```
/* Michael E. Sparks, 11-14-16

   acey_ducey.c - Exercise to introduce basic elements of decision
                  making under uncertainty, capacity of data types,
                  etc. The rules used here correspond to those
                  given in David Ahl's "BASIC Computer Games" book,
                  available online at
                  http://www.vintage-basic.net/bcg/aceyducey.bas .
*/

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

/* Set aliases for agent classes and create
   an array of printable, descriptive strings. */
typedef enum ad_agent_ind {
  DUMB_AGENT, /* defaults to 0 */
  INTELLIGENT_AGENT
} AD_Agent_Ind;

/* Though the compiler doesn't need the size of the array
   to be specified, doing so can provide a bit of inexpensive,
   compile-time error checking. */
#define NUM_AGENTS 2
const char *AD_Agent_Names[(int)NUM_AGENTS] = {
  "Dumb",
  "Intelligent"
};

/* Defines variable type used to represent player's cash.
   There are definite limits to what computing machinery can
   represent, and unless we're monitoring the microprocessor's
   FLAGS register like a hawk, it's easy for overflow-related
   errors to squeak by unnoticed and/or unannounced.

   Hence, it's necessary but not sufficient that a program's logic
   be designed correctly (think "shape"): it also must provide reasonable
   assurance that it won't run afoul of what limits the host system can
   accommodate (think "size"). Ultimately, machine code gets the last
   laugh, which is why it's a very good idea to learn debugging down to
   the assembly level.

   For simplicity, I've manually adjusted the format specifiers
   in relevant printf statements (search for "%ld"). */
#define PURSE_T long

/* Initial amount players bring to the table */
#define STARTING_CASH (PURSE_T)100

/* Maximum number of dealer draws players can consider per game.
   Beware of setting this too high--if the agent goes on a serious
   winning streak, you can overflow the cash variable's type. */
#define MAX_HANDS 50

/* We ignore the concept of suit */
#define DRAW_CARD rand()%13+2
```

```
/* Convert numeric code to string representation. In C, it's generally
   preferable to use macros over functions for simple code units, as it
   helps avoid the performance hits associated with function calls. */
#define LOOKUP_CARD(S,C) \
{ \
  switch(C) { \
    case (11) : \
      sprintf(S,"Jack"); \
      break; \
    case (12) : \
      sprintf(S,"Queen"); \
      break; \
    case (13) : \
      sprintf(S,"King"); \
      break; \
    case (14) : \
      sprintf(S,"Ace"); \
      break; \
    default : \
      sprintf(S,"%d",C); \
  } \
}

/* This agent just randomly bids a cash amount between $0 and its total
   purse, regardless of spread on the cards dealt. Though the cardA and
   cardB parameters aren't used here, they are nonetheless part of the
   function interface to provide consistency with that of the intelligent
   agent function, shown infra.  This should become clearer when we prepare
   the array of function pointers for use in the player_moves function. */
PURSE_T dumb_agent(PURSE_T cash,int cardA,int cardB)
{
  return (PURSE_T)(rand()%(cash+1));
}

/* This agent's approach is to scale its existing purse by some
   weight connoting the likelihood that it will win the draw. */
PURSE_T intelligent_agent(PURSE_T cash,int cardA,int cardB)
{
  /* These weights are used by the intelligent agent only and
     are indexed based on the difference between the cards dealt,
     which must be .GE. 1. Subtract 1 from the difference in order
     to "dial up" the corresponding weight from this array.
     These were hand-picked, based on mere guesstimates/ intuition.
