Screencasts I-II-III - Capstone Problem

1. Many of you have used a computer to play games. If a game is to hold the player's attention, it must not be too predictable. One of the ways that we can introduce a degree of unpredictability to games is to use a random number generator. C includes the following function in the library `stdlib.h`:

```
int rand( void );
/* returns a pseudo-random integer between 0 and RAND_MAX */
```

Note that the value of `RAND_MAX` varies from system to system. It is defined in the header file `stdlib.h`. We use the term "pseudo-random" because the function doesn't return a truly random integer – it uses an algorithm to generate numbers that *appear* to be randomly generated and uniformly distributed in the given range. So, if we knew the algorithm, we'd be able to predict the next number generated by the function.

a) What is the range of possible values that will be printed by the following code segment?

```
int nextVal = rand() % 10;
printf( "%d", nextVal );
```

b) With your answer to part a) in mind, write a function `rollDie` that simulates the roll of a regular 6-sided die. In other words, write a function that returns a random integer in the range 1 through 6, inclusive.

c) In the game of craps a user has a certain amount of money to bet. On each round the user places a bet that is at least the minimum bet established for the game and that's no more than the amount of money remaining in the user's wallet. We'll assume that the minimum bet is $5.

Having placed a bet, the user rolls a pair of dice to establish a point value. If the user rolls a 2, 3 or 12, they automatically lose their bet. If they roll a 7 or 11, they automatically win their bet. If they roll any other value, this value is called the *point* value. The user then continues to roll the dice until they roll a 7, in which case they lose, or they roll their point value, in which case they win.

After each round, the user is asked if they wish to play again. If, at the end of a round, the user has less than the minimum bet available in their wallet, the game ends automatically.

Here's a sample run:

```
How much money do you have to play with?
25

You have $25 in your wallet.
Place your bet (minimum $5): 5
You rolled a 8.
```

If first roll isn't a 2, 3, 7, 11 or 12, it establishes a point value.

```
Rolling for point: 8...
You rolled a 8.
You win :-)
```

If a point value is established, keep rolling until you roll the point value again (you win) or you roll a 7 (you lose).

```
Enter 1 to play again, 0 to quit:
1
```

```
You have $30 in your wallet.
Place your bet (minimum $5): 5
You rolled a 11.
You win :-)
Enter 1 to play again, 0 to quit:
1

You have $35 in your wallet.
Place your bet (minimum $5): 5
You rolled a 8.

Rolling for point: 8...
You rolled a 7.
You lose :-(
Enter 1 to play again, 0 to quit:
1

You have $30 in your wallet.
Place your bet (minimum $5): 35
You've bet more than you have in your wallet, try again.

You have $35 in your wallet.
Place your bet (minimum $5): 2
Your bet is below the minimum of $5, try again.

You have $35 in your wallet.
Place your bet (minimum $5): 25
You rolled a 9.

Rolling for point: 9...
You rolled a 2.
You rolled a 10.
You rolled a 4.
You rolled a 9.
You win :-)
Enter 1 to play again, 0 to quit:
1

You have $60 in your wallet.
Place your bet (minimum $5): 5
You rolled a 3.
You lose :-(
Enter 1 to play again, 0 to quit:
0

You have $55 left in your wallet. Goodbye!
Press any key to continue . . .
```

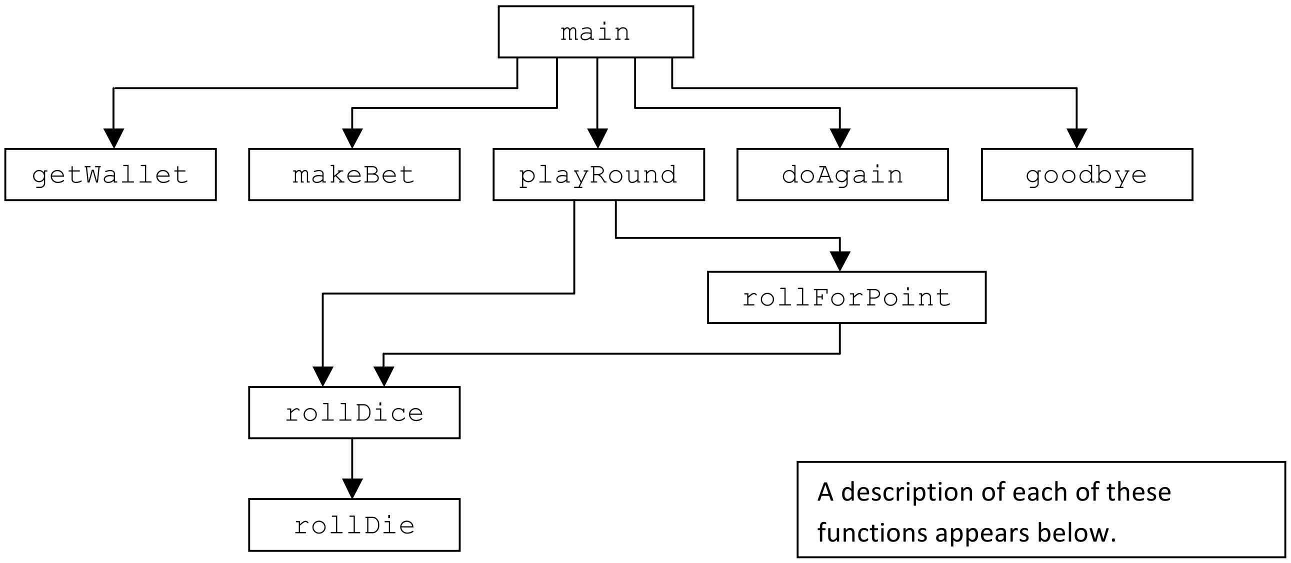
If the first roll is 7 or 11 you automatically win – no need for further rolls of dice.

