos a flag `-lncurses`. E tenha certeza que você não mudou os tabs para espaços! Irritantemente, `make` requer que os comandos sejam prefixado com tabs real, não com espaços. Agora, dê uma olhada em, digamos, `n00b.bin`, mas não com `nano` desta vez! Em vez disso, execute o comando abaixo.[1]

```
xxd -b n00b.bin
```

Uau, um monte de números provavelmente passou voando pela sua tela. Você só olhou para o conteúdo de um arquivo binário. Esse arquivo contem um monte de `ints` de 32-bit, 1024 x 81 = 82.944 deles, na verdade, como esse arquivo contém 1.024 configurções n00b e cada uma delas possui 81 números e/ou espaços em branco (para a grade 9x9).[2] Da mesma forma que `l33t.bin` contêm1.024 placas l33t.

Agora que você executou `sudoku` pelo menos uma vez, você também pode ter notado um novo arquivo chamado `log.txt` que não estava lá antes. Você é bem vindo para olhar, mas não precisa

[1] Note que "n00b" é escrito com dois zeros!

[2] Nós poderiamos ter usado `unsigned chars ao invés de ints, pois o Sudoku só precisa dos números de 0 a 9, mas nós decidimos que ints seriam mais simples, apesar do custo adicional de espaço.`

prestar muita atenção pois ele é gerado pelo nosso esqueleto, a fim de facilitar os testes automatizados do seu código.

Tudo bem, nós estamos quase chegando lá. Só mais um arquivo!

☐ Droga, tem um monte de código nesse.

Vá em frente e abra `sudoku.c` (com `nano` ou similar).

A melhor maneira de encarar este Pset é começar a entender este arquivo. Nós vamos te ajudar.

Primeiro tome nota de uma das primeiras linhas do arquivo:

```
#define CTRL(x) ((x) & ~0140)
```

Assim como você pode definir o que conhecemos como constantes com `#`define, você também pode definir "macros", pequenos trechos de código que se comportam um pouco como funções, mas sem a sobrecarga de uma função real. Esta macro específica irá permitir-lhe detectar caracteres com ctrl. O nosso esqueleto já entende ctrl-L, uma combinação de teclas para induzir um redesenho da tela do jogo.

Agora, dê uma olhada na `struct` chamado `g` logo abaixo dessa macro. Pense em uma `struct` como uma caixa que agrupa variáveis relacionadas juntas. Dentro desta `struct` em particular estão vários campos, cada um dos quais pode ser acessado através do operador ponto `(.)` (por exemplo `g.level)`. Pois `g` é uma variável global, assim todos os campos também são efetivamente globais. Verdade seja dita, poderíamos ter definido esses campos como variáveis globais sem usar uma `struct`, assim como fizemos para o Jogo dos Quinze. Mas porque há tantos, todos relacionados a este jogo, decidimos mantê-los juntos em uma estrutura grande chamada `g`. Dessa forma, vai ser ainda mais evidente que estas variáveis não são, de fato, locais.

Depois disso vêm as declarações de funções do nosso esqueleto. Bem como dividimos o jogo dos Quinze em funções cujos nomes descreviam seus papeis no programa, usaremos a mesma abordagem aqui. Mas veremos mais sobre isso mais tarde.

Agora mergulhe em `main`. Embora agora seja melhor não perder muito tempo nessa parte. Ousamos dizer que, para aprender a programar, ler programas escritos por outros é tão importante quanto escrever seus próprios programas, particularmente quando sua tarefa (ou trabalho) é melhorar ou construir por cima de programas feitos por outros. Você vai provavelmente nos agradecer algum dia por todos os comentários no nosso código!

Hmm, há muita coisa acontecendo neste arquivo, mas você não precisa ler todas as linhas (ainda) para ter uma noção do fluxo geral do programa.

Leia, no entanto, todas as linhas de `main`. Afinal, essa é a função que impulsiona todo o programa. E porque os nomes das nossas outras funções dizem o que essas funções fazem, você deve conseguir ler a `main` de cima para baixo e ter uma boa ideia de como o programa está

funcionando no momento. Note, em particular, o loop `do-while` e o `switch` que pegam o input do usuário.

