```c
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sodium.h> // Must be compiled with -lsodium as an argument
#include <unistd.h>

#define DECK_SIZE 52
#define BLACKJACK 21
#define ACE_HIGH 11
#define FACE_VAL 10
#define CARD_HEIGHT 7
#define MAX_HAND 5
#define DEALER_MIN 16
#define PAYOUT 1.5
#define UNFLIPPED "\e[0;31m\e[44mXXXXXXXXXX\e[0m"
#define WHITE "\e[0;47m"
#define BLACK_ON_WHITE "\e[0;30m\e[47m"
#define RED_ON_WHITE "\e[0;31m\e[47m"
#define RESET "\e[0m"

const char *ranks[] = {"A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", };
const char *suites[] = {"♣","♦","♥", "♠"};
int cards[DECK_SIZE];
int cards_size = DECK_SIZE;

void
print_unflipped() {
        fputs(UNFLIPPED, stdout);
        putchar('\t');
}

char *
get_color(int suite) {
        if (suite == 1 || suite == 2)
                return RED_ON_WHITE;
        return BLACK_ON_WHITE;
}

char *
get_spaces(int rank) {
        if (rank == -1)
                return  "           ";
        if (!strcmp(ranks[rank], "10"))
                return  "         ";
        return "          ";
}

int
get_rank(int n) {
        n--;
        return n % (sizeof(ranks) / sizeof(ranks[0]));
}

int
get_suite(int n) {
        n--;
        return (float) n / DECK_SIZE * (sizeof(suites) / sizeof(suites[0]));
}

void
print_footer(int card) {
        if (card == 0) {
                print_unflipped();
        } else {
                int suite = get_suite(card), rank = get_rank(card);
                printf("%s%s%s%s\t", get_color(suite), get_spaces(rank), ranks[rank], RESET);
        }

}

void
print_body(int card) {
        if (card == 0)
                print_unflipped();
        else
                printf("%s%s%s\t", WHITE, get_spaces(-1), RESET);
}

void
print_middle(int card) {
        if (card == 0) {
                print_unflipped();
```

```
84          } else {
85                  int suite = get_suite(card);
86                  printf("%s     %s     %s\t", get_color(suite), suites[suite], RESET);
87          }
88  }
89
90  void
91  print_header(int card) {
92          if (card == 0) {
93                  print_unflipped();
94          } else {
95                  int suite = get_suite(card), rank = get_rank(card);
96                  printf("%s%s%s%s\t", get_color(suite), ranks[rank], get_spaces(rank), RESET);
97          }
98  }
99
100 void
101 print_hand(int *hand, int lo, int to) {
102         putchar('\n');
103         for (int i = lo; i < to; i++) {
104                 print_header(hand[i]);
105         }
106         putchar('\n');
107         for (int i = 0; i < CARD_HEIGHT / 2; i++) {
108                 for (int i = lo; i < to; i++)
109                         print_body(hand[i]);
110                 putchar('\n');
111         }
112         for (int i = lo; i < to; i++)
113                 print_middle(hand[i]);
114         putchar('\n');
115         for (int i = 0; i < CARD_HEIGHT / 2; i++) {
116                 for (int i = lo; i < to; i++)
117                         print_body(hand[i]);
118                 putchar('\n');
119         }
120         for (int i = lo; i < to; i++)
121                 print_footer(hand[i]);
122         putchar('\n');
123         putchar('\n');
124 }
125
126 void
127 print_cards(int *hand, int size) {
128         for (int i = MAX_HAND; i <= size; i += MAX_HAND)
129                 print_hand(hand, i - MAX_HAND, i);
130         int leftover = size % MAX_HAND;
131         if (leftover > 0)
132                 print_hand(hand, size - leftover, size);
133 }
134
135 void
136 fill_deck() {
137         for (int i = 0; i < DECK_SIZE; i++)
138                 cards[i] = i + 1;
139 }
140
141 void
142 shuffle_deck() {
143         uint32_t j, tmp;
144         for (int i = DECK_SIZE - 1; i > 0; i--) {
145                 j = randombytes_uniform(i+1);
146                 tmp = cards[i];
147                 cards[i] = cards[j];
148                 cards[j] = tmp;
149         }
150 }
151
152 void
153 print_deck() {
154         print_cards(cards, DECK_SIZE);
155 }
156
157
158 void
159 help() {
160         printf("CARDS -- Authored by Mitch Feigenbaum\n");
161         printf("Options:\n");
162         printf("\t-b\t\tPlay an interactive round of Blackjack\n");
163         printf("\t-c<n>\t\tPrint a specific card from an ordered deck\n");
164         printf("\t-r\t\tPrint a random card\n");
165         printf("\t-o\t\tPrint an ordered deck\n");
166         printf("\t-s\t\tPrint a random deck.\n");
```

```
	printf("\t-h\t\tPrint this help message\n");
}

typedef struct
Gamblers {
	int hand[BLACKJACK];
	int balance;
	int n_cards;
} Gambler;

char
action() {
	char choice;
	printf("[H]it\t[S]tay\t[F]old\t[Q]uit\t");
	do {
	scanf("%c", &choice);
	while(getchar() != '\n');
	} while (choice != 'H'
			&& choice != 'S'
			&& choice != 'F'
			&& choice != 'Q'
			&& choice != 'h'
			&& choice != 's'
			&& choice != 'f'
			&& choice != 'q');
	return choice;
}

int
card_val(int card) {
	int rank = get_rank(card);
	if (rank >= FACE_VAL)
		return FACE_VAL;
	else if (rank > 1)
		return rank + 1;
	else
		return 1;
}

int
hand_sum(int *hand, int size) {
	int sum = 0, aces = 0, val;
	for (int i = 0; i < size; i++) {
		val = card_val(hand[i]);
		if (val == 1)
			aces++;
		else
			sum += val;
	}
	if (aces > 0)
		sum += aces - 1;
	if (sum + ACE_HIGH <= BLACKJACK && aces > 0)
		sum += ACE_HIGH;
	else if (aces > 0)
		sum++;
	return sum;
}

int
get_bet(Gambler *g) {
	int bet;
	do {
		putchar('$');
