```
//  Project 3:  House Party

#include <string>
#include <cctype>
using namespace std;

bool isValidUppercaseStateCode(string stateCode);

//************************************
//  hasProperSyntax
//************************************

bool hasProperSyntax(string pollData)
{
      // An empty poll data string is well-formed

    if (pollData.size() == 0)
        return true;

      // Each iteration of the loop recognizes one state forecast

    size_t k = 0;
    while (k != pollData.size())
    {
          // The state forecast must start with two letters

        if (! isalpha(pollData[k]))
            return false;
        k++;
        if (k == pollData.size()  ||  ! isalpha(pollData[k]))
            return false;
        k++;

          // Those letters must be the code for a state

        string state = pollData.substr(k-2, 2);
        state[0] = toupper(state[0]);
        state[1] = toupper(state[1]);
        if (!isValidUppercaseStateCode(state))
            return false;

          // The state code must be a followed by zero or more party results.
          // Each iteration of the loop recognizes one party result

        while (k != pollData.size()  &&  pollData[k] != ',')
        {
              // A party result must start with a digit

            if (! isdigit(pollData[k]))
                return false;
            k++;

              // There might be a second digit

            if (k != pollData.size()  &&  isdigit(pollData[k]))
                k++;

              // There must be a party code

            if (k == pollData.size()  ||  ! isalpha(pollData[k]))
                return false;
            k++;
        }

          // If there's nothing after the state forecast, we're done

        if (k == pollData.size())
            return true;

          // There's a comma, so move past it

        k++;
    }

      // We get here if pollData ends with a comma

    return false;

}

//************************************
//  tallySeats
//************************************

int tallySeats(string pollData, char party, int& seatTally)
{
      // Define return values

    const int RET_OK             = 0;
    const int RET_BAD_SYNTAX     = 1;
    const int RET_BAD_PARTY      = 2;

      // A bad party character prevents tallying

    if (!isalpha(party))
        return RET_BAD_PARTY;

      // A pollData string with improper syntax prevents tallying

    if (!hasProperSyntax(pollData))
```

```
        return RET_BAD_SYNTAX;

      // We will later compare party codes in uppercase, so adjust party

    party = toupper(party);

      // We will tally seats in an int named result, and modify the seatTally
      // parameter only if processing the entire pollData string succeeds.

    int result = 0;

      // Each iteration of the loop deals with one state forecast.  Since we
      // know at this point the pollData string has proper syntax, we are
      // guaranteed there are one or two digits, etc.

    size_t k = 0;
    while (k != pollData.size())
    {
          // Skip over the state code (we know there must be one)

        k += 2;

          // Each iteration of the loop recognizes one party result

        while (k != pollData.size()  &&  pollData[k] != ',')
        {
                // Determine the party seat tally

            int partySeatTally = pollData[k] - '0';  // we know this is a digit
            k++;
            if (isdigit(pollData[k]))  // Is there a second digit?
            {
                partySeatTally = 10 * partySeatTally + pollData[k] - '0';
                k++;
            }

              // If the party code (we know there must be one) matches, record
              // the votes

            if (toupper(pollData[k]) == party)
                result += partySeatTally;
            k++;
        }

          // If there's another character, we know it's a comma, so move
          // past it

        if (k != pollData.size())
            k++;
    }

      // We've successfully processed the entire string, so set seatTally.

    seatTally = result;

    return RET_OK;
}

//*************************************
//  isValidUppercaseStateCode
//*************************************

// Return true if the argument is a two-uppercase-letter state code, or
// false otherwise.

bool isValidUppercaseStateCode(string stateCode)
{
    const string codes =
        "AL.AK.AZ.AR.CA.CO.CT.DE.FL.GA.HI.ID.IL.IN.IA.KS.KY."
        "LA.ME.MD.MA.MI.MN.MS.MO.MT.NE.NV.NH.NJ.NM.NY.NC.ND."
        "OH.OK.OR.PA.RI.SC.SD.TN.TX.UT.VT.VA.WA.WV.WI.WY";
    return (stateCode.size() == 2  &&
            stateCode.find('.') == string::npos  &&  // no '.' in stateCode
            codes.find(stateCode) != string::npos);  // match found
}
```