1. Introduction. This small program will allow an interactive exploration of the I Ching. The outline below sets us up with *ncurses*, and runs a semi-infinite loop updating the display and getting user input. One might–correctly–assume that some user input will cause us to **goto** *done*, terminating the program.

```
#define infinite_loop while (1)
#include <ncurses.h>
#include <stdio.h>
#include <stdlib.h>
  ⟨ Hexagram Data 2 ⟩
  ⟨ Hexagram Functions 3 ⟩
  ⟨ UI Data 17 ⟩
  ⟨ UI Functions 15 ⟩
  int main(int argc, char **argv)
  {
    initscr( );
    cbreak( );
    noecho( );
    curs_set(0);
    ⟨ Read in the hexagram descriptions 24 ⟩;
    ⟨ Display the command list 10 ⟩;
    infinite_loop
    {
      ⟨ Display the current hexagram 9 ⟩;
      ⟨ Parse user input 11 ⟩;
    }
  done:
    endwin( );
    return 0;
  }
```

2. Displaying Hexagrams. We will encode a hexagram as a 6-bit number, with each bit representing a line of the hexagram.

```
⟨ Hexagram Data 2 ⟩ ≡
  typedef unsigned char hexagram;
```

See also sections 5, 7, and 16.

This code is used in section 1.

3. To display a hexagram, we will split off bits one at a time, treating 1 bits as *yang* and 0 bits as *yin*. Here's a function to do that:

```
#define HEX_BOTTOM 12
#define LINE_COORD(line) 12 − ( ( line ) ≪ 1 ) , 2
⟨ Hexagram Functions 3 ⟩ ≡
  void draw_hex(hexagram hex)
  {
    static const char yinyang[][14] ← {"#####␣␣␣#####", "##############"};
    for (int ln ← 0; ln < 6; ++ln) {
      mvprintw(LINE_COORD(ln), yinyang[hex & 1]);
      hex ← hex ≫ 1;
    }
  }
```

See also sections 4 and 6.

This code is used in section 1.

4. We'll want to draw more than just the lines, though. It would be nice to display the *trigram* information on-screen as well. A second function will accomplish this.

```
#define LOWER_TRI_COORD HEX_BOTTOM − 2, 19
#define UPPER_TRI_COORD HEX_BOTTOM − 8, 19
⟨ Hexagram Functions 3 ⟩ +≡
  void draw_trigrams(hexagram hex){
      static const char trigrams[][31] ← ⟨ Trigram Names 23 ⟩;
      mvprintw(LOWER_TRI_COORD, trigrams[hex & ^07]);
      mvprintw(UPPER_TRI_COORD, trigrams[hex ≫ 3]);
      }
```

5. Finally, we'll want to put the name of the hexagram below it, as well as a summary of what it means. Later, we'll cover reading this data in from a file so I can change it without recompiling the program. For now, all that matters is the structure which holds the data.

```
⟨ Hexagram Data 2 ⟩ +≡
  struct {
    char *title;
    char *desc1;
    char *desc2;
  } hex_info[64];
```

6. Once we have that structure, we can make a simple function to display the data:

```
#define TEXT_COORD(ln) HEX_BOTTOM + 2 + ln, 2
#define TEXT_LN(ln, txt) wmove(stdscr, TEXT_COORD(ln));
        clrtoeol( ); mvprintw(TEXT_COORD(ln), txt)
```

⟨ Hexagram Functions 3 ⟩ +≡

```
void draw_title(hexagram hex)
{
  TEXT_LN(0, hex_info[hex].title);
  TEXT_LN(1, hex_info[hex].desc1);
  TEXT_LN(2, hex_info[hex].desc2);
}
```

7. Hexagram History. The user, when browsing hexagrams, may want to explore back-and-forth through the hexagrams they have called up. For instance, if they progressively alter lines, or go through a series of inner hexagrams, they might want to undo and redo their changes. So, we will maintain a circular buffer that contains the last ten hexagrams displayed, and allow the user to go forward and back through their recent history.

```
⟨ Hexagram Data 2 ⟩ +≡
  hexagram history[10] ← {63};
  int hidx ← 0;
```

8. We'll provide some helper macros to navigate the history. Note that `NEW_HEX` does a little error-correcting on the input hexagram, making sure it falls within the valid range of 0 to 63. We will make use of this in the UI code to make the error cases smoother.

```
#define CURRENT_HEX history[hidx]
#define HEX_BACK if (--hidx < 0) hidx ← 9
#define HEX_FORW if (++hidx > 9) hidx ← 0
#define NEW_HEX(hex) HEX_FORW;
        CURRENT_HEX ← (hex) & 63;
```

9. Finally, with the history in place, we can work out how to draw the current hexagram for the user.

```
⟨ Display the current hexagram 9 ⟩ ≡
  hexagram cur_top ← CURRENT_HEX;
  draw_hex(cur_top);
  draw_trigrams(cur_top);
  draw_title(cur_top);
  refresh( );
```

This code is used in section 1.

