```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <string.h>
#include <stdlib.h>
#include "lcd.c"

/*
 * key_pressed : The key last pressed in binary form. Taken from PINB.
 * key         : The key pressed in integer form. Converted from
key_pressed, and used in the actual computation.
 * character   : The current value of the character to be transmitted to
the LCD.
 * timer_count : The elapsed wait count of our timer.
 */
unsigned char   key_press,
key = 0,
character,
timer_count;

// Loopers
unsigned char i, c;

/*
 * count[i] gives you the number of times key number i+1 is pressed in
the current state cycle.
 * tchar[i] gives you the top character associated with key number i+1.
 * For example, the top character associated with key #1 is 'a'. Pressing
key #1 again immediately sets count[0] to 2 from 1.
 * And the character to be printed becomes 'a' + 1 = 'b'.
 */
unsigned char   count[8] = {0, 0, 0, 0, 0, 0, 0, 0},
tchar[8] = {'a', 'e', 'i', 'm', 'q', 'u', 'y', '0'};

/*
 * We can have a maximum input string length of 10, and we keep the 11th
byte for the terminating null character.
 * wlength keeps track of the current string length, so we may know when
to add the terminating null.
 */
char   input[11], wlength=0;

/*
 * The hash key stores the hash value associated with each character. It
is advisable to keep all of the "weights" prime
 * to minimize hash collisions. Also keep them unique, as apart as
possible.
 */
int hashKey[] = { 7, 11, 13, 17, 29, 37, 43, 41, 57, 59, 61, 67, 71, 79,
83, 87, 97, 101, 203, 313, 497, 421, 211, 197, 139, 241 };

// Compute the integer hash of a word using the hash key. This simply
adds up all the hash values for the characters in the string.
int  hash(char* word);
```

```
/*
 * Check if string1 is a permutation of string2. This is an expensive
operation,
 * and should only be used if you are reasonably sure about it being a
permutation.
 */
int  isPerm(char* string1, char* string2);

/*
 * Transmits the given character to the LCD. Keeps track of the word
length and prints the given character
 * to the right of the previously printed character, this way you can
"type" a word and see it on the LCD.
 */
void transmit(char c, int inPlace);

/*
 * This handles the timer delays, and causes any still-in-state-cycle
character typed previously to be printed out to the LCD
 * if you enter a new character by pressing a different key, thus
breaking the old state cycle and starting a new one for the new key.
 */
void set_timer(void);

void marquee(void);

void _delay(int time);
// The word dictionary. Can be extended as much as desired (within memory
constraints), just remember to update the dictSize correspondigly.
char *dict[] = {
    "god",
    "dog",
    "lame",
    "zigbee",
    "dirac",
    "laplace",
    "euler",
    "zeta",
    "gamma",
    "magma",
    "contour",
    "idiot",
    "muon",
    "male",
    "female",
    "insert",
    "dummy",
    "muddy",
    "deaf",
    "fade",
    "cafe",
    "ship",
    "face",
    "cat",
    "act",
    "embedded",
    "intel",
    "kids",
    "professor",
    "disk"

};

// Maintains a count of the dictionary size.
int dictSize = 30;

char isBackspacePressed = 0;

// Overflow handler, most of the state machine is implemented here.
SIGNAL(SIG_OVERFLOW0)
{
    if( (timer_count + 1) % 4 == 0)
    {
        // An overflow occured, so calculate the character to be printed
from the state.
        for(i = 0; i < 8; i++)
        {
            if(count[i] != 0)
            {
                character = tchar[i] + (count[i] - 1)%4;
            }
        }

        // Transmit the character.
        if(!isBackspacePressed)
            transmit(character, 0);
        else
            isBackspacePressed = 0;

        // Reset the state machine.
        for(i = 0; i < 8; i++)
        {
            count[i] = 0;
        }
        TCCR0 = 0x00;
        timer_count = 0;
    }

    else
    {
        // Update state.
        TCNT0 = 4;
        timer_count++;
    }
}
```

```
int main(void)
{
    int i, inputHash;
    int matchCount = 0;
    // Allocate and populate a hash table of the same length as the
dictionary, and do a one-time computation of the hash for further usage.
    int *dictHash = (int *)malloc(dictSize);
    for(i = 0; i < dictSize; ++i)
    {
        dictHash[i] = hash(dict[i]);
    }

    // Preliminary setup.
    // PORTA is for output (LCD).
    // PORTB is for input (switches).

