SOLITUDE

THE COMPLETE GAMES

LEWIS FREEDMAN
KEVIN RYDBERG

TROLL THREAD 2013

SOLITUDE:
The Complete Games

by

Lewis Freedman & Kevin Rydberg

Running C programs on a Mac

You have a couple options:

1) Use gcc if it’s already on your machine.
2) Use XCode if gcc isn’t already on your machine.

You can do #2 even if gcc is already on your machine.

To check if gcc is on your machine, open a terminal window. The directions on how to do this are included below. Then, at the prompt, type gcc, by itself. If it says something to the effect of “gcc - unknown command”, then you don’t have gcc. If it instead says something like, “no input files”, then you do have gcc.

If you have gcc and don’t want to deal with XCode, here’s what you can do:

Using gcc on Mac OS X

1. Type and Save your program in TextEdit.
   a. Type your program in TextEdit.
   b. Save your program as a .c file. To do this, go to the format menu, and then choose “make plain text.” If that option is not there, then your file is being stored as plain text. (The other option is “make rich text format.”) Once you’ve chosen make plain text, then choose save as and enter program.c. Make sure you answer the warning questions in dialog boxes properly.

2. Open Terminal
   a. Open Finder.
   b. Go to your Applications folder.
   c. Double-click on Utilities.
   d. Double-click on Terminal.

3. Use UNIX commands to change to the directory that contains your program.
   a. For example, if your program (named prog1.c) is on the desktop, type “cd Desktop” into the Termial. This will change the current directory to your Desktop.
   b. Type “ls” to see a listing of the files in the currently selected directory. If you see prog1.c included, then you’re ready for the next step.

4. Compile your code.
   a. Make sure your program is saved.
   b. Type “gcc prog1.c” into the Terminal, or “gcc -o prog1 prog1.c”
   c. If there are errors listed, go back into your text editor (such as TextEdit), and fix the problems in your code.
   d. If there aren’t errors, move on to the next step.

5. Run your program.
   a. Once your program has been compiled, a new file names a.out should appear in your current directory. (To check this, type “ls” to see the directory’s list of files). Or, if you compiled the latter way, an executable program named prog1 should appear.

   b. To run your program, type "./a.out" and your program will start. Or, type "./prog1" to run your program. Alternatively, you can run your program by clicking on the icon for the executable created by compiling.
   c. Note: if your program isn't working properly, you can stop it by typing CTRL+C and it will stop abruptly.

Now, if you want to use Xcode, follow these directions:

Mac Directions for getting a nice compiler for OSX
----

1) Go to connect.apple.com
2) Make a free account
3) Login and go to Download
4) Go to Developer Tools on Right hand side
5) Download XCode 2.4

Its a very big file that includes many important programming applications

6) Open the dmg file
7) Install XCodeTools.mpkg

This will be compatible with all C/C++ Compiler for the course

How to use Xcode for the purposes of this class:

1) When you open the program, select, "New Project"
2) It will ask you what type of project and there are many, many choices. Go to the set of options under "Command Line Utliity."
3) In these set of options, select "Standard Tool."
4) When you do this, you should get a new project with a single .c file. Edit this file. Once you are done with that, save it.
5) At this point, the "Build and Run" button should be active. Just click on it. Alternatively, you can go to the build menu and choose build and run.

How to get XCode to run the C99 compiler instead of the C89 one:

1) Open a project
2) Click on "Project" from the taskbar
3) Click on "Edit Project Settings"
4) Click on "Build" button on the top
5) Scroll down to the section titled "GCC -4.0 Language"
6) In the section, click ont he box next to "C Language Dialect"
7) Select either GNU99 or C99

A third option:

Also, if you have XCode, but don't want to compile from it. You can type your programs in it (so you see the indenting and such), and then you can compile them from the command line.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

struct pile {
      char tableau[12];
      char reserve[6];
      char t_number;
      char r_number;
};

struct state {
      struct pile piles[7];
      char foundations[4];
      char stock[24];
      char s_number;
      struct state **branches;
      char b_number;
      struct state *next;
};

struct state *hash[491];


unsigned long int factorial(int n)
{
      if (n == 0)
            return 0;
      if (n == 1)
            return 1;
      else
            return (n * factorial(n - 1));
}


unsigned long int choose(int n, int k)
{

      double product = 1;
      unsigned long int checkifint;
      int i;

      if (k == 0)
            return 1;
      if (n == 0)
            return 0;
      for (i = 1; i <= k; i++)
            product *= (n - (k - i)) / (double) i;
      checkifint = (unsigned long int) product;
      if (product - checkifint > .5)
            return (checkifint + 1);
      else
            return checkifint;
}


