ITP 165: Introduction to C++ Programming

Homework 6: Chuck-a-luck — Due: 11:59pm, October 16th, 2015

Goal

Chuck-a-luck is a gambling dice game found in some casinos. We'll simulate the game (though not all of the betting combinations). For more information on chuck-a-luck, see the Wikipedia article.

Setup

- Create a new project in Visual Studio or Xcode called **homework06** and a new C++ file named **chuck.cpp**.
- Your **chuck.cpp** file must begin with the in the following (replace the name and email with your actual information):

```
// Name
// ITP 165, Fall 2015
// Homework 06
// USC email
```

Part 1: Functions setup

- We will use several functions to play this game. For now, create the following 5 functions:
  - "`roll`"
    - Accepts no input
    - Returns a single `int` representing the number rolled
      - For now, in its body, have it return 1
  - "`roll`" (yes, a 2nd – *overloaded* – version of the function)
    - Accepts 2 pieces of input (you may name the input anything you'd like):
      - An `int` array (eventually containing the dice rolls)
      - A single `int` (containing the size of the array)
    - Returns nothing
  - "`displayRoll`"
    - Accepts 2 pieces of input (you may name the input anything you'd like):
      - An `int` array (eventually containing the dice rolls)
      - A single `int` (containing the size of the array)
    - Returns nothing
  - "`computeBetResult`"
    - Accepts 4 pieces of input (you may name the input anything you'd like):
      - An `int` array (containing the dice rolls)
      - A single `int` (containing the size of the array)
      - A single `int` (containing the amount the user bet)
      - A single `int` (containing the number the user bet on)
    - Returns a single `int` representing the amount the user won
      - For now, in its body, have it return 0
  - "`main`"
    - This is our regular `main`. It should take no inputs and return 0.

- Make sure your code runs before moving on to the next step.

Part 2: User interface

- Setup an array of `int`s, but don't use an integer literal like the number "12". Instead make a `const int` to hold the size of the array. This will hold the number of dice used in the game. For now we'll play with 3 dice.
- Create an `int` to hold the user's "money". Initialize it to 100.
- Introduce the user to your game, announce how much money they have, and ask for an amount to bet (which you should store in another `int` variable).
    - Make sure this bet is valid. A valid bet is greater than 0 and less than or equal to the total amount of money the user has. If the user enters an invalid bet, tell the user what the valid betting range is and prompt them again.
- Ask the user for a number to bet on.
    - Make sure the number is valid. Valid numbers are 1, 2, 3, 4, 5, and 6. If the user enters an invalid number, tell the user the proper range and ask for another number.
- Summarize and repeat the user's decisions.
- Deduct the bet from the current user's money – after all, this is a betting game!
- Leave some space in main after this point, we'll return here in a bit.
- We'll now call our empty (or "stubbed") functions. First we'll "roll" the dice. The purpose of the 2nd `roll` function (the one that takes input) is to fill the inputted array with random numbers from 1 to 6. Call that function (not the `roll` that accepts no input). For now it will do nothing, but we'll fill in functionality soon.
- The purpose of `displayRoll` is to use `std::cout` to display the results of the dice thrown. While it does nothing for now, we'll call the function now, knowing it will eventually tell the user the results of their roll.
- The function `computeBetResults` will determine the user's winnings and return the money won or lost. For now, call that function giving it:
    - The dice array
    - The size of the dice array
    - The amount of money the user bet
    - The number the user bet on

  Store the results of the function call in an `int`.
- For now the results of the `computeBetResults` will always be 0, but eventually it will be a positive number or 0. If it's positive, then the user won the wager. Congratulate the user and add the winnings to the user's total. If the number is 0, the user lost the bet, encourage the user to try again.
- Tell the user their new money total.
- Ask the user if they'd like to play again. If the user enters N or n, stop the game. The game should also stop if the user runs out of money.
- Output for Part 2 should look something like this (user input in **red**):

```
STEP RIGHT UP AND PLAY SOME CHUCK-A-LUCK!
You have $100
Enter a bet amount: $20
What number do you want to bet on (1-6)? 3
```

```
You bet $20 on 3
You rolled:
You lost your bet :(
You now have $80
Would you like to play again (y/n)? y
Enter a bet amount: $100
Invalid bet. Please enter an amount from $1 to $80
Enter a bet amount: $7
What number do you want to bet on (1-6)? 7
Invalid number. Must be between 1 and 6.
What number do you want to bet on (1-6)? 1
You bet $7 on 1
You rolled:
You lost your bet :(
You now have $73
Would you like to play again (y/n)? n
You ended the game with $73
```

Part 3: Die rolling

- The function `roll` (with no input) generates a random number between 1 and 6. To do that we'll use a new set of `std` functions.
    - Random number generation is accessible through the `cstdlib` library, so add an line at the beginning of your program to include that library
    - The first function is `std::rand`. It accepts no input and returns a random `int` as output. The random number will be somewhere between 0 and (about) 32000.
    - The random number generation system must be primed before you can call `std::rand`. The function that primes the random number generator is `std::srand`. It returns no output, but receives an `int` as input – this is sometimes called the "seed". However, we need to choose the seed carefully. Because it initializes the random number generator, it shouldn't be something that the user can figure out…
    - A number that is always changing is time. In fact, the function `std::time` can return the ever-changing number of seconds since 1/1/1970. It just needs the input of zero.
- Prime the random number generation system in main. Make it the first line of code in main. To prime the system, call `std::srand`. It's input should be a call to `std::time` with 0 as it's input.
- Change `roll`'s return call now. It's supposed to generate a random number between 1 and 6. We can call `std::rand` and generate a random number beyond of this range. Somehow you've got to reduce that random number's range to something between 1 and 6.
    - HINT: It involves modulus (%).
- When you get roll working, test it for a bit. Test it by calling `std::cout` on the values it generates. If you've got it working right, you'll get a different set of values each time.

