Tori Wang

605561243

Notable obstacles I overcame included understanding the assignment in the beginning (I was confused as to if we had to implement the entire blackjack game or just the first strategy), as well as determining the most efficient way to go through the options depending on the cards. I decided on using Switch statements that had (if) and (else if) statements in order to go through all possible hands.

List of test data (reason for test in comments):

| Card c1;  Card c2;  Hand h( c1, c2);  Card ace( Suit::CLUBS, Face::ACE );  Card two( Suit::CLUBS, Face::DEUCE );  Card three( Suit::CLUBS, Face::THREE );  Card four( Suit::CLUBS, Face::FOUR );  Card five( Suit::CLUBS, Face::FIVE );  Card six( Suit::CLUBS, Face::SIX );  Card seven( Suit::CLUBS, Face::SEVEN );  Card eight( Suit::CLUBS, Face::EIGHT );  Card nine( Suit::CLUBS, Face::NINE );  Card ten( Suit::CLUBS, Face::TEN );  Card jack( Suit::CLUBS, Face::JACK );  Card queen( Suit::CLUBS, Face::QUEEN );  Card king( Suit::CLUBS, Face::KING );      Hand h9(nine, nine);  Hand h10(ten, ten);      //Checking non ace non pair hands:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*       Hand n8(three, five);  assert(n8.evaluateHand(four) == Choice::HIT );  assert(n8.evaluateHand(ace) == Choice::HIT );   Hand n9(four, five);  assert(n9.evaluateHand(ace) == Choice::HIT );  assert(n9.evaluateHand(two) == Choice::HIT );  assert(n9.evaluateHand(three) == Choice::DOUBLEHIT );  assert(n9.evaluateHand(six) == Choice::DOUBLEHIT );  assert(n9.evaluateHand(seven) == Choice::HIT );   Hand n10(four, six);  assert(n10.evaluateHand(two) == Choice::DOUBLEHIT );  assert(n10.evaluateHand(nine) == Choice::DOUBLEHIT );  assert(n10.evaluateHand(ten) == Choice::HIT );  assert(n10.evaluateHand(ace) == Choice::HIT );     Hand n11(five, six);  assert(n11.evaluateHand(two) == Choice::DOUBLEHIT );  assert(n11.evaluateHand(two) == Choice::DOUBLEHIT );  assert(n11.evaluateHand(two) == Choice::DOUBLEHIT );  assert(n11.evaluateHand(two) == Choice::DOUBLEHIT );     Hand n12(five, seven);  assert(n12.evaluateHand(two) == Choice::HIT );  assert(n12.evaluateHand(three) == Choice::HIT );  assert(n12.evaluateHand(four) == Choice::STAND );  assert(n12.evaluateHand(six) == Choice::STAND );  assert(n12.evaluateHand(seven) == Choice::HIT );  assert(n12.evaluateHand(ace) == Choice::HIT );     Hand n13(six, seven);  assert(n13.evaluateHand(two) == Choice::STAND );  assert(n13.evaluateHand(six) == Choice::STAND );  assert(n13.evaluateHand(seven) == Choice::HIT );  assert(n13.evaluateHand(ace) == Choice::HIT );    Hand n14(six, eight);  assert(n14.evaluateHand(two) == Choice::STAND );  assert(n14.evaluateHand(six) == Choice::STAND );  assert(n14.evaluateHand(seven) == Choice::HIT );  assert(n14.evaluateHand(ace) == Choice::HIT );    Hand n15(seven, eight);  assert(n15.evaluateHand(two) == Choice::STAND );  assert(n15.evaluateHand(six) == Choice::STAND );  assert(n15.evaluateHand(seven) == Choice::HIT );  assert(n15.evaluateHand(nine) == Choice::HIT );  assert(n15.evaluateHand(ten) == Choice::SURRENDERHIT );  assert(n15.evaluateHand(ace) == Choice::HIT );    Hand n16(seven, nine);  assert(n16.evaluateHand(two) == Choice::STAND );  assert(n16.evaluateHand(six) == Choice::STAND );  assert(n16.evaluateHand(seven) == Choice::HIT );  assert(n16.evaluateHand(eight) == Choice::HIT );  assert(n16.evaluateHand(nine) == Choice::SURRENDERHIT );  assert(n16.evaluateHand(ace) == Choice::SURRENDERHIT );      Hand n17(eight, nine);  assert(n17.evaluateHand(ace) == Choice::STAND );         //Checking one ace hands:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      Hand s2(ace, two);  assert(s2.evaluateHand(four) == Choice::HIT );  assert(s2.evaluateHand(six) == Choice::DOUBLEHIT );  assert(s2.evaluateHand(seven) == Choice::HIT );  assert(s2.evaluateHand(ace) == Choice::HIT );     Hand s3(ace, three);  assert(s3.evaluateHand(four) == Choice::HIT );  assert(s3.evaluateHand(six) == Choice::DOUBLEHIT );  assert(s3.evaluateHand(seven) == Choice::HIT );  assert(s3.evaluateHand(ace) == Choice::HIT );    Hand s4(ace, four);  assert(s4.evaluateHand(three) == Choice::HIT );  assert(s4.evaluateHand(four) == Choice::DOUBLEHIT );  assert(s4.evaluateHand(seven) == Choice::HIT );  assert(s4.evaluateHand(ace) == Choice::HIT );    Hand s5(ace, five);  assert(s5.evaluateHand(three) == Choice::HIT );  assert(s5.evaluateHand(four) == Choice::DOUBLEHIT );  assert(s5.evaluateHand(seven) == Choice::HIT );  assert(s5.evaluateHand(ace) == Choice::HIT );    Hand s6(ace, six);  assert(s6.evaluateHand(two) == Choice::HIT );  assert(s6.evaluateHand(three) == Choice::DOUBLEHIT );  assert(s6.evaluateHand(six) == Choice::DOUBLEHIT );  assert(s6.evaluateHand(ace) == Choice::HIT );    Hand s7(ace, seven);  assert(s7.evaluateHand(two) == Choice::STAND );  assert(s7.evaluateHand(three) == Choice::DOUBLESTAND );  assert(s7.evaluateHand(six) == Choice::DOUBLESTAND );  assert(s7.evaluateHand(seven) == Choice::STAND);  assert(s7.evaluateHand(ace) == Choice::HIT);     Hand s8(ace, eight);  assert(s8.evaluateHand(ace) == Choice::STAND);  assert(s8.evaluateHand(seven) == Choice::STAND);        //Checking all pairs:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    // two aces...  assert( h.evaluateHand(six) == Choice::SPLIT );  //two tens  assert( h10.evaluateHand(six) == Choice::STAND );   //two nines  assert(h9.evaluateHand(six) == Choice::SPLIT );  assert(h9.evaluateHand(seven) == Choice::STAND );  assert(h9.evaluateHand(ten) == Choice::STAND );  assert(h9.evaluateHand(ace) == Choice::STAND );     //two eigths  Hand h8(eight, eight);  assert(h8.evaluateHand(six) == Choice::SPLIT );    //two sevens    Hand h7(seven, seven);  assert(h7.evaluateHand(seven) == Choice::SPLIT );  assert(h7.evaluateHand(eight) == Choice::HIT );   Hand h6(six, six);  assert(h6.evaluateHand(two) == Choice::SPLITHIT );  assert(h6.evaluateHand(six) == Choice::SPLIT );  assert(h6.evaluateHand(ace) == Choice::HIT );   Hand h5(five, five);  assert(h5.evaluateHand(two) == Choice::DOUBLEHIT );  assert(h5.evaluateHand(nine) == Choice::DOUBLEHIT );  assert(h5.evaluateHand(ten) == Choice::HIT );  assert(h5.evaluateHand(ace) == Choice::HIT );    Hand h4(four, four);  assert(h4.evaluateHand(four) == Choice::HIT );  assert(h4.evaluateHand(five) == Choice::SPLITHIT );  assert(h4.evaluateHand(six) == Choice::SPLITHIT );  assert(h4.evaluateHand(ace) == Choice::HIT );    Hand h3(three, three);  assert(h3.evaluateHand(three) == Choice::SPLITHIT );  assert(h3.evaluateHand(four) == Choice::SPLIT );  assert(h3.evaluateHand(seven) == Choice::SPLIT );  assert(h3.evaluateHand(ace) == Choice::HIT );     Hand h2(two, two);  assert(h2.evaluateHand(three) == Choice::SPLITHIT );  assert(h2.evaluateHand(four) == Choice::SPLIT );  assert(h2.evaluateHand(seven) == Choice::SPLIT );  assert(h2.evaluateHand(ace) == Choice::HIT );          Hand h1( two, two );  assert( h1.evaluateHand(two) == Choice::SPLITHIT );  assert( h1.evaluateHand(three) == Choice::SPLITHIT );  assert( h1.evaluateHand(four) == Choice::SPLIT );  assert( h1.evaluateHand(five) == Choice::SPLIT );  assert( h1.evaluateHand(six) == Choice::SPLIT );  assert( h1.evaluateHand(seven) == Choice::SPLIT );  assert( h1.evaluateHand(eight) == Choice::HIT );  assert( h1.evaluateHand(nine) == Choice::HIT );  assert( h1.evaluateHand(ten) == Choice::HIT );  assert( h1.evaluateHand(jack) == Choice::HIT );  assert( h1.evaluateHand(queen) == Choice::HIT );  assert( h1.evaluateHand(king) == Choice::HIT ); |
| --- |
| //Check ace pair  assert( h.evaluateHand(king) == Choice::SPLIT ); |
|  |