Observe também que implementamos um nível secreto `debug` que tem 9 configurações. Você deve perceber que essas configurações podem facilitar o debug do seu programa, porque elas são rapidamente solucionáveis.
Agora mergulhe em alguma das funções que `main` chama. Uma boa para começar é a `startup` que inicializa o ncurses. Observe como ela chama um monte de outras funções que parecem configurar o ncurses. Embora tenhamos comentado as linhas, você pode querer dar uma olhada na página man de algumas ou todas essas funções, só para você ficar ainda mais confortável com o ncurses.

Em seguida dê uma olhada na `load_board`, a função que carrega uma configuração n00b ou l33t (ou debug) do disco, dependendo do valor, se houver, no `argv`[2] de `main`. Você não precisa entender como `fopen`, `fseek`, `fread`, ou `fclose` funcionam para este conjunto de problemas, mas é bem interessante como elas armazenam todos os bits na memória. O que esta função faz, em última instância, é carregar 81 `ints` no array global chamado `g.board`. Nem é tão difícil!

Vamos ver, agora dê uma olhada em `draw_borders`. É esta função que cria as bordas do jogo. De particular interesse nesta função é como usar ncurses. Observe, por exemplo, que a função primeiro determina as dimensões da janela do Terminal usando uma macro chamado `getmaxyx` (que vem com o ncurses).[1] Ela usa esses máximos para preencher as linhas superior e inferior com algumas cores (e instruções). Observe como a função liga as cores, especificamente ao ligar o atributo `COLOR_PAIR` que chamamos de `PAIR_BORDER` (em `suduko.h`). Ele então começa a desenhar as bordas do jogo se movendo da esquerda para a direita, de coordenada para coordenada, deixando todos os espaços em branco. (Porque nós, pelo menos, definimos `PAIR_BORDER` com um fundo vermelho, os resultados são linhas "em branco" cheias de cor) Mas a próxima função cria algum texto, centrando o título e o autor do seu programa na parte superior usando um pouco de aritmética simples. Em seguida, a função escreve algumas instruções na parte de baixo, antes de desligar a utilização de cor.

Agora dê uma olhada na `draw_grid`. É esta a função que estabelece a arte ASCII que representa o tabuleiro do nosso jogo. Da mesma forma, ela determina primeiro as dimensões da sua janela, em seguida, usa esses valores para determinar as coordenadas do canto superior esquerdo do tabuleiro. (Nós decidimos que queríamos a grade aproximadamente no meio da janela, mas um pouco para a esquerda, e assim chegamos a essas fórmulas por tentativa e erro.) Ao invés de gerar esse tabuleiro caractere por caractere, esta função estabelece strings inteiras (usando a função `mvaddstr` do ncurses). Especificamente, esta função move o cursor para uma coordenada específica e adiciona uma string lá. Em seguida, ela faz isso de novo e de novo (em um loop `for`), a fim de imprimir a maior parte do tabuleiro. (Mais uma vez, a maioria das coordenadas foram determinadas por tentativa e erro) Então pensamos que seria interessante lembrar o usuário do nível e da configuração que ele ou ela está jogando, e assim construímos uma string usando `sprintf`, e depois a adicionamos à tela com uma chamada final de `mvaddstr`.

[1] Você não precisa passar o argumento de `getmaxyx` por referência pois ela é uma macro e não uma função.

Aliás, se você está curioso para saber mais sobre todas essas funções do ncurses, o `man` é seu amigo.[1]

Depois disso dê uma olhada em draw_logo. Observe como ela baseia as suas próprias coordenadas nas do tabuleiro. Note, também, como o nosso logotipo parece distorcido. Isso é porque nós tivemos que escapar algumas das barras invertidas com nossas próprias barras invertidas! Mas sinta-se livre para alterar o logotipo como quiser. Você talvez pode encontrar coisas interessantes nesse site:

`http://www.network-science.de/ascii/`

Agora veja `draw_numbers`. É esta a função que preenche o tabuleiro com os números de `g.board`. Por que tem tanta aritmética nessa função? Temos que admitir que passamos por vários experimentos de tentativa e erro para chegar nisso, mas isso tudo simplesmente garante que os números acabem onde deveriam na tela e não em cima das linhas do tabuleiro.

Agora olhe para `show_banner` e `hide_banner`. Estas funções, ambas muito simples, existem para que você possa mostrar (e esconder) mensagens para os usuários. Ao usar o ncurses, não use `printf`. Coisas ruins vão acontecer.

