**CS161 - Programming Assignment - Comprehensive (Twenty-One)**

The purpose of this assignment is to use all you've learned this term to develop an application of some substantial size. It also give you more experience with arrays.

This program builds upon the previous assignment, 7, CardSuperFun, and will implement the game of 21 (i.e. blackjack). In assignment 7, we wrote (and tested) some routines that enable us to use an array of integers as a deck of cards. Now we'll add a few additional functions and implement the game logic.

---

**Part 1 - Additional card/deck manipulation routines**

Details

To recap, in the last assignment, we wrote and tested the following functions:

```
void InitializeDeck( int deck[]);
void DumpDeck(const int deck[]);
void Shuffle(int deck[]);
void ShowCard(int card);
void ShowCards(const int cards[], int numCards,  bool hideFirstCard);
```

We'll now write the routines described below. After writing a routine, test it thoroughly to verify it works properly before moving on.

```
int CardValue(int card);
```

returns the cards value. Ace's are 11, face cards are 10, and all other cards are their rank (the number shown on them). (note: in actual 21, an ace can be one or eleven, but for this function we'll just make them 11 to simplify things).

```
int getTopCard(int deck[]);
```

returns the "top" card off the deck. To implement this you'll need some way of knowing what the "top" card is. Here are two possibilities:

**Method 1**: The top card is the first card in the given array that has a value greater than zero. After pulling this value off, replace it with zero (as the card's no longer in the deck).

**Method 2:** Define a static int that contains the index of the top card. Increment this after "pulling off" the the top card. This is easier but less flexible, as we can't ever restart the deck by recreating it.

If the deck is empty, return a 0 (no card).

---

**Part 2 - Game Logic Implementation**

Finish implementing the 21 game by first implementing the functions described below, and then writing the main game logic as per the pseudocode we generated in class.

These two functions deal with a players hand. A "hand" will be an array of 10 integers, with each integer representing a card (or a possible card). For an empty hand, they'll all be zero, but as cards get added, the zeros will be replaced with actual card values. Since the array has 10 integers, a hand could consist of a maximum of 10 cards, which is more than enough for 21. It's unlikely we'd ever exceed 5 or 6.

Additional needed functions:

```
void addToHand(int hand[], int cardToAdd);
```

Adds the given card to an array representing a players hand. (locate the first element that is 0 and replace it with the cardToAdd.

```
int getHandValue(const int hand[]);
```

computes and returns the value of the given hand. This can easily be done by adding up the value of each card (call cardValue for each non-zero element of the array)

Possibly others....

After you have completed writing and testing each function, you are then ready to implement the overall game logic. For the purpose of the assignment, you only need to have it play one hand, but after doing that, it's quite easy to have it play hands until either the user chooses to quit or their aren't enough cards left in the deck for another round (15 or so).

To do so, you'd need to move the game logic into a function named playOneHnad

```
void playOneHand(int deck[]);
```

and call that from main as per the pseudocode generated in class.

---