/*  selects 13 elements from the first n elements of pool[] according to the
index, and
      stores them in elements[]
      also replaces the first n - 13 elements of pool[] with the unselected
```

```
elements */
void select_elements(char *pool, int n, unsigned long int index, char
*elements)
{
      unsigned long int cutoff = 1;
      unsigned long int block;
      int base = 0;
      int i, j;

      // in this function, to make calculations more straightforward, the
arrays are indexed from 1 to 13

      for (i = 1; i <= 13; i++) {
            while (1) {
                  base++;
                  block = choose((n - base), (13 - i));
                  if (index >= block + cutoff)
                        cutoff += block;
                  else
                        break;
            }
            elements[i - 1] = pool[base - 1];
            pool[base - 1] = -1;
      }

      j = 1;
      for (i = 1; i <= n; i++)
            if (pool[i - 1] != -1) {
                  pool[j - 1] = pool[i - 1];
                  j++;
            }
}


/* permutes the first 13 elements of elements[] according to the index, and
stores them back in elements[] */
void order_elements(unsigned long int index, char *elements)
{
      unsigned long int cutoff = 1;
      unsigned long int block;
      int nshifts, i, j;
      char temp;

      // in this function, to make calculations more straightforward, the
arrays are indexed from 1 to 13

      for (i = 1; i < 13; i++) {
            nshifts = 0;
            while (1) {
                  block = factorial(13 - i);
                  if (index >= cutoff + block) {
                        cutoff += block;
                        nshifts++;
                  }
                  else {
                        if (nshifts > 0) {
                              temp = elements[i - 1];
                              elements[i - 1] = elements[(i + nshifts) - 1];
                              for (j = nshifts; j > 1; j--)
```

```
                                    elements[(i + j) - 1] = elements[(i + (j -
1)) - 1];
                                elements[(i + 1) - 1] = temp;
                        }
                        break;
                }
            }
        }
}


/* stores a random ordering of 52 cards in deck[] according to the seven
indices */
void shuffle(unsigned long int sel_1ndex, unsigned long int order_1ndex,
unsigned long int sel_2ndex,
                        unsigned long int order_2ndex, unsigned long int
sel_3ndex, unsigned long int order_3ndex,
                        unsigned long int order_4ndex, char *deck)
{
        int i;
        char pool[52], elements[13];

        for (i = 0; i <= 51; i++)
                pool[i] = (char)i;

        select_elements(pool, 52, sel_1ndex, elements);
        order_elements(order_1ndex, elements);
        for (i = 0; i <= 12; i++)
                deck[i] = elements[i];

        select_elements(pool, 39, sel_2ndex, elements);
        order_elements(order_2ndex, elements);
        for (i = 0; i <= 12; i++)
                deck[i + 13] = elements[i];

        select_elements(pool, 26, sel_3ndex, elements);
        order_elements(order_3ndex, elements);
        for(i = 0; i <= 12; i++)
                deck[i + 26] = elements[i];

        order_elements(order_4ndex, pool);
        for(i = 0; i <= 12; i++)
                deck[i + 39] = pool[i];
}