Falando de `show_banner`, por que não olhamos também para `show_cursor`. Lembre-se que funções como `mvaddch` e `mvaddstr` movem o cursor, a fim de adicionar texto à tela. Isso é um pouco problemático se você quiser usar esse mesmo cursor para jogar o jogo. E por isso é preciso lembrar onde o cursor estava no tabuleiro. Olhar para aquela variável global chamada `g` e você verá como fazemos isso. Esta função `show_cursor` depende daquela `struct` para retornar o cursor para onde ele deveria estar quando a tela for atualizada.

Você não precisa se preocupar muito com a função `handle_signal`. Só saiba que quando janelas do terminal são redimensionadas (de 80X24 a algo maior ou menor), "sinais" são gerados. Nosso código fica atento a esses sinais para que possamos responder a redimensionamentos de tela, recentrando tudo.

Ok, agora só restam `log_move`, `redraw_all`, `restart_game` e `shutdown`. Deixamos esses inteiramente para você! Use a mesma abordagem que utilizamos aqui, passando por cada função, olhando páginas do manual, conforme necessário, e apenas siga em frente após entender o fluxo de funcionamento de cada função.

Então, isso é tudo. Nada mau para 600+ linhas.

☐ Ok, algumas perguntas para você. Crie um arquivo chamado `questions.txt` em `~/cc50/pset4/` usando o `nano` (lembra-se como?) e grave nele as suas respostas às questões abaixo!

I. Repare que main chama `strcmp`. O que significa se `strcmp`, após receber duas strings como argumentos, retornar `0`? (Dica: RTFM)

[1] Pena que nem todas as funções tem a sua página no manual. :-(

II. Como você reescreveria a linha abaixo, extraída de `main`, usando `if` e `else`?

```
int max = (!strcmp(g.level, "debug")) ? 9 : 1024;
```

III. Em que circunstâncias o uso de `sscanf` abaixo, extraído de `main`, retorna 2 em vez de 1?

```
sscanf(argv[2], " %d %c", &g.number, &c)
```

IV. Que atributos de `g` representam as coordenadas onde cursor do usuário está?

V. Qual função (que nós escrevemos) você pode chamar para levar o cursor para essas coordenadas? (Dica: nós lhe dissemos alguns parágrafos atrás)

VI. Perto de que linha de `main` você poderia adicionar elementos `case` para lidar com outras teclas além de N, R, e ctrl-L?

VII. A maioria das placas n00b e l33t têm muitos espaços em branco. Mas quantos espaços em branco tem debug #1? E debug #2? E debug #9?

☐ Debugar `sudoku` com o `gdb` pode ser um pouco complicado, pois ele tem uma GUI e você provavelmente não quer o output do `gdb` bagunçando a interface do jogo. Mas isso não é um problema! Você pode executar o `sudoku` em uma janela do Terminal e o `gdb` em outra! Vamos aprender a fazer isso agora.

No momento, você pvovavelmente só tem uma janela do Terminal aberta agora, abra uma segunda janela e, em uma delas, execute o comando abaixo:

```
./sudoku n00b 42 &
```

`Esse símbolo &` no fim do comando vai dizer à sua shell para rodar `sudoku` em segundo plano, como uma janela minimizada). A implicação é que você não verá a GUI do `sudoku` (ainda), mas você verá o "ID do processo" (aka PID), que é simplesmente um número que identifica o seu programa em execução `sudoku`. Vá em frente e copie esse número (ou, pelo menos, lembre-se dele) e depois maximize o `sudoku` executando este comando:

```
fg
```

Você deverá ver a GUI do jogo. Agora, na sua segunda janela do Terminal, execute

```
gdb ./sudoku #
```

onde `#` é o número que você copiou. Você forneceu o PID de um processo, então `gdb` irá proceder visualizando esse processo para que você possa debugá-lo remotamente (em uma janela separada). Finalmente você deve ver o prompt do `gdb`. Vamos em frente definindo um ponto de quebra em `draw_numbers`, uma função que você já viu antes. Execute o comando abaixo no `gdb`:

```
break draw_numbers
```

Agora, quando o `gdb` ligou-se ao `sudoku` anteriormente, ele supendeu *(*pausou) a execução do jogo. E para isso precisamos contar ao `gdb` que queremos retomar a execução do `sudoku` para que você possa começar a interagir com ele novamente. Execute o comando abaixo no prompt do `gdb`:

```
continue
```

