EE231002 Introduction to Programming

Lab13. Blackjack Strategy

Due: Dec. 26, 2015

Using simple programming we can try out different strategies and see the effects quickly. In contrast to the real life experiments, software experiments are much faster and, if all parameters are accounted for, can be very accurate as well.

In this lab, you will need to implement 4 functions that enable you to play Blackjack card game automatically. If you are not familiar with the Blackjack game, you can visit the website, `http://en.wikipedia.org/wiki/Blackjack`, or simply play with the demo version on workstations by the following command.

```
$ ~ee231002/lab13/demo
```

This demo version has an option to show the cumulative probability of the next cards when you are making the decision to **stand** or **hit**. To try this, type in

```
$ ~ee231002/lab13/demo c
```

The cumulative probabilities from Ace on are shown on the screen. Will these information help you to win your game? On a computer, you can get and use the same information to improve your winning percentage. However, there are still the issue of what is the strategy of using these information.

The 4 functions you will write are

1. `int play4_bet_init(int credit);`
   This function places bet for each round of Blackjack game. Input parameter `credit` is the amount of credits available to you for the current round. And, the return value is the bet you place.

2.
   ```
   int play4_hit_or_stand(const struct CARD mycard[],int Nmycard,
                          const struct CARD dealercard[],int Ndealercard);
   ```
   This function determines if you want to **hit** (to request for more cards) or **stand** (no more card) by returning 1 (for **hit**) or 0 (for **stand**). (The third option for **double** can also be selected by returning 2.) Two sets of input are given: `mycard` and `Nmycard` are the cards and the number of cards in your hand; and `dealercard` and `Ndealercard` are the cards and the number of cards that the dealer possesses. Note that you can see only one of the dealer's card. The other card (`hole`) is facing down and is not available to you. The structure of the `CARD` and some useful functions are declared in the file `card.h`.

3. `void play4_shuffle();`
   This function is called when the dealer reshuffles the decks. In this game, we use 4 decks of playing cards. Thus, total number of cards available is $4 \times 52 = 208$. When the number of cards is less than 20, before dealer starts dealing cards for that particular run, the cards are reshuffled. This function needs no return values.

4. `void play4_deal_one_card(const struct CARD cd);`
   This function is called whenever a card is dealt by the dealer. Again, it needs no return values. This function and the `shuffle` function enable you to count the cards.

Example of the four functions are given at the end of this pdf file. You can start by copying all the files in the ∼ee231002/lab13 directory to your local directory, and create a player4.c file that contains the codes shown at the end of this file. Once that is done, then you can compile your codes by

```
$ gcc -o bj *.o player4.c
```

This would produce the program bj, and you can run the program by typing

```
$ ./bj
```

Which is very similar to the demo version except now the computer is making decision for placing bet and hit/stand.

Notes.

1. Create a directory **lab13** and use it as the working directory.
2. You will need to turn in the file **player4.c** only.
3. The first few lines of your program should be comments as the following.

   ```
   /* EE231002 Lab13. Blackjack Strategy
      ID, Name
      Date:
   */
   ```

4. After you finish verifying your program, you can submit your source code by

   ```
   $ ~ee231002/bin/submit lab13 player4.c
   ```

   If you see a "submitted successfully" message, then you are done. In case you want to check which file and at what time you submitted your labs, you can type in the following command:

   ```
   $ ~ee231002/bin/subrec lab13
   ```

   It will show the submission records of lab13.
5. You should try to write the program as efficient as possible. The format of your program should be compact and easy to understand. These are part of the grading criteria.

```
/* blackjack player functions
 *    4 functions are needed
 *    bet_init: before cards are dealt, initial bet is made
 *    hit_or_stand: after cards are dealt, player needs to make a decision
 *                  whether to hit or stand
 *    shuttle: when the decks of cards are all used, dealer reshuffle
 *             the decks, and each player is notified
 *    deal_one_card: when each card is dealt by the dealer, each player is
 *                   notified what card is dealt
 *
 */

#include <stdio.h>
#include "card.h"
extern int step_display;
extern int Nhands;

char play4_name[7]="me";

// this function places a bet before cards are dealt
//   input: the amount of credit left for the user
//   output: how much the player want to bet for this round
//   algorithm: bet no more than 3
int play4_bet_init(const int credit)
{
    step_display=1;        // by turning off this, it speed up the game
    Nhands=100;            // number of hands to be played
    if (credit>=3) return 3;
    else return credit;
}

// this function decides if to hit for more cards or not
//   input: player's cards (and number of cards)
//          dealer's cards (and number of cards)
//            Note, at this time dealer has only one card
//   output: 1 to hit; 0 to stand, 2: double (receive only 1 card), 3: surrender
//   algorithm: hit if player's points is less than 17
int play4_hit_or_stand(const struct CARD mycard[], int Nmycard,
                       const struct CARD dealercard[], int Ndealercard)
{
    if (sum_cards(mycard,Nmycard)>=17) return 0;
    else return 1;
}
```

```
// this function is called when the cards are reshuffled
//   input: none
//   output: none
//   algorithm: do nothing
void play4_shuffle()
{
    return;
}

// this function is called when a card is dealt by the dealer
//   input: the card being dealt
//   output: none
//   algorithm: do nothing
void play4_deal_one_card(const struct CARD cd)
{
    return;
}
```