1.

One obstacle I overcame was to understand the functions we suppose to write on the PowerballLottery.cpp. For example, PowerballLottery::WinningPossibility PowerballLottery::checkTicket(PowerballTicket ticket), it was quit difficult to understand the syntax, because it contains two type member functions in PowerballLottery. It took me a while to understand what I suppose to write on this function. So basically, I supposed to write a function that would check the powerball Ticket that is given with powerball lottery ticket. In this case, there are also several conditions that I had to check, which I never thought about it. The first condition I need to check is the first five balls, and if they match with the lottery ticket balls then it supposed to implement by 1. Then I also needed to write the switch statement accounts for the fact if the powerball in lottery is matched with the ticket. At this step, it took me a while to think that what is the best way to account for conditions like if there is one ball that match with the ticket or there is none of them. Or there is one with powerball that matched with the ticket. With that in mind, I would have to write if else statement to account two type of condition, one is if ticket powerball is matched with the powerball in lottery, otherwise goes to another braces. And I would also have to write switch statement inside the function to account every possible situation. At the end, I need to return that result.

// test code

PowerballTicket ticket( 3, 4, 6, 8, 5, 10 );

assert( ticket.getBall1() == 1);

assert( ticket.getBall2() == 2);

assert( ticket.getBall3() == 3);

assert( ticket.getBall4() == 4);

assert( ticket.getBall5() == 5);

assert( ticket.getPowerball() == 6);// TEST TO SEE IF THE POWERBALLTICKET WORKS PROPERLY

PowerballLottery lottery( 10, 2, 33, 55, 12, 14 );

assert( lottery.getBall1() == 1);

assert( lottery.getBall2() == 2);

assert( lottery.getBall3() == 3);

assert( lottery.getBall4() == 4);

assert( lottery.getBall5() == 5);

assert( lottery.getPowerball() == 6); // TEST TO SEE IF THE POWERBALLLOTTERY FUNCTIONS PROPERLY

assert( lottery.checkTicket(ticket) == PowerballLottery::WinningPossibility::FIVEPLUSPOWERBALL);

ticket = PowerballTicket( 1, 2, 3, 4, 5, 12 );

assert( lottery.checkTicket(ticket) == PowerballLottery::WinningPossibility::FIVE);

ticket = PowerballTicket( 1, 2, 3, 4, 15, 12)

PowerballLottery::WinningPossibility::FOURPLUSPOWERBALL);

ticket = PowerballTicket( 1, 2, 3, 4, 5, 12 );

assert( lottery.checkTicket(ticket) == PowerballLottery::WinningPossibility::FOUR);

ticket = PowerballTicket( 1, 2, 3, 4, 15, 12)

THIS CODE WILL TEST THE WINNINGPOSSIBILITY AND DIFFERENT SCENARIOS.

PowerballTicket quickPickTicket( 1, 2, 3, 4, 5, 6);

for (int i = 0; i < 20; i++)

{

quickPickTicket = lottery.quickPick();

// all the ball numbers need to be different...

assert( quickPickTicket.getBall1() != quickPickTicket.getBall2() &&

quickPickTicket.getBall1() != quickPickTicket.getBall3() &&

quickPickTicket.getBall1() != quickPickTicket.getBall4() &&

quickPickTicket.getBall1() != quickPickTicket.getBall5() &&

quickPickTicket.getBall2() != quickPickTicket.getBall3() &&

quickPickTicket.getBall2() != quickPickTicket.getBall4() &&

quickPickTicket.getBall2() != quickPickTicket.getBall5() &&

quickPickTicket.getBall3() != quickPickTicket.getBall4() &&

quickPickTicket.getBall3() != quickPickTicket.getBall5() &&

quickPickTicket.getBall4() != quickPickTicket.getBall5());

}

// CHECK TO SEE IF THE QUICKPICK HAS REPEATED VALUES IN THE TICKET.