Ok, neste ponto, tanto o `sudoku` quanto o `gdb` estão rodando, cada um em sua própria janela, com `gdb` "assistindo" o que acontece no `sudoku`, apenas esperando que o jogo chegue no ponto de quebra estabelecido: `draw_numbers`. Vamos fazer isso acontecer. Na janela do `sudoku`, aperte **N** para começar um novo jogo. Por causa daquele `do-while` loop (e do `switch`) em `main`, `restart_game` será chamado, o que acabará por chamar `draw_numbers`. De fato, se você olhar agora para a janela do `gdb`, você deve ver que o ponto de quebra foi encontrado! Vá em frente e inicie a execução de `draw_numbers`, linha por linha, executando

```
next
```

no prompt do `gdb` de novo e de novo. Você deve notar que, após cada iteração do loop interno de `draw_numbers`, um novo número é escrito no tabuleiro do `sudoku`. Legal, né? De qualquer forma, a execução de `next` tantas vezes vai ficar entediante, então execute

```
continue
```

que vai dizer ao `gdb` para concluir a execução de `draw_numbers`. Uma vez que você ainda não escreveu nenhum código que precisa ser debugado, vá em frente e saia do `sudoku` agora apertando **Q**. Em seguida, execute

```
quit
```

no prompt do `gdb` para sair do `gdb` também. Ufa. Isso pode ter sido um pouco complicado, mas você vai pegar o jeito bem rápido. E com certeza chegará a apreciar o poder que você tem agora!

Aliás, se você achar que o `sudoku` ou o `gdb` não está(ão) se comportando como deveria(m), você provavelmente pulou algum passo acima ou fez algo na ordem errada. Não se preocupe. Simplesmente saia tanto do `sudoku` quanto do `gdb`, feche as janelas do Terminal, e comece tudo de novo!

Quando chega a hora de procurar algum bug no seu próprio código, lembre-se que você tem agora esse poder!

☐ Agora, o engraçado é que nenhuma das 600+ linhas que nós escrevemos realmente implementa o jogo Sudoku. Mas é aí que você entra! Seu desafio neste Pset é a implementação real deste jogo! Especificamente, você deve implementar todos os RECURSOS NECESSÁRIOS abaixo e qualquer um (1) dos add-ons.

O foco deste Pset é talvez mais design do que qualquer outra coisa, para fazer você pensar um pouco sobre a melhor forma de implementar algum recurso. Dito isto, você está convidado a mudar qualquer aspecto do nosso código se a mudança se adequar melhor ao seu projeto. No entanto, o que você não pode mudar é qualquer coisa relacionada aos logs, incluindo `log_move`. Para que possamos automatizar alguns testes do seu código, o seu programa ainda deve chamar nossa implementação de `log_move` depois de cada tecla apertada pelo usuário, não importa se essa ação realmente alterou de alguma forma o tabuleiro do jogo.

Tudo bem, agora é com você! Aqui está a sua lista de recurso. Saiba que nós listamos os RECURSOS NECESSÁRIOS na ordem em que você provavelmente irá querer implementá-los.

RECURSOS NECESSÁRIOS

☐ No momento, o cursor está "preso" no centro do tabuleiro. Permita aos usuários mover o cursor para o alto, para baixo, para a esquerda e para a direita por meio das setas do teclado. Você é bem vindo a utilizar outras teclas para outros tipos de movimento também (como diagonal, por exemplo), mas você deve utilizar `KEY_UP`, `KEY_DOWN`, `KEY_LEFT` e `KEY_RIGHT`, constantes que são retornadas pela função `getch` do ncurses quando as setas do teclado forem pressionadas (veja a página man de `getch` para outras constantes). Você só deve permitir que o usuário mova o seu cursor para as coordenadas onde existem números ou espaços em branco *(*o cursor deve "pular" sobre as linhas em branco e as linhas de demarcação do tabuleiro), mas você vai perceber que a aritmética já implementada em `show_cursor` ajuda bastante com isso! Mesmo que você esteja tentado a fazer o cursor pular por cima dos números que já vieram com o tabuleiro *(os* que não podem ser alterados), resista à tentação; permita que o cursor vá para qualquer uma das 81 casas do jogo.

☐ Deixar o usuário preencher qualquer casa em branco com um número, movendo o cursor para cima daquela casa e apertando o botão de um número de 1 a 9.