     That's not a great approach! A more principled way to learn
     the optimal weights would be to use something from the
     field of machine learning (e.g., a neural network), but
     here we just want to keep things simple. */
  const float weights[] = {
    0.000, /* A difference of 1 implies you can't win. Don't bid! */
    0.010,
    0.025,
    0.050, /* diff of 4 */
    0.100,
    0.150,
    0.300, /* diff of 7 */
    0.400,
    0.550,
    0.700, /* diff of 10 */
    0.900,
```

```
    0.950 /* max spread of 12: 2 & Ace */
  };

  /* The weighting scheme guarantees the agent's purse won't dip
     below $1, but it also guarantees it won't claw its way out of
     the $1 rut if it gets there. Thus, we take on a little risk. */
  return (cash == 1 && abs(cardA - cardB) > 8) ?
    1 : (PURSE_T)floor(cash * weights[abs(cardA - cardB) - 1]);
}

/* Develop an array of function pointers, pointing to agent-specific
   executable code, to be indexed using an AD_Agent_Ind-typed value. */
typedef PURSE_T (*ad_agent_fctT)(PURSE_T,int,int);
const ad_agent_fctT ad_agent_functors[(int)NUM_AGENTS] = {
  &dumb_agent,
  &intelligent_agent
};

/* function coordinates agents' handling of dealt hand */
PURSE_T player_moves(AD_Agent_Ind agent_class,int A,int B)
{
  static PURSE_T cash=STARTING_CASH; /* bidding cash */
  static unsigned int times_invoked=0; /* tracks hands dealt per game */
  PURSE_T bid, temp;

  bid=ad_agent_functors[agent_class](cash,A,B);
  printf("%s agent bid $%ld",AD_Agent_Names[agent_class],bid);

  if(bid) {
    int card=DRAW_CARD;
    char name[6];
    LOOKUP_CARD(name,card)

    printf(", drew %s ",name);

    if( (A < B && (card <= A || card >= B)) ||
        (A > B && (card <= B || card >= A)) ) {
      printf("(lose)");
      cash -= bid;
    }
    else {
      printf("(win)");
      cash += bid;
    }
  }
  printf("\nCash now $%ld\n",cash);

  /* There are two ways to signal the end of a game:
     1) you run out of cash .OR.
     2) you've played the max number of hands allowed.

     Note that the order of terms in the disjunction below does not matter:
     if cash .LE. 0 is true, then times_invoked will be reset in this branch
     and skipping the increment actually saves us an instruction or two.
     If false, then we're certain to increment and test this counter,
     as intended.

     Another note: if cash were to duck under 0, there would
     be a bug in our program. Our mathematical design excludes this
```

```
     possibility. Nonetheless, it's considered good form to program
     defensively and account for such a possibility.
     An invariant of this function is that it returns a value
     .GE. 0, and the calling code depends on this characteristic.
     During development/testing/debugging stages, it's a good idea
     to incorporate "sanity checks," which may or may not persist
     into the mature codebase after correctness is established. */
  if(cash <= 0 || ++times_invoked >= MAX_HANDS) {
    if(cash < 0) /* This should never occur. */
      fprintf(stderr,"Error: Negative ending cash!??\n");
    temp=cash < 0 ? 0 : cash;
    cash=STARTING_CASH;
    times_invoked=0;
    return(temp);
  }
  else
    return(cash);
}

/* Randomly select two cards of differing value.