/* deals the deck[] and returns a pointer to the first game state */
struct state *deal(const char *deck)
{
        int i, j, k, l, hashvalue;
        struct state *pstate = (struct state *)malloc(sizeof(struct state));

        for (i = 0; i <= 3; i++)
                pstate->foundations[i] = -1;

        pstate->b_number = -1;

        for (i = 0; i <= 6; i++) {
                pstate->piles[i].t_number = 0;
                pstate->piles[i].r_number = (char)i - 1;
            for (j = i, l = i, k = 0; j >= 0; j--) {
                  if (j == 0)
                        pstate->piles[i].tableau[0] = deck[l];
                  else {
                        pstate->piles[i].reserve[k] = deck[l];
                        k++;
                  }
                  l += 6 - (i - j);
            }
      }

      for (i = 0; i <= 23; i++)
            pstate->stock[i] = deck[i + 28];
      pstate->s_number = 23;

      hashvalue = 0;
      for (i = 0; i <= 3; i++)
            hashvalue += (pstate->foundations[i] >= 0) ? (int)pstate-
>foundations[i] : 0;
      for (i = 0; i <= 6; i++)
            hashvalue += (pstate->piles[i].t_number >= 0) ? (int)pstate-
>piles[i].tableau[(int)pstate->piles[i].t_number] : 0;
      hash[hashvalue] = pstate;
      pstate->next = NULL;

      return pstate;
}


/* returns 0 if *p1state and *p2state have identical piles, foundations, and
stock, otherwise returns 1 */
int compare(const struct state *p1state, const struct state *p2state)
{
      int i, j;

      for (i = 0; i <= 3; i++)
            if (p1state->foundations[i] != p2state->foundations[i])
                  return 1;

      for (i = 0; i <= 6; i++) {
            if (p1state->piles[i].t_number != p2state->piles[i].t_number)
                  return 1;
            for(j = p1state->piles[i].t_number; j >= 0; j--)
                  if (p1state->piles[i].tableau[j] != p2state->piles[i].
tableau[j])
                        return 1;
            if (p1state->piles[i].r_number != p2state->piles[i].r_number)
                  return 1;
            for(j = p1state->piles[i].r_number; j >= 0; j--)
                  if (p1state->piles[i].reserve[j] != p2state->piles[i].
reserve[j])
                        return 1;
      }

      if (p1state->s_number != p2state->s_number)
            return 1;
      for (i = 0; i <= p1state->s_number; i++)
            if (p1state->stock[i] != p2state->stock[i])
                  return 1;
```

```
	return 0;
}


/* returns 0 if *pstate has already been tried, otherwise inserts pstate into
hash table and returns 1*/
int lookup_branch(struct state *pstate)
{
	int i, hashvalue;
	struct state *pnstate;

	hashvalue = 0;
	for (i = 0; i <= 3; i++)
		hashvalue += (pstate->foundations[i] >= 0) ? (int)pstate-
>foundations[i] : 0;
	for (i = 0; i <= 6; i++)
		hashvalue += (pstate->piles[i].t_number >= 0) ? (int)pstate-
>piles[i].tableau[(int)pstate->piles[i].t_number] : 0;

	for (pnstate = hash[hashvalue]; pnstate != NULL; pnstate= pnstate->next)
		if (compare(pnstate, pstate) == 0)
			return 0;

	pstate->next = hash[hashvalue];
	hash[hashvalue] = pstate;
	return 1;
}


int t_to_f(int from, int index, int to, struct state *pstate)
{
	pstate->foundations[to] = pstate->piles[from].tableau[index];
	if (--pstate->piles[from].t_number < 0 && pstate->piles[from].r_number
>= 0)
		pstate->piles[from].tableau[(int)++pstate->piles[from].t_number] =
pstate->piles[from].reserve[(int)pstate->piles[from].r_number--];

	return lookup_branch(pstate);
}


int t_to_t(int from, int index, int to, struct state *pstate)
{
	int i, j;

	for (i = index, j = 0; i <= (int)pstate->piles[from].t_number; i++, j++)
{
		pstate->piles[to].tableau[(int)++pstate->piles[to].t_number] =
pstate->piles[from].tableau[i];
	}
	if ((pstate->piles[from].t_number -= (char)j) < 0 && pstate-
>piles[from].r_number >= 0)
		pstate->piles[from].tableau[(int)++pstate->piles[from].t_number] =
pstate->piles[from].reserve[(int)pstate->piles[from].r_number--];

	return lookup_branch(pstate);
}


int s_to_f(int from, int index, int to, struct state *pstate)
```

```
{
      int i;

      pstate->foundations[to] = pstate->stock[index];

      for (i = index; i <= (int)pstate->s_number - 1; i++)
            pstate->stock[i] = pstate->stock[i + 1];
      pstate->s_number--;