☐ Permita ao usuário alterar um número que ele ou ela já inseriu de volta para um espaço em branco apertando 0, `KEY_BACKSPACE` ou `KEY_DC` ou alterar para algum outro número de 1 a 9 apertando esse número.[1] Mas não permitem a usuário para alterar os números que "veio com" a bordo.

☐ Cada vez que o usuário altera o tabuleiro, verifique se o jogo foi vencido. Se isso acontecer, exiba um banner dando os parabéns e impeça que o usuário altere o tabuleiro.

☐ Cada vez que o usuário altera o tabuleiro, verifique se ele ou ela inseriu um número onde ele é inválido no momento (porque esse mesmo número já existe na mesma coluna, linha ou caixa 3x3). Se ele for inválido, exiba um banner avisando ao usuário do problema, que desaparece no momento em que o usuário muda o tabuleiro de novo (a menos que a mudança também seja um problema, caso em que o usuário deve ser novamente avisado).

[1] Saiba que `KEY_BACKSPACE` e `KEY_DC` geralmente representam a tecla Backspace e a tecla Delete respectivamente, se elas estão presentes. Não se preocupe se as suas teclas Backspace e/ou Delete parecem não funcionar, apesar de você estar utilizando `KEY_BACKSPACE` e/ou `KEY_DC`; alguns teclados enviam sinais diferentes.

ADD-ONS (IMPLEMENTAR PELO MENOS UM (1) DESTES)
Deixe claro nos comentários de `sudoku.c` qual destes Add-ons você implementou.

- ☐ Além de exibir um banner dando os parabéns, faça todos os 81 números ficarem verdes quando o jogo é ganho.

- ☐ Além de avisar o usuário sobre erros óbvios com um banner, pinte a coluna, linha ou caixa 3x3 de vermelho até que o erro seja corrigido.

- ☐ Habilite o cursor a voltar da linha de cima para a de baixo, de baixo para cima, da esquerda para a direita e da direita para a esquerda, se o usuário pressionar uma seta quando o cursor já estiver na borda do tabuleiro.

- ☐ Mostrar os números que vieram com o tabuleiro de uma cor diferente da cor dos números que o usuário digitou.

- ☐ Acompanhe (em segundos) a quantidade de tempo que o usuário está jogando este tabuleiro e permita ao usuário mostrar ou ocultar o relógio, a qualquer momento apertando a tecla T. Certifique-se de parar o relógio no momento em que o jogo for ganho.

- ☐ Permita que o usuário desfaça a última alteração feita na placa apertando U ou ctrl-Z.

☐ Não se esqueça do forum e de `ajuda@cc50.com.br`!

Verificando.

Antes de considerar este Pset feito e entregar, melhor fazer estas perguntas a si mesmo e depois voltar e melhorar o seu código conforme o necessário! Não considere estas perguntas a seguir como uma lista de nossas expectativas, apenas como alguns lembretes úteis. A lista de recursos apresentada acima diz todas as nossas expectativas! Para ser claro, considere as perguntas abaixo como retóricas. E não há a necessidade de respondê-las escrevendo para nós já que todas as respostas devem ser “Sim!”.

- ☐ Você identificou algum projeto com um problema de design e relatou isso em `design.txt`?
- ☐ Você criou `questions.txt` e respondeu aquelas perguntas?
- ☐ Você conseguiu fazer todas as quatro setas do teclado funcionarem?
- ☐ Um usuário pode chegar a todas as 81 casas do tabuleiro através dessas setas?
- ☐ Quando o usuário mexe o cursor, este pula por cima das linhas do tabuleiro automaticamente?
- ☐ O usuário pode preencher espaços em branco com números e mudar números que já foram colocados para outros números?
- ☐ O usuário pode trocar números (que ele ou ela escreveram) de volta para espaços em branco?
- ☐ Você está detectando se o usuário ganhou o jogo?
- ☐ Você impede o usuário de mover o cursor quando o jogo termina?
- ☐ Você avisa ao usuário com um banner quando ele coloca um número em algum lugar que quebre as regras do jogo?
- ☐ Você implementou pelo menos um add-on?

☐ Todos os arquivos para esse pset (e o arquivo questions.txt) estão dentro da pasta pset4?

Como sempre, se você não conseguir responder “sim” a uma ou mais dessas perguntas devido a alguma dificuldade que você está tendo, se dirija ao forum ou mande um e-mail para ajuda@cc50.com.br!

☐ Esse foi o Pset 4.