   Yanking this code out improves readability of
   the acey_ducey function, nothing more. */
#define DEALER_DRAWS_TWO(FIRST,SECOND) \
{ \
  FIRST=DRAW_CARD; \
  LOOKUP_CARD(name,FIRST) \
  printf("\nCards: %s ",name); \
\
  do { \
    SECOND=DRAW_CARD; \
  } while(SECOND==FIRST); \
  LOOKUP_CARD(name,SECOND) \
  printf("%s\n",name); \
}

/* implements the main game control logic */
void acey_ducey(int curr_game,int max_games,AD_Agent_Ind agent_class)
{
  int cardA, cardB;
  char name[6]; /* space for card name + null terminator */

  PURSE_T cash_in_play, /* pegged to player's cash amount */
          max_cash_held=STARTING_CASH; /* denotes agent survivability */

  unsigned int hands_dealt=0; /* denotes agent fitness */

  if(curr_game > max_games) {
    printf("\n--The casino's closed for the %s agent!\n\n",
      AD_Agent_Names[agent_class]);
    return;
  }

  printf("\n--%s agent is now playing game number %d of %d ",
    AD_Agent_Names[agent_class],curr_game,max_games);
  printf("(Starting cash is $%ld)\n",STARTING_CASH);

  do {
    DEALER_DRAWS_TWO(cardA,cardB)
```

```
    if((cash_in_play=player_moves(agent_class,cardA,cardB)) > max_cash_held)
      max_cash_held=cash_in_play;

    /* Note that we would realize a nasty bug in the following
       conditional expression due to short circuiting had
       the terms of the conjunction been presented in the
       reverse order--to wit, if cash_in_play .EQ. 0 was true,
       then we'd fail to (pre-)increment the hands_dealt counter
       variable on that cycle. That would result in issuing an
       inaccurate report after exiting this looping construct.
       These kinds of bugs are pretty common, so always be careful
       when modifying state in the context of a conditional expression!
       Note that cash_in_play .GE. 0 is guaranteed by the player_moves
       function (recall that negative amounts evaluate to true). */
  } while(++hands_dealt < MAX_HANDS && cash_in_play);

  printf("\n--Hands dealt: %i\n",hands_dealt);
  printf("--Ending cash held: $%ld\n",cash_in_play);
  printf("--Max cash held: $%ld\n",max_cash_held);

  acey_ducey(curr_game+1,max_games,agent_class); /* call up next game */
}

/* driver application */
int main(int argc,char **argv)
{
  int num_games=1,
      agent_index=0;

  if(argc > 1)
    num_games=atoi(argv[1]); /* yes, this is potentially dangerous! */
  num_games=num_games > 0 ? num_games : 1;

  srand(time(NULL));
  while(agent_index < (int)NUM_AGENTS)
    acey_ducey(1,num_games,agent_index++);

  return(EXIT_SUCCESS);
}
```

```
#!/usr/bin/perl -w
use strict;

# Michael E. Sparks, 11-14-16

# Just some grungy Perl code to scrape together
# a tab-delimited results file

open(DE,"<Dumb_End.txt") or die "$!\n";
open(DM,"<Dumb_Max.txt") or die "$!\n";
open(DH,"<Dumb_Hands.txt") or die "$!\n";

open(IE,"<Intel_End.txt") or die "$!\n";
open(IM,"<Intel_Max.txt") or die "$!\n";
open(IH,"<Intel_Hands.txt") or die "$!\n";

print "EndCashDumb\tMaxCashDumb\tEnd2MaxDumb\tHandsDealtDumb\t";
print "EndCashIntel\tMaxCashIntel\tEnd2MaxIntel\tHandsDealtIntel\n";

my($de,$dh,$dm,$ie,$ih,$im);
while($de=<DE>) {
  chomp($de);
  $de=~/\$(\d+)$/;
  my $end=$1;
  print "$end\t";

  $dm=<DM>;
  chomp($dm);
  $dm=~/\$(\d+)$/;
  my $max=$1;
  printf("%d\t%.5f\t",$max,$end/$max);

  $dh=<DH>;
  chomp($dh);
  $dh=~/: (\d+)$/;
  print "$1\t";

  $ie=<IE>;
  chomp($ie);
  $ie=~/\$(\d+)$/;
  $end=$1;
  print "$end\t";

  $im=<IM>;
  chomp($im);
  $im=~/\$(\d+)$/;
  $max=$1;
  printf("%d\t%.5f\t",$max,$end/$max);

  $ih=<IH>;
  chomp($ih);
  $ih=~/: (\d+)$/;
  print "$1\n";
}

close DE;
close DM;
close DH;
close IE;
```

```
close IM;
close IH;

exit 0;
```