10. User Commands. We offer the user a number of commands through keyboard interaction. Before the program really starts, we'll put a list of commands at the bottom of the display area.

```
⟨ Display the command list 10 ⟩ ≡
  mvprintw(HEX_BOTTOM + 6, 2,
      "(n)ext/(p)rev␣(f)orw/(b)ack␣(i)nner␣in(v)ert␣(c)hange␣(g)oto␣(q)uit");
```

This code is used in section 1.

11. Here is the code that grabs a key and figures out how to respond:

```
⟨ Parse user input 11 ⟩ ≡
  int cmd ← getch( );
  switch (cmd) {
  case 'n': ⟨ Go to the next King Wen hexagram 19 ⟩;
    break;
  case 'p': ⟨ Go to the previous King Wen hexagram 20 ⟩;
    break;
  case 'q': case 'Q': goto done;
  case 'f': HEX_FORW;
    break;
  case 'b': HEX_BACK;
    break;
  case 'c': ⟨ Change a hex line 14 ⟩;
    break;
  case 'g': ⟨ Go to a specific King Wen hexagram 21 ⟩;
    break;
  case 'i': ⟨ Create the inner hexagram 13 ⟩;
    break;
  case 'v': ⟨ Create the inverted hexagram 12 ⟩;
    break;
  }
```

This code is used in section 1.

12. Inverting a hexagram is just basic bit-twiddling.

```
⟨ Create the inverted hexagram 12 ⟩ ≡
  NEW_HEX(∼cur_top);
```

This code is used in section 11.

13. Creating an inner hexagram is slightly more advanced bit-twiddling.

```
⟨ Create the inner hexagram 13 ⟩ ≡
  NEW_HEX(cur_top ≪ 1 & ∼7 | cur_top ≫ 1 & 7);
```

This code is used in section 11.

14. To change a hex line, we'll need to be able to ask the user which line to change. Then, changing it is a simple exclusive or operation.

```
⟨ Change a hex line 14 ⟩ ≡
  NEW_HEX(cur_top ⊕ 1 ≪ which_line( ) − 1);
```

This code is used in section 11.

15. The *which_line* function is the first question we need to ask the user. We'll define a couple of macros to help us write to and clear a question area. In the case of *which_line*, if the user types invalid input, we just specify a non-existent line 7, which will be ignored by the rest of the code.

```
#define QUESTION move(HEX_BOTTOM + 8, 2)
#define CLR_QUESTION QUESTION; clrtoeol( )
```

⟨ UI Functions 15 ⟩ ≡

```
int which_line(void)
{
  QUESTION;
  printw("Which␣line␣do␣you␣want␣to␣change␣(1␣-␣6)?␣");
  refresh( );
  int num ← getch( ) − '0';
  if (num > 6 ∨ num < 1) num ← 7;
  CLR_QUESTION;
  return num;
}
```

See also sections 18 and 22.

This code is used in section 1.

16. The King Wen Sequence. Nearly all study of the I Ching happens in the context of the "King Wen" arrangement of the hexagrams. So, it is natural to expect that a user will want to explore the hexagrams in this order. We'll define it in a simple array:

```
⟨ Hexagram Data 2 ⟩ +≡
  hexagram king_wen[ ] ← {63, 0, 17, 34, 23, 58, 2, 16, 55, 59, 7, 56, 61, 47, 4, 8, 25, 38, 3, 48, 41, 37, 32, 1, 57, 39,
      33, 30, 18, 45, 28, 14, 60, 15, 40, 5, 53, 43, 20, 10, 35, 49, 31, 62, 24, 6, 26, 22, 29, 46, 9, 36, 52, 11, 13, 44, 54,
      27, 50, 19, 51, 12, 21, 42};
```

17. When we look up where we are in the sequence, we'll cache it in a variable called *cur_wen*. This will save us from scanning *king_wen* to re-orient ourselves most of the time.

```
#define INC_WEN if (++cur_wen > 63) cur_wen ← 0
#define DEC_WEN if (--cur_wen < 0) cur_wen ← 63
⟨ UI Data 17 ⟩ ≡
  int cur_wen ← 0;
```

This code is used in section 1.