      return lookup_branch(pstate);
}


int s_to_t(int from, int index, int to, struct state *pstate)
{
      int i;

      pstate->piles[to].tableau[(int)++pstate->piles[to].t_number] = pstate-
>stock[index];

      for (i = index; i <= (int)pstate->s_number - 1; i++)
            pstate->stock[i] = pstate->stock[i + 1];
      pstate->s_number--;

      return lookup_branch(pstate);
}


struct state *next_state(int from, int index, int to, int (*move)(int , int ,
int , struct state * ), struct state *pstate)
{
      int i, j;
      struct state *pnstate = (struct state *)malloc(sizeof(struct state));

      pnstate->next = NULL;

      for (i = 0; i <= 3; i++)
            pnstate->foundations[i] = pstate->foundations[i];

      pnstate->b_number = -1;

      for (i = 0; i <= 6; i++) {
            pnstate->piles[i].t_number = pstate->piles[i].t_number;
            for(j = (int)pnstate->piles[i].t_number; j >= 0; j--)
                  pnstate->piles[i].tableau[j] = pstate->piles[i].tableau[j];
            pnstate->piles[i].r_number = pstate->piles[i].r_number;
            for(j = (int)pnstate->piles[i].r_number; j >= 0; j--)
                  pnstate->piles[i].reserve[j] = pstate->piles[i].reserve[j];
      }

      pnstate->s_number = pstate->s_number;
      for (i = 0; i <= (int)pnstate->s_number; i++)
            pnstate->stock[i] = pstate->stock[i];

      if ((*move)(from, index, to, pnstate))
            return pnstate;
      else {
            free(pnstate);
            return NULL;
      }
```

```
}


int find_branches(struct state *pstate)
{
	int i, j, k;
	char icard, jcard, kcard;
	struct state *temp;

	for (i = 0; i <= 6; i++) {
		if (pstate->piles[i].t_number < 0)
			continue;
		icard = pstate->piles[i].tableau[(int)pstate->piles[i].t_number];
		if (icard % 13 == pstate->foundations[(int)icard / 13] % 13 + 1) {
			if ((temp = next_state(i, pstate->piles[i].t_number, icard / 
13, t_to_f, pstate))) {
				pstate->b_number++;
				if (pstate->b_number == 0)
					pstate->branches = (struct state **) 
malloc(sizeof(struct state *));
				else
					pstate->branches = (struct state **) 
realloc(pstate->branches, (pstate->b_number + 1) * sizeof(struct state *));
				pstate->branches[(int)pstate->b_number] = temp;
			}
			if (icard % 13 == 0)
				continue;
		}
		for (j = 0; j <= 6; j++) {
			if (j == i)
				continue;
			if (pstate->piles[j].t_number < 0) {
				if (pstate->piles[i].tableau[0] % 13 == 12 && pstate-
>piles[i].r_number >= 0) {
					if ((temp = next_state(i, 0, j, t_to_t, 
pstate))) {
						pstate->b_number++;
						if (pstate->b_number == 0)
							pstate->branches = (struct state **) 
malloc(sizeof(struct state *));
						else
							pstate->branches = (struct state **) 
realloc(pstate->branches, (pstate->b_number + 1) * sizeof(struct state *));
						pstate->branches[(int)pstate->b_number] = 
temp;
					}
				}
				continue;
			}
			jcard = pstate->piles[j].tableau[(int)pstate->piles[j].t_
number];
			for (k = 0; k >= 0; k--) {
				kcard = pstate->piles[i].tableau[k];
				if (jcard % 13 - kcard % 13 == 1 && (jcard / 13 + 
kcard / 13) % 2) {
					if ((temp = next_state(i, k, j, t_to_t, 
pstate))) {
						pstate->b_number++;
						if (pstate->b_number == 0)
							pstate->branches = (struct state **) 
malloc(sizeof(struct state *));
                                    else
                                          pstate->branches = (struct state **)
realloc(pstate->branches, (pstate->b_number + 1) * sizeof(struct state *));
                                    pstate->branches[(int)pstate->b_number] =
temp;
                              }
                              break;
                        }
                  }
            }
      }
      for (i = 2; i <= (int)pstate->s_number + 2; i += 3) {
            if (i > (int)pstate->s_number)
                  i = (int)pstate->s_number;
            icard = pstate->stock[i];
            if (icard % 13 == pstate->foundations[(int)icard / 13] % 13 + 1) {
                  if ((temp = next_state(-1, i, (int)icard / 13, s_to_f,
pstate))) {
                        pstate->b_number++;
                        if (pstate->b_number == 0)
                              pstate->branches = (struct state **)
malloc(sizeof(struct state *));
                        else
                              pstate->branches = (struct state **)
realloc(pstate->branches, (pstate->b_number + 1) * sizeof(struct state *));
                        pstate->branches[(int)pstate->b_number] = temp;
                  }
                  if (icard % 13 == 0)
                        continue;
            }
            for (j = 0; j <= 6; j++) {
                  if (pstate->piles[j].t_number < 0) {
                        if (icard % 13 == 12) {
                              if ((temp = next_state(-1, i, j, s_to_t,
pstate))) {
                                    pstate->b_number++;
                                    if (pstate->b_number == 0)
                                          pstate->branches = (struct state **)
malloc(sizeof(struct state *));
                                    else
                                          pstate->branches = (struct state **)
realloc(pstate->branches, (pstate->b_number + 1) * sizeof(struct state *));
                                    pstate->branches[(int)pstate->b_number] =
temp;
                              }
                        }
                  continue;
                  }
                  jcard = pstate->piles[j].tableau[(int)pstate->piles[j].t_
number];
                  if (jcard % 13 - icard % 13 == 1 && (jcard / 13 + icard /
13) % 2) {
                        if ((temp = next_state(-1, i, j, s_to_t, pstate))) {
                              pstate->b_number++;
                              if (pstate->b_number == 0)
                                    pstate->branches = (struct state **)
malloc(sizeof(struct state *));
                              else
                                    pstate->branches = (struct state **)
```

