```
/* lab3.c
 * Logan S. Lewis
 * lslewis
 * ECE 222, Fall 2016
 * MP3
 * Subject: ECE222-1,#3
 *
 * Purpose:  This machine problem places focus on manipulating strings and arrays
 *           as well as performing operations on a predefined finite field
 *           determined by specific characters and operators.
 *
 * Assumptions:
 *  #1   The user is prompted to enter a pseudo arithmetic command.  The
 *       input must be verified to be grammatically correct.
 *
 *  #2:  The string and character type library cannot be used under
 *       any circumstances.  You are encouraged to develop your own
 *       functions to perform any similar operations that are needed.
 *
 *  #3   No changes to the code in main.  Your code must be placed in
 *       functions.  Additional functions are encouraged.
 *
 * Bugs:
 *
 *
 *
 *
 * Notes:   Remember the null character (\0) must exist at the end of each string
 *
 *
 * See the ECE 222 programming guide
 *
 * If your formatting is not consistent you must fix it.  You can easily
 * reformat (and automatically indent) your code using the astyle
 * command.  In a terminal on the command line do
 *
 *     astyle --style=kr lab3.c
 *
 * See "man astyle" for different styles.  Replace "kr" with one of
 * ansi, java, gnu, linux, or google to see different options.  Or, set up
 * your own style.
 */

// do not include any additional libraries
#include <stdio.h>

// do not change these constants
#define MAXLINE 80
#define MAXOPER 13

// named constants and strings
enum operations { NOOP, ADD, MUL, DIV, POW};
const char *operation_str[] = {"Invalid", "+", "*", "/", "^"};

// function prototypes
int process_input(const char *input, char *op_left, char *op_right);
void calc_output(const char *op_l, int op, const char *op_r, char *result);
void strcopy (char *dest, const char*src);
int string_len (char *string);
char chng_case(char character);
int conv_to_int(char character);
char conv_to_char(int number);

// do not change any code in main.  We are grading based on the format
// of the output as given in the printfs in main.
int main()
{
    char input_line[MAXLINE];
    char left_operand[MAXOPER];
    char right_operand[MAXOPER];
    char answer[MAXOPER];
    int  operator;

    printf("\nMP3: Arithmetic on GF(47) with + * / ^ using letters\n");
    printf("Commands:\n\tabc+bbc\n\tturtle/frog\n\ttiger^one");
    printf("\tOperands are no more than 12 letters and no spaces\n");
    printf("\tCtrl-d to quit\n\n");
    printf("> ");

    // each call to fgets collects one line of input and stores in input_line
    // BEWARE: fgets includes the end-of-line character ’\n’ in input_line
    while (fgets(input_line, sizeof input_line, stdin) != NULL)
    {

        // clear for next round
        left_operand[0] = right_operand[0] = answer[0] = ’\0’;

        // check for valid grammar
        operator = process_input(input_line, left_operand, right_operand);

        if (operator == ADD || operator == MUL
                || operator == DIV || operator == POW)
        {

            // print parsed input
            printf("’%s’", left_operand);
            printf(" %s ", operation_str[operator]);
            printf("’%s’ => ", right_operand);

            // perform pseudo arithmetic
            calc_output(left_operand, operator, right_operand, answer);

            // print result
            printf("’%s’\n\n", answer);
        }
        else
        {
            printf("# %s", input_line);
        }
        printf("> ");
    }
    printf("\nGoodbye\n");
    return 0;
}

/* Parse input of the form SOS where S is a string and O is a character.
 *
 * A string S must consist of up to 12 valid symbols a-z and A-U.
 * The operand O must be one character from: + * / ^
 * Any other characters found in the input, including spaces, are
 * grammatically incorrect and invalidate the input.
 *
 * There must be no spaces anywhere in the input, including between
 * either SO, OS, or leading or trailing spaces.
 *
 * Input: The input string is collected using fgets.  Recall the end-of-line
 *        character is included in the input string and marks the end of
 *        the input.  This string must not be changed.
 *
 * Output: There are three outputs from this function.
 *
 *   The return value is one of NOOP, ADD, MUL, DIV, POW which are
 *      named constants.  If the input is invalid for any reason
 *      then the output must be NOOP.  Otherwise the return value
```

```c
 *      corresponds to operand 0.
 *
 *   If the input is grammatically correct, then two strings are also
 *   returned, one for each of the left and right operands.  If the input
 *   in invalid the two output strings are undefined.