Part 4: Rolling sets of dice and displaying them

- The function `roll` (with 2 inputs) should fill the inputted array with random numbers. The size of that array is the 2nd argument to the function. Fill the array with random numbers 1 through 6.
    - HINT: it involves looping over the size of the array and calling `roll` (the version with no inputs)
- Fill in the function `displayRoll` with similar functionality. Its job is to display the contents of the inputted array.
    - HINT: it involves looping over the size of the array and using `std::cout`.
- Once you've completed Parts 3 and 4, your program output should resemble the following (user input in **red**):

```
STEP RIGHT UP AND PLAY SOME CHUCK-A-LUCK!
You have $100
Enter a bet amount: $21
What number do you want to bet on (1-6)? 4
You bet $21 on 4
You rolled:
        5
        2
        3
You lost your bet :(
You now have $79
Would you like to play again (y/n)? y
Enter a bet amount: $50
What number do you want to bet on (1-6)? 6
You bet $50 on 6
You rolled:
        2
        2
        2
You lost your bet :(
You now have $29
Would you like to play again (y/n)? n
You ended the game with $29
```

Part 5: Computing bet results

- Just to review, the function `computeBetResult` accepts the following input:
    - An `int` array (containing the dice rolls)
    - A single `int` (containing the size of the array)
    - A single `int` (containing the amount the user bet)
    - A single `int` (containing the number the user bet on)
- Go through the inputted dice rolls and determine how many times the bet upon number occurs. It should be a number between 0 and 3.
- Announce to the user how many dice they've matched.

- The function should return the payout based on how many times the user's number showed up on the dice.
    - If the number occurred 0 times, the payout is 0.
    - If the number occurred 1 time, payout is 1 times the amount of money the user bet (plus their original bet amount).
    - If the number occurred 2 times, payout is 3 times the amount of money the user bet (plus their original bet amount).
    - If the number occurred 3 times, payout is 10 times the amount of money the user bet (plus their original bet amount).
- Calculate the proper payout and return that value.
- After completing part 5, your program output should resemble the following (user input in **red**):

```
STEP RIGHT UP AND PLAY SOME CHUCK-A-LUCK!
You have $100
Enter a bet amount: $20
What number do you want to bet on (1-6)? 1
You bet $20 on 1
You rolled:
        6
        4
        5
You matched 0 dice!
You lost your bet :(
You now have $80
Would you like to play again (y/n)? y
Enter a bet amount: $20
What number do you want to bet on (1-6)? 5
You bet $20 on 5
You rolled:
        2
        1
        1
You matched 0 dice!
You lost your bet :(
You now have $60
Would you like to play again (y/n)? y
Enter a bet amount: $20
What number do you want to bet on (1-6)? 5
You bet $20 on 5
You rolled:
        2
        2
        4
You matched 0 dice!
You lost your bet :(
You now have $40
Would you like to play again (y/n)? y
```

```
Enter a bet amount: $20
What number do you want to bet on (1-6)? 5
You bet $20 on 5
You rolled:
        1
        5
        5
You matched 2 dice!
YOU WIN $80
You now have $100
Would you like to play again (y/n)? n
You ended the game with $100
```

A Note on Style

Be sure to comment your code.

As we discussed in lecture, it is extremely important that your code is properly indented as it greatly adds to readability. Because of this, if you submit a code file that is not reasonably indented, you will have points deducted.

Likewise, you will lose points if your variable names are not meaningful. Make sure you use variable names that correspond to what you are actually storing in the variables.

Full Sample Output

Below is sample output for a full run-through of the program. User input is in **red**.

```
STEP RIGHT UP AND PLAY SOME CHUCK-A-LUCK!
You have $100
Enter a bet amount: $20
What number do you want to bet on (1-6)? 2
You bet $20 on 2
You rolled:
        4
        3
        1
You matched 0 dice!
You lost your bet :(
You now have $80
Would you like to play again (y/n)? y
Enter a bet amount: $20
What number do you want to bet on (1-6)? 6
You bet $20 on 6
You rolled:
        1
        5
        2
You matched 0 dice!
You lost your bet :(
```

```
You now have $60
Would you like to play again (y/n)? y
Enter a bet amount: $20
What number do you want to bet on (1-6)? 4
You bet $20 on 4
You rolled:
        4
        4
        5
You matched 2 dice!
YOU WIN $80
You now have $120
Would you like to play again (y/n)? n
You ended the game with $120
```

Deliverables

1. A compressed **Hw06** folder containing ***only*** the file named **chuck.cpp**. It must be submitted through Blackboard.

Grading

Item	Points
Part 1: Functions setup	5
Part 2: User interface	5
Part 3: Die rolling	5
Part 4: Rolling sets of dice and displaying them	10
Part 5: Computing bet results	5
Total*	**30**

* Points will be deducted for poor code style, or improper submission.