```
realloc(pstate->branches, (pstate->b_number + 1) * sizeof(struct state *));
                              pstate->branches[(int)pstate->b_number] = temp;
                        }
                  }
            }
      }

return pstate->b_number;
}


int play_game(struct state *pstate)
{
      int i; //, j, k;

      if (find_branches(pstate) < 0) {
            for (i = 0; i <= 3; i++)
                  if (pstate->foundations[i] % 13 != 12)
                        return 0;
            return 1;
      }

      for (i = 0; i <= (int)pstate->b_number; i++)
            if (play_game(pstate->branches[i]) == 1)
                  return 1;

      free(pstate->branches);
      return 0;
}


int main(void)
{
      char deck[52];
      unsigned long int a, b, c, d, e, f, g;
      struct state *origin;

      srand(time(NULL));
      rand();

      a = 1 + (unsigned long int)(choose(52, 13) * (rand() / (RAND_MAX +
1.0)));
      b = 1 + (unsigned long int)(factorial(13) * (rand() / (RAND_MAX +
1.0)));
      c = 1 + (unsigned long int)(choose(39, 13) * (rand() / (RAND_MAX +
1.0)));
      d = 1 + (unsigned long int)(factorial(13) * (rand() / (RAND_MAX +
1.0)));
      e = 1 + (unsigned long int)(choose(26, 13) * (rand() / (RAND_MAX +
1.0)));
      f = 1 + (unsigned long int)(factorial(13) * (rand() / (RAND_MAX +
1.0)));
      g = 1 + (unsigned long int)(factorial(13) * (rand() / (RAND_MAX +
1.0)));

      shuffle(a, b, c, d, e, f, g, deck);

      origin = deal(deck);

      printf("\nplaying (this may take a while)...\n");
      if (play_game(origin) == 1)
            printf("The shuffle indexed by (%lu, %lu, %lu, %lu, %lu, %lu, %lu)
can be won.\n", a, b, c, d, e, f, g);
      else
            printf("The shuffle indexed by (%lu, %lu, %lu, %lu, %lu, %lu, %lu)
promises no future is a loser and can not be won.\n", a, b, c, d, e, f, g);
      return 0;
}
```