 *   Strings are treated as if function was void, the memory locations are updated
 *   so, ideally, the strings are "returned"
 *
 *   user will try 78 characters, should not allow more than 78 characters
 */
int process_input(const char *input, char *op_left, char *op_right)
{

    int i = 0;
    int j = 0;
    int k = 0;
    char temp_string1[MAXLINE];
    char temp_string[MAXLINE];
    char temp_string2[MAXLINE];
    strcopy(temp_string1,input);
    int length = string_len(temp_string1);
    int l2 = 0;
    int l3 = 0;
    char op = ' ';
    int x = 0;

    // NOOP conditons:
    if(length > 78)
    {
        return NOOP;
    }

    for (i = 0; i < length; i++)        // string can not contain spaces
    {

        if (input[i] == ' ' || input[i] == '#')
        {
            return NOOP;
        }
    }

    int count = 0;

    while(input[count] != '\0')
    {
        // parse the input into the left and right operands
        if (input[count] == '+' || input[count] == '*' || input[count] == '/' || in
put[count] == '^')
        {

            op = input[count];
            strcopy (temp_string, input);
            temp_string[count] = '\0';  // stops the string once the operator is re
ached
            strcopy (op_left, temp_string);

            //strcopy(temp_string,input); // re-copies the input into the temporary
 string

            for (j = count + 1; j < length; j++)
            {

                temp_string2[k] = input[j]; // sets values past the operator into t
he temp_string and stops when reaching null character

                k++;
            }
            temp_string2[j-1] = '\0';
            strcopy(op_right, temp_string2);

            for (x = 0; x < string_len(op_right); x++)
            {

                if (op_right[x] == '\n')
                {
                    op_right[x] = '\0'; // replace the newline character with the n
ull character
                }
            }

            // after parsing into the two operands, check to see if the characters
are valid

            l2 = string_len(op_left);
            l3 = string_len(op_right);

            if ((l2 > 12 || l2 < 1) || (l3 > 12 || l3 < 1))
            {
                return NOOP;
            }

            for (j = 0; j < string_len(op_left); j++)
            {

                if (op_left[j] < 'A' || (op_left[j] > 'U' && op_left[j] < 'a') || o
p_left[j] > 'z' || op_left[j] == ' ')
                {

                    return NOOP;
                }
            }

            for (k = 0; k < string_len(op_right); k++)
            {

                if (op_right[k] < 'A' || (op_right[k] > 'U' && op_right[k] < 'a') |
| op_right[k] > 'z' || op_right[k] == ' ')
                {

                    return NOOP;
                }
            }
        }       // end of outer if statement

        count++;
    }   // end of while loop

    if (op == '+')
    {
        return ADD;
    }
    else if (op == '*')
    {
        return MUL;
    }
    else if (op == '/')
    {
        return DIV;
    }
    else if (op == '^')
    {
        return POW;
    }
    else
    {
        return NOOP;
    }
}

/* Pseudo mathematical opertions on the two operands work as follows.
 *
 * Each character is converted to an integer in the range 1...46, where a is 0,
 * b is 1, c is 2, ..., z is 25.  The operation is then performed using
 * math on a finite field with no carries.
 *
 * If the two input strings are not the same length, then each output character
 * beyond the length of the shorter string should be a copy of the character
 * from the longer string but with the opposite case.
 *
 * Input: The two operand strings and the operator are assumed to be valid (and
 *        are verified as valid in the parse_input function).
 *
 * Output: The final string generated by the above rules is stored in the
 *         output string named result.  The input strings must not be
 *         changed.