Having established a point value, if you roll a 7 before rolling the point value again, you lose.

If a point value is established, keep rolling until you roll the point value again (you win) or you roll a 7 (you lose).

If the first roll is 2, 3 or 12 you automatically lose – no need for further rolls of the dice.

The code for such a game can quickly get out of hand unless we take a modular approach and break the problem down into smaller tasks, each of which can be coded with one or more functions. Here's a modular structure chart that represents one possible solution:



Function documentation:

```
/* Function: getWallet
 * Prompts the user for the amount of money they have to play with - assumed to be
 * measured in whole dollars.  Rejects values that are not at least MIN_BET in size and
 * prompts for another value until a value of size MIN_BET or greater is entered.
 * Returns: amount of money entered by user (a positive, integer dollar value)
 */

/* Function: makeBet
 * Prompts user to make a bet (minimum value: MIN_BET; maximum value: amount in wallet).
 * Keeps prompting user until a value of at least MIN_BET but no more than amount
 * in wallet is entered.
 * Parameter: wallet - the amount in the user's wallet
 * Returns: user's bet (minimum MIN_BET)
 */

/* Function: playRound
 * Plays a single round of craps with the user.
 * Returns: true if user won round, false otherwise.
 */

/* Function: doAgain
 * Ask user if they want to play again.
 * Returns: true if user wants to play again, false otherwise.
 */

/* Function: goodbye
 * Prints goodbye message to user based on whether or not they
 * went broke while playing the game.  Tells the user they're broke
 * if they have less than MIN_BET in their wallet, otherwise tells
 * them how much they have in their wallet.
 * Parameter: wallet - amount of money in wallet
 */

/* Function: rollForPoint
 * Repeatedly rolls dice until either the point value or the value 7 is rolled.
 * Parameter: point - the current point value
 * Returns: true if user rolled point value before rolling a 7 (user won round),
 * false otherwise (user lost round)
 */

/* Function: rollDice
 * Rolls a pair of dice.
 * Returns: sum of face values rolled
 */

/* Function: rollDie
 * Rolls a single die.
 * Returns: face value rolled
 */
```

Develop the game by coding and testing each function, one by one.

Arrays

Screencast I

1. What is the purpose of the following program? Another way of stating this question is: "What statement would you provide in the opening documentation that describes what this program will do when it runs?"

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

#define SIZE 4

int main( void ) {
   double data[] = { 5.1, 23.7, 2.0, -4.3 };
   int    index = 0;
   double sum = 0.0;

   while( index < SIZE ) {
       sum += data[ index ];
       index++;
   }

   printf( "%f\n", sum / SIZE );

   system( "PAUSE" );
   return 0;
}
```

2. How would you complete the sentence printed by the `printf` statement at the end of the following program to accurately describe how the array has been transformed? Note: you may wish to start by drawing a trace table and trace the value of key variables and expressions for the first few passes through the loop until you can figure out what the code does in general.

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

#define SIZE 6

int main( void ) {
   int data[] = { 5, 23, 2, -4, 7, 12 };
   int indexL = 0;
   int indexR = SIZE - 1;
   int temp;

   while( indexL < indexR ) {
       temp = data[ indexL ];
       data[ indexL ] = data[ indexR ];
       data[ indexR ] = temp;
       indexL++;
       indexR--;
   }

   printf( "Array has been …\n" );

   system( "PAUSE" );
   return 0;
}
```