Shaan Mathur

Stahl

CS31

February 24, 2016

Project 5 Report

This project was fairly straightforward. All that was asked of us was to implement two simple classes that the professor designed (even game.cpp had nothing special to it). So in terms of obstacles, there were not many. I come from a Java background from AP Computer Science, so I am used to writing every .java file as a class, since it’s a class-based and fully OOP language. I have never had to make separate files with function prototypes and class declaration in one and implementation in another. Java always had direct implementation in the class (what would be deemed inline in C++). So perhaps the most difficult part of this project for myself personally is getting used to this idea of having separate files for my classes, .h for declaration and .cpp for implementation. Other than that, the task that was asked of us was fairly straightforward.

Test Data:

//Makes sure that random lottery tickets always have random, //different values for balls 1-6

/\*Functions tested: PowerballLottery()

quickPick()

getBall1()

getBall2()

getBall3()

getBall4()

getBall5()

\*/

PowerballLottery lottery;

for(int i = 0; i < 1000; i++){

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());

lottery.quickPick();

}

//Makes sure accessor functions return properly

/\*Functions tested: PowerballTicket(int,int,int,int,int,int)

getBall1()

getBall2()

getBall3()

getBall4()

getBall5()

getPowerball()

\*/

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

Assert(t.getBall1() == 1);

Assert(t.getBall2() == 2);

Assert(t.getBall3() == 3);

Assert(t.getBall4() == 4);

Assert(t.getBall5() == 5);

Assert(t.getPowerball() == 6);

PowerballTicket t2(1, 5, 4, 3, 2, 6);

Assert(t2.getBall1() == 1);

Assert(t2.getBall2() == 5);

Assert(t2.getBall3() == 4);

Assert(t2.getBall4() == 3);

Assert(t2.getBall5() == 2);

Assert(t2.getPowerball() == 6);

//Assures that order does not matter in evaluating the tickets

/\*Functions tested: PowerballLottery(int,int,int,int,int,int)

checkTicket(PowerballTicket);

enum Tested: WinningPossibility

\*/

PowerballLottery l(1,2,3,4,5,6);

Assert(l.checkTicket(t) == PowerballLottery::FIVEPLUSPOWERBALL);

Assert(l.checkTicket(t2) == PowerballLottery::FIVEPLUSPOWERBALL);

//The fact that this initializes indicates a flaw, showing that

//the Constructor for PowerballLottery and PowerballTicket

//assumes safe parameters

/\*Functions tested: PowerbalLottery(int,int,int,int,int,int)

PowerballTicket(int,int,int,int,int,int)

checkTicket(PowerballTicket)

enum tested: WinningPossibility

\*/

PowerballLottery l2(-1,-2,-3,70,94,27);

PowerballTicket tick(-2,-1,-3,94,70,28);

Assert(l2.checkTicket(tick) == PowerballLottery::FIVE);

//Makes sure appropriate WinningPossibility is returned for every

//Possible case

/\*Functions tested: PowerballLottery(int,int,int,int,int,int)

PowerballTicket(int,int,int,int,int,int)

checkTicket(PowerballTicket);

enum tested: WinningPossibility

\*/

PowerballLottery lottery(1,2,3,4,5,6);

PowerballTicket t1(5,4,3,2,1,6);

PowerballTicket t2(5,4,3,2,1,7);

PowerballTicket t3(15,4,3,2,1,6);

PowerballTicket t4(15,4,3,2,1,7);

PowerballTicket t5(15,14,3,2,1,6);

PowerballTicket t6(15,14,3,2,1,16);

PowerballTicket t7(15,14,13,2,1,6);

PowerballTicket t8(15,14,13,2,1,16);

PowerballTicket t9(15,14,13,12,1,6);

PowerballTicket t10(15,14,13,12,1,16);

PowerballTicket t11(15,14,13,12,11,6);

PowerballTicket t12(15,14,13,12,11,16);

Assert(lottery.checkTicket(t1)==PowerballLottery::FIVEPLUSPOWERBALL);

Assert(lottery.checkTicket(t2)==PowerballLottery::FIVE);

Assert(lottery.checkTicket(t3)==PowerballLottery::FOURPLUSPOWERBALL);

Assert(lottery.checkTicket(t4)==PowerballLottery::FOUR);

Assert(lottery.checkTicket(t5)==PowerballLottery::THREEPLUSPOWERBALL);

Assert(lottery.checkTicket(t6)==PowerballLottery::THREE);

Assert(lottery.checkTicket(t7)==PowerballLottery::TWOPLUSPOWERBALL);

Assert(lottery.checkTicket(t8)==PowerballLottery::NOTWINNING);

Assert(lottery.checkTicket(t9)==PowerballLottery::ONEPLUSPOWERBALL);

Assert(lottery.checkTicket(t10)==PowerballLottery::NOTWINNING);

Assert(lottery.checkTicket(t11)==PowerballLottery::POWERBALL);

Assert(lottery.checkTicket(t12)==PowerballLottery::NOTWINNING);

//These should print progressively better statements, going from //smallest prize to largest

/\*Functions tested: printWhatHappened(PowerballTicket)

Enum tested(in function): PowerballLottery::WinningPossibility

\*/

lottery.printWhatHappened(t12);

lottery.printWhatHappened(t11);

lottery.printWhatHappened(t9);

lottery.printWhatHappened(t7);

lottery.printWhatHappened(t6);

lottery.printWhatHappened(t5);

lottery.printWhatHappened(t4);

lottery.printWhatHappened(t3);

lottery.printWhatHappened(t2);

lottery.printWhatHappened(t1);

//All assertions shown can alternatively use //lottery.printWhatHappened() to show to console that everything worked

cout << “All tests passed!” << endl;