```
//  Find the expression in a string of digits that evaluate
//  to a particular answer
//  Created by Stewart Bracken on 11/13/13.
//  Copyright (c) 2013 Stewart Bracken. All rights reserved.
//

#include <iostream>
#include <unordered_map> //lookup table
#include <vector>
#include <sstream> //splitting strings

#include <string>

using namespace std;

void doScript(const string& script){

}

void yaddida(){
    vector<string> s;
    const vector<string>* scripts = &s;
    doScript(scripts->at(0));
}

/*******************************
 start test question 1
 *********************************/

//a string of an expression and it's evaluated numeric answer
typedef unordered_map<string, int> expr_lookup;

// split a string at every occurance of delim
// @param s - the string to split
// @param delim - the delimiter
// @param (optional) elems - the vector to push the delimited strings into
vector<string>& split(const string &s, char delim, vector<string>& elems) {
    stringstream ss(s);
    string item;
    while (getline(ss, item, delim)) {
        elems.push_back(item);
    }
    return elems;
}
```

```
vector<string> split(const string &s, char delim) {
    vector<string> elems;
    split(s, delim, elems);
    return elems;
}

template <typename T>
T StringToNumber ( const string &Text ){
    stringstream ss(Text);
    T result;
    return ss >> result ? result : 0;
}

// Evaluate a string expression.
// @param expr - a string expression with digits, +, or *
// @param lookup_table - an unordered_map<string, int>
// @return - 0 if blank string, or the answer
int evaluate(string expr, expr_lookup& lookup_table){
    auto lookup = lookup_table.find(expr);
    if (lookup != lookup_table.end()){
        return lookup->second; //already know solution
    }
    //actually evaluate it
    vector<string> plus = split(expr,'+'), mult;
    int out = 0;
    for(int i=0; i<plus.size();++i){
        mult.clear();
        mult = split(plus[i],'*');
        int m = StringToNumber<int>(mult[0]);
        for(int j=1;j<mult.size();++j){
            m *= StringToNumber<int>(mult[j]);
        }
        out += m;
    }
    lookup_table.insert({expr,out}); //SAVE ANSWER
    return out;
}


// Recursively try to evaluate combinations of digits
// @param expr_so_far - supply a blank string initially.
//                      It must have a + or * at the end or be blank.
// @param digits - pass in the initial digits string
// @param answer - the desired answer
// @param lookup_table - initally supply an empty expr_lookup table
// @return - the desired expression or an empty string if not found.
//expr_so_far already has a + or * at the end, or blank
string find_expression_rec(string& expr_so_far, string& digits, const int answer,
expr_lookup& lookup_table){
    for( int i = 1; i < digits.size(); ++i){
        string l = digits.substr(0, i); //new digits
        string r = digits.substr(i);  //new digits
        string plus = expr_so_far + l + "+";
        string mult = expr_so_far + l + "*";
        if (evaluate(plus + r, lookup_table) == answer) return plus + r;
        if (evaluate(mult + r, lookup_table) == answer) return mult + r;
        string out = find_expression_rec(plus, r, answer, lookup_table);
        if ( out != "" ) return out;
        out = find_expression_rec(mult, r, answer, lookup_table);
        if( out != "" ) return out;
    }
    return "";
}

// Attempt to find an expression by adding +/* within the digits string to
// evaluate to answer. Brute force :(
// @param digits - a string of digits
// @param answer - integer which you desire to find an expression for
// @return - either the expression or "no solution"
string find_expression(string digits, int answer){
    const int a = answer;
    string d = digits;
    string expr = "";
    expr_lookup lookup_table;
    string out = find_expression_rec(expr,d,a,lookup_table);
    if ( out == "" ) out = "no solution";
    return out;
}

int main(int argc, const char * argv[])
{
    string test1 = find_expression( "1231231234", 11353 );
    cout << test1 <<endl;

    string test2 = find_expression( "3456237490", 1185 );
    cout << test2 <<endl;

    string test3 = find_expression( "3456237490", 9191 );
    cout << test3 <<endl;
```

```
        return 0;
    }
```