18. We'll need a way to make sure our cached *cur_wen* index is current. We do this by comparing it to the current hexagram. If they don't match, we'll need to scan the sequence until we find the current hexagram.

```
⟨ UI Functions 15 ⟩ +≡
  void orient_wen_seq_idx(void)
  {
    if (king_wen[cur_wen] ≡ CURRENT_HEX) return;
    cur_wen ← 0;
    while (king_wen[cur_wen] ≠ CURRENT_HEX) ++cur_wen;
  }
```

19. Now it is easy to move to the next hexagram in King Wen order:

```
⟨ Go to the next King Wen hexagram 19 ⟩ ≡
  orient_wen_seq_idx( );
  INC_WEN;
  NEW_HEX(king_wen[cur_wen]);
```

This code is used in section 11.

20. Ditto for previous:

```
⟨ Go to the previous King Wen hexagram 20 ⟩ ≡
  orient_wen_seq_idx( );
  DEC_WEN;
  NEW_HEX(king_wen[cur_wen]);
```

This code is used in section 11.

21. To go to a specific King Wen hexagram, we just need to look it up:

```
⟨ Go to a specific King Wen hexagram 21 ⟩ ≡
  cur_wen ← which_hex( ) & 63;
  NEW_HEX(king_wen[cur_wen]);
```

This code is used in section 11.

22. The more challenging part of the previous code is hidden in the *which_hex* function, because of the user interaction.

```
⟨ UI Functions 15 ⟩ +≡
  int which_hex(void)
  {
    QUESTION;
    printw("Which␣hexagram␣to␣visit?␣(␣1␣-␣64␣)?␣");
    refresh( );
    int digit1 ← getch( );
    int digit2 ← getch( );
    int ans ← digit1 − '0';
    if (digit2 ≠ '\n') ans ← ans ∗ 10 + digit2 − '0';
    CLR_QUESTION;
    return ans − 1 & 63;
  }
```

23. I Ching Text Data. There are a couple sources of text data left to define. The names of the trigrams are very static, and not very large, so I've just included them in the code.

⟨ Trigram Names 23 ⟩ ≡
```
{"K'UN␣/␣RECEPTIVE␣/␣EARTH␣␣␣␣␣␣",
"CHEN␣/␣AROUSING␣␣/␣THUNDER␣␣␣␣",
"K'AN␣/␣ABYSMAL␣␣␣/␣WATER␣␣␣␣␣␣",
"TUI␣␣/␣JOYOUS␣␣␣␣/␣LAKE␣␣␣␣␣␣␣",
"KEN␣/␣KEEPING␣STILL␣/␣MOUNTAIN",
"LI␣␣␣/␣CLINGING␣␣/␣FIRE␣␣␣␣␣␣␣",
"SUN␣␣/␣GENTILE␣␣␣/␣WIND␣␣␣␣␣␣␣",
"CH'IEN␣/␣CREATIVE␣/␣HEAVEN␣␣␣␣"}
```
This code is used in section 4.

24. The other major source of text data is the file of hexagram names and descriptions. The format of the file is very simple... three lines per hexagram mapping directly to *title*, *desc1*, *desc2* in *hex_info*. The descriptions are in "King Wen" order, so I need to map them to hexagrams as I read them in.

⟨ Read in the hexagram descriptions 24 ⟩ ≡
```
FILE *desc_file ← fopen("iching.txt","r");
⟨ Complain if the description file isn't found 26 ⟩;
int cur;
size_t str_sz;
for (int counter ← 0; counter < 64; ++counter) {
  cur ← king_wen[counter];
  ⟨ Read a string-triple from the file 25 ⟩
}
fclose(desc_file);
```
This code is used in section 1.