 */
void calc_output(const char *l_op, int op, const char *r_op, char *result)
{
    // convert each symbol into an integer (a=0 and U = 46)
    // addition: (X+Y) % 47 for each array position
    // multiplication: (X*Y) % 47
    // Inversion: X = (Y*Z) % 47 find Z, if Y is zero, set output to zero (’a’)
    // Power: (X^Y) %47, if Y is zero output = 1 (’b’)


    // compare the two string lengths
    // find shorter string
    // each output character past length of shorter string
    // should be a COPY of the character from longer string with opposite case

    char short_string[MAXLINE]; // gathers the shorter string so that it can then b
e copied with added characters
    char long_string[MAXLINE];  // the longer string, whose additonal characters mu
st be added to the end of the result, with switched case
    int extra_char = 0;         // extra characters past the length of the shorter
string
    int i = 0;
    int j = 0;
    int k = 0;
    char temp_string1[MAXLINE];
    char temp_string2[MAXLINE];
    int temp_result[MAXLINE];
    int length1 = 0;
    int length2 = 0;
    long value = 1;     // used for the power operation

    // set up finite field of integers
    int finite_field[47];

    for (i = 0; i < 26; i++)
    {
        finite_field[i] = ’a’ + i;
    }
    i = 0;
    for(j = 26; j < 47; j++)
    {
        finite_field[j] = ’A’ + i;
        i++;
    }
```

```
    strcopy(temp_string1, l_op);
    length1 = string_len(temp_string1);

    strcopy(temp_string2, r_op);
    length2 = string_len(temp_string2);


    // the following code is executed for strings of differing lengths
    if (length1 != length2)
    {
        if (length1 > length2)
        {
            strcopy(short_string,temp_string2);
            strcopy(long_string,temp_string1);
            extra_char = (length1 - length2);   // number of characters past the le
ngth of shorter string
        }
        else
        {
            strcopy(short_string,temp_string1);
            strcopy(long_string,temp_string2);
            extra_char = (length2 - length1);
        }

        int val1 = 0;
        int val2 = 0;
        int result_length = 0;

        if (op == 1)    // ’+’
        {

            for (i = 0; i < string_len(short_string); i++)
            {

                for (j = 0; j <= 46; j++)
                {

                    if (conv_to_int(temp_string1[i]) == finite_field[j])
                    {
                        val1 = j;
                    }
                    if (conv_to_int(temp_string2[i]) == finite_field[j])
                    {
                        val2 = j;
                    }

                }

                temp_result[i] = (val1 + val2) % 47;
                result_length++;
            }

            for (i = 0; i < result_length; i++)
            {

                result[i] = conv_to_char(temp_result[i]);

            }

            for (k = result_length; k <= (result_length + extra_char); k++)
            {
                result[k] = chng_case(long_string[k]);

            }
            result[string_len(result)] = ’\0’;

        }
    else if (op == 2)               // ’*’
    {

        for (i = 0; i < string_len(short_string); i++)
        {

            for (j = 0; j <= 46; j++)
            {

                if (conv_to_int(temp_string1[i]) == finite_field[j])
                {
                    val1 = j;
                }
                if (conv_to_int(temp_string2[i]) == finite_field[j])
                {
                    val2 = j;
                }

            }

            temp_result[i] = (val1 * val2) % 47;
            result_length++;
        }

        for (i = 0; i < result_length; i++)
        {

            result[i] = conv_to_char(temp_result[i]);

        }

        for (k = result_length; k <= (result_length + extra_char); k++)
        {
            result[k] = chng_case(long_string[k]);

        }
        result[string_len(result)] = ’\0’;

    }

    else if (op == 3)               // ’/’
    {
        int Z = 0;
        int inverse = 0;
        for (i = 0; i < string_len(short_string); i++)
        {

            for (j = 0; j <= 46; j++)
            {

                if (conv_to_int(temp_string1[i]) == finite_field[j])
                {
                    val1 = j;
                }
                if (conv_to_int(temp_string2[i]) == finite_field[j])
                {
                    val2 = j;
                }

            }
            if (val2 == 0)
            {
                temp_result[i] = 0;
                result_length++;
            }
            else
            {
                for (Z = 0; Z <= 47; Z++ )
                {
                    inverse = ((val2 * Z) % 47);

                    if (inverse == val1)
                    {
                        temp_result[i] = Z;
                        result_length++;
                        break;
                    }
                }
            }
        }

        for (i = 0; i < result_length; i++)
        {

            result[i] = conv_to_char(temp_result[i]);

        }

        for (k = result_length; k <= (result_length + extra_char); k++)
        {
            result[k] = chng_case(long_string[k]);

        }
        result[string_len(result)] = ’\0’;

    }

    else if (op == 4)               // ’^’
    {

        for (i = 0; i < string_len(short_string); i++)
        {

            for (j = 0; j <= 46; j++)
            {

                if (conv_to_int(temp_string1[i]) == finite_field[j])
                {
                    val1 = j;
                }
                if (conv_to_int(temp_string2[i]) == finite_field[j])
                {
                    val2 = j;
                }

            }
            if (val2 == 0)
            {
                temp_result[i] = 1;
                result_length++;
            }

            else if (val2 == 1)
            {
                temp_result[i] = val1;
                result_length++;
            }

            else
            {
                value = 1;
                for(k = 1; k < val2; k++)
                {
```

```c
                        if(value == 1)
                        {
                            value = val1;
                        }
                        value *= val1;
                    }