		scanf("%d", &bet);
		while(getchar() != '\n');
	} while (bet <= 0 || bet > g->balance);
	g->balance -= bet;
	return bet;
}

void
deal(Gambler *g) {
	cards_size--;
	g->hand[g->n_cards] = cards[cards_size];
	g->n_cards++;
}

void
contest(Gambler *dealer, Gambler *player, int bet) {
	printf("Dealer's turn:\n");
	cards_size--;
	dealer->hand[0] = cards[cards_size];
	print_cards(dealer->hand, dealer->n_cards);
```

```
250         while(hand_sum(dealer->hand, dealer->n_cards) < DEALER_MIN) {
251                 deal(dealer);
252                 print_cards(dealer->hand, dealer->n_cards);
253         }
254         int dealer_sum = hand_sum(dealer->hand, dealer->n_cards);
255         int player_sum = hand_sum(player->hand, player->n_cards);
256         if (dealer_sum > BLACKJACK || dealer_sum < player_sum) {
257                 printf("You win! (+$%d)\n", (int) (bet * PAYOUT - bet));
258                 bet *= PAYOUT;
259                 player->balance += bet;
260         } else if (dealer_sum == player_sum) {
261                 player->balance += bet;
262                 puts("Draw.");
263         } else {
264                 printf("You lose! (-$%d)\n", bet);
265         }
266 }
267
268
269 void
270 game(Gambler *dealer, Gambler *player) {
271         cards_size = DECK_SIZE;
272         shuffle_deck();
273         printf("Make a bet (balance: %d)\t", player->balance);
274         int bet = get_bet(player);
275         for(int i = 0; i < 2; i++) {
276                 deal(player);
277         }
278         dealer->hand[0] = 0;
279         dealer->n_cards++;
280         deal(dealer);
281         printf("Dealer hand:\n");
282         print_cards(dealer->hand, dealer->n_cards);
283         printf("Player hand:\n");
284         print_cards(player->hand, player->n_cards);
285         int over = 0, fold = 0, stay = 0;
286         printf("Your turn:\n");
287         do {
288                 switch(action()) {
289                 case 'H':
290                 case 'h':
291                         deal(player);
292                         print_cards(player->hand, player->n_cards);
293                         over = hand_sum(player->hand, player->n_cards) > BLACKJACK;
294                         break;
295                 case 'S':
296                 case 's':
297                         stay = 1;
298                         break;
299                 case 'F':
300                 case 'f':
301                         fold = 1;
302                         break;
303                 default:
304                         break;
305         }
306         } while (!over && !fold && !stay);
307         if (over) {
308                 printf("You've gone bust! (-%d)\n", bet);
309         } else if (fold) {
310                 bet /= 2;
311                 printf("You've surrendered this round. (-%d)\n", bet);
312                 player->balance += bet;
313         } else if (stay) {
314                 contest(dealer, player, bet);
315         } else {
316                 puts("It appears an error has occurred, what a shame.");
317         }
318 }
319
320 void
321 blackjack() {
322         Gambler player = {.balance = 100, .n_cards = 0};
323         Gambler bot = {.balance = 100, .n_cards = 0};
324         while (player.balance > 0) {
325                 game(&bot, &player);
326                 player.n_cards = 0;
327                 bot.n_cards = 0;
328         }
329 }
330
331 int
332 main(int argc, char **argv) {
```

```
333         fill_deck();
334         randombytes_stir();
335         if (argc == 1) {
336                 help();
337                 return 1;
338         }
339         int c;
340         int card;
341         while ((c = getopt(argc, argv, "bc:rosh")) != -1)
342                 switch(c) {
343                         case 'b':
344                                 blackjack();
345                                 break;
346                         case 'c':
347                                 card = abs(atoi(optarg));
348                                 if (card > DECK_SIZE) {
349                                         fprintf(stderr, "%s: option -c requires an argument within the range of a
standard deck (1-52).\n", argv[0]);
350                                         return 1;
351                                 }
352                                 print_cards(&card, 1);
353                                 break;
354                         case 'r':
355                                 card = randombytes_uniform(DECK_SIZE + 1);
356                                 print_cards(&card, 1);
357                                 break;
358                         case 'o':
359                                 fill_deck();
360                                 print_deck();
361                                 break;
362                         case 's':
363                                 shuffle_deck();
364                                 print_deck();
365                                 break;
366                         case 'h':
367                                 help();
368                                 break;
369                         case '?':
370                                 return 1;
371                         default:
372                                 abort();
373                 }
374         for (int i = optind; i < argc; i++)
375                 fprintf(stderr, "non-option argument: %s\n", argv[i]);
376         return 0;
377 }
378
```