25. I allocate 31 bytes for the title, and 71 bytes for each description line. If the file lines are longer, then *getline*( ) is guaranteed to allocate enough memory. But, these should be good guesses.

```
#define READ_DESC(which, sz) str_sz ← sz;
        which ← malloc(sz + 1); getline(&(which), &str_sz, desc_file)
```
⟨ Read a string-triple from the file 25 ⟩ ≡
```
READ_DESC(hex_info[cur].title, 31);
READ_DESC(hex_info[cur].desc1, 71);
READ_DESC(hex_info[cur].desc2, 71);
```
This code is used in section 24.

26. ⟨ Complain if the description file isn't found 26 ⟩ ≡
```
if (¬desc_file) {
  mvprintw(0, 0, "Cannot␣find␣iching.txt!");
  refresh( );
  getch( );
  goto done;
}
```
This code is used in section 24.

27. Index.

ans: 22.
argc: 1.
argv: 1.
cbreak: 1.
CLR_QUESTION: 15, 22.
clrtoeol: 6, 15.
cmd: 11.
counter: 24.
cur: 24, 25.
cur_top: 9, 12, 13, 14.
cur_wen: 17, 18, 19, 20, 21.
CURRENT_HEX: 8, 9, 18.
curs_set: 1.
DEC_WEN: 17, 20.
desc_file: 24, 25, 26.
desc1: 5, 6, 24, 25.
desc2: 5, 6, 24, 25.
digit1: 22.
digit2: 22.
done: 1, 11, 26.
draw_hex: 3, 9.
draw_title: 6, 9.
draw_trigrams: 4, 9.
endwin: 1.
fclose: 24.
fopen: 24.
getch: 11, 15, 22, 26.
getline: 25.
hex: 3, 4, 6, 8.
HEX_BACK: 8, 11.
HEX_BOTTOM: 3, 4, 6, 10, 15.
HEX_FORW: 8, 11.
hex_info: 5, 6, 24, 25.
hexagram: 2, 3, 4, 6, 7, 9, 16.
hidx: 7, 8.
history: 7, 8.
INC_WEN: 17, 19.
infinite_loop: 1.
initscr: 1.
king_wen: 16, 17, 18, 19, 20, 21, 24.
LINE_COORD: 3.
ln: 3, 6.
LOWER_TRI_COORD: 4.
main: 1.
malloc: 25.
move: 15.
mvprintw: 3, 4, 6, 10, 26.
ncurses: 1.
NEW_HEX: 8, 12, 13, 14, 19, 20, 21.
noecho: 1.
num: 15.
orient_wen_seq_idx: 18, 19, 20.
printw: 15, 22.
QUESTION: 15, 22.
READ_DESC: 25.
refresh: 9, 15, 22, 26.
stdscr: 6.
str_sz: 24, 25.
sz: 25.
TEXT_COORD: 6.
TEXT_LN: 6.
title: 5, 6, 24, 25.
trigrams: 4.
txt: 6.
UPPER_TRI_COORD: 4.
which: 25.
which_hex: 21, 22.
which_line: 14, 15.
wmove: 6.
yinyang: 3.

⟨ Change a hex line 14 ⟩ Used in section 11.
⟨ Complain if the description file isn't found 26 ⟩ Used in section 24.
⟨ Create the inner hexagram 13 ⟩ Used in section 11.
⟨ Create the inverted hexagram 12 ⟩ Used in section 11.
⟨ Display the command list 10 ⟩ Used in section 1.
⟨ Display the current hexagram 9 ⟩ Used in section 1.
⟨ Go to a specific King Wen hexagram 21 ⟩ Used in section 11.
⟨ Go to the next King Wen hexagram 19 ⟩ Used in section 11.
⟨ Go to the previous King Wen hexagram 20 ⟩ Used in section 11.
⟨ Hexagram Data 2, 5, 7, 16 ⟩ Used in section 1.
⟨ Hexagram Functions 3, 4, 6 ⟩ Used in section 1.
⟨ Parse user input 11 ⟩ Used in section 1.
⟨ Read a string-triple from the file 25 ⟩ Used in section 24.
⟨ Read in the hexagram descriptions 24 ⟩ Used in section 1.
⟨ Trigram Names 23 ⟩ Used in section 4.
⟨ UI Data 17 ⟩ Used in section 1.
⟨ UI Functions 15, 18, 22 ⟩ Used in section 1.

ICHING

	Section	Page
Introduction	1	1
Displaying Hexagrams	2	2
Hexagram History	7	4
User Commands	10	5
The King Wen Sequence	16	7
I Ching Text Data	23	9
Index	27	10