                    temp_result[i] = value % 47;
                    result_length++;
                }
            }

            for (i = 0; i < result_length; i++)
            {

                result[i] = conv_to_char(temp_result[i]);

            }

            for (k = result_length; k <= (result_length + extra_char); k++)
            {
                result[k] = chng_case(long_string[k]);

            }
            result[string_len(result)] = '\0';

        }


    }


    // the following is done for strings of the same length
    else
    {

        int val1 = 0;
        int val2 = 0;
        int result_length = 0;
        value = 0;

        if (op == 1)    // '+'
        {

            for (i = 0; i < length1; i++)
            {

                for (j = 0; j <= 46; j++)
                {

                    if (conv_to_int(temp_string1[i]) == finite_field[j])
                    {
                        val1 = j;
                    }
                    if (conv_to_int(temp_string2[i]) == finite_field[j])
                    {
                        val2 = j;
                    }

                }

                temp_result[i] = (val1 + val2) % 47;
                result_length++;
            }
        }

        else if (op == 2)               // '*'
        {

            for (i = 0; i < length1; i++)
            {

                for (j = 0; j <= 46; j++)
                {

                    if (conv_to_int(temp_string1[i]) == finite_field[j])
                    {
                        val1 = j;
                    }
                    if (conv_to_int(temp_string2[i]) == finite_field[j])
                    {
                        val2 = j;
                    }

                }
                temp_result[i] = (val1 * val2) % 47;
                result_length++;
            }
        }

        else if (op == 3)               // '/'
        {

            int Z = 0;
            int inverse = 0;
            for (i = 0; i < length1; i++)
            {

                for (j = 0; j <= 46; j++)
                {

                    if (conv_to_int(temp_string1[i]) == finite_field[j])
                    {
                        val1 = j;
                    }
                    if (conv_to_int(temp_string2[i]) == finite_field[j])
                    {
                        val2 = j;
                    }

                }
                if (val2 == 0)
                {
                    temp_result[i] = 0;
                    result_length++;
                }
                else
                {
                    for (Z = 0; Z <= 47; Z++ )
                    {
                        inverse = ((val2 * Z) % 47);

                        if (inverse == val1)
                        {
                            temp_result[i] = Z;
                            result_length++;
                            break;
                        }
                    }
                }
            }
        }


        else if (op == 4)               // '^'
        {

            for (i = 0; i < length1; i++)
            {

                for (j = 0; j <= 46; j++)
                {

                    if (conv_to_int(temp_string1[i]) == finite_field[j])
                    {
                        val1 = j;
                    }
                    if (conv_to_int(temp_string2[i]) == finite_field[j])
                    {
                        val2 = j;
                    }

                }
                if (val2 == 0)
                {
                    temp_result[i] = 1;
                    result_length++;
                }
                else if (val2 == 1)
                {
                    temp_result[i] = val1;
                    result_length++;
                }
                else
                {
                    value = 1;
                    for(k = 1; k < val2; k++)
                    {
                        if(value == 1)
                        {
                            value = val1;
                        }
                        value *= val1;
                    }
                    temp_result[i] = value % 47;
                    result_length++;
                }
            }
        }

        for (i = 0; i < result_length; i++)
        {
            result[i] = conv_to_char(temp_result[i]);

        }
        result[result_length] = '\0';   // set null character to the end of the res
ult string

    }

}

/* This function is used to copy a source string into another string, the desinatio
n
 * The loop iterates until the null character is found, ending the string
 * Then places the null character at the end of the new string and updates the memo
ry location
 */
void strcopy (char *dest, const char *src)
{
    int i = 0;
    while (src[i] != '\0')
```

```
    {
        dest[i] = src[i];
        i++;
    }
    dest[i] = '\0';
}

/* The string length funciton takes an input string
 * and returns the length of the string as an integer value
 */
int string_len (char *string)
{
    int i = 0;
    int length = 0;

    while (string[i] != '\0')
    {
        length++;
        i++;
    }
    return length;
}

// converts a given lower case letter to upper case
// converts a given upper case letter to lower case
// capital letters are smaller than lower case letters as viewed in the ASCII table
char chng_case(char character)
{
    char c = character;

    if (character >= 'a' && character <= 'z')
    {
        c -= 'a' - 'A';
    }

    else if (character >= 'A' && character <= 'Z')
    {
        c += 'a' - 'A';
    }

    return c;
}

// this function takes a letter and converts it to an integer value for the finite
field
int conv_to_int(char letter)
{
    int num_val = 0;

    if (letter >= 'a'  && letter <= 'z')
    {
        num_val = (int)letter;
    }

    else if (letter >= 'A' && letter <= 'U')
    {
        num_val = (int)letter;
    }
    return num_val;

}

// this function takes a number and converts it into a character value for results
char conv_to_char(int number)
{
    char char_value;
    if (number >= 0 && number <= 25)
    {
        char_value =  (char) (number + 97);
    }

    else if (number >= 26 && number <= 46)
    {
        char_value = (char) (number + 39);
    }

    return char_value;
}
```