    DDRB  = 0x00;
    PORTB = 0xFF;
    DDRA  = 0xFF;
    TCCR0 = 0x00;
    TIMSK = 0x01;
    sei();

    // Start your LCD
    lcd_init(LCD_DISP_ON_CURSOR);
    lcd_gotoxy(1,0);
    lcd_puts("Welcome To");
    lcd_gotoxy(1,1);
    lcd_puts("eUnagram");
    _delay(8);

    marquee();

    while(1)
    {
        // Store PINB's state in key_press before you lose it.
        key_press = PINB;

        // Decide what key is pressed and what action to take, based on
different values of key_pressed.
        switch(key_press)
        {
            case 0b01111111:
            {
                key = 1;
                set_timer();
                break;
            }

            case 0b10111111  :
            {
                key = 2;
                set_timer();
                break;
            }

            case 0b11011111  :

            {
                key = 3;
                set_timer();;
                break;
            }

            case 0b11101111  :
            {
                key = 4;
                set_timer();
                break;
            }

            case 0b11110111  :
            {
                key = 5;
                set_timer();
                break;
            }

            case 0b11111011  :
            {
                key = 6;
                set_timer();
                break;
            }

            case 0b11111101  :
            {
                key = 7;
                set_timer();
                break;
            }

            case 0b11111110  :
            {
                // Pressed the 'Submit' key. We're not going to increase
the size of the input string now, so append the terminating null
                // to the input array to make it ready for processing.
                input[(int)wlength] = '\0';

                // Compute the hash of the input word.
                inputHash = hash(input);

                // Now loop through the dictionary and compare the hashes
with the hash of the input word.
                for(i = 0; i < dictSize; ++i)
                {
                    // If you find a suspect (same hash), then there's a
chance that it is a solution. So check if it is a permutation.
```

```
                    if(inputHash == dictHash[i])
                    {
                        // If it is a premutation, then it is a solution.
Print it and wait for one second before proceeding to display another
possible answer.
                        if(isPerm(input, dict[i]))
                        {
                            // Print this solution in the bottom row.
                            lcd_gotoxy(0,1);
                            lcd_puts(dict[i]);

                            // 1-second (4 milliseconds) wait.
                            _delay(4);

                            // Increment success count
                            matchCount++;

                        }
                    }
                }

                wlength = 0;
                if (matchCount == 0)
                {
                    lcd_gotoxy(0,1);
                    lcd_puts("No match found.");
                }

                _delay(4);
                lcd_clrscr();
                break;
            }

            default: break;
        } // End switch
    } // End while(1)

    // Free the hash table.
    free(dictHash);
    return 0;
}


// Definitions

int hash(char* word)
{
    int i, len, sum = 0;
    len = strlen(word);

    for(i = 0; i < len; ++i)
        sum += hashKey[ *(word + i) - 97 ];

    return sum;
```

```
}

int isPerm(char* string1, char* string2)
{
    if( hash(string1) != hash(string2) )
        return 0;

    char str1[100];
    char str2[100];

    strcpy(str1,string1);
    strcpy(str2,string2);

    int len, i, j;
    len = strlen(str1);

    for(i = 0; i < len; ++i)
    {
        for(j = 0; j < len; ++j)
        {
            if(str1[i] == str2[j])
            {
                str1[i]='0';
                str2[j]='0';
                continue;
            }
        }
    }

    int wrongs = 0;
    for(i = 0; i < len; ++i)
    {
        if( (str1[i] == str2[i]) && (str1[i] == '0') )
            wrongs += 0;
        else
        {
            wrongs = 1;
            return 0;
        }
    }
    return 1;
}

void transmit(char character, int inPlace)
{

    lcd_gotoxy(wlength, 0);

    if(!inPlace)
    {
        input[(int)wlength] = character;
        wlength++;
    }
    lcd_putc(character);
    lcd_gotoxy(wlength, 0);
}


void set_timer()
{
    TCCR0 = 0x00;

    // Add a suitable delay so that if the user presses the key a tad too
long, it won't be taken as another input. 300ms seems good enough.
    _delay_ms(300);

    count[key - 1]++;
    for(i = 0; i < 8; i++)
    {
        // If another key is pressed while still in the state machine of
a previous key, kill that previous state machine by setting count as 0,
        // and transmit the old key's results to the LCD.
        if(i != key - 1)
        {
            if(count[i] != 0)
            {
                character = tchar[i] + (count[i] - 1)%4;
                transmit(character, 0);
                count[i] = 0;
            }
        }

        // If the same key is pressed as before, just echo the updated
state to the LCD. This way, you can "see as you type"
        else
        {
            if( ( (key == 7) && (count[key -1 ] )%3  == 0))
            {
                 //wlength--;
                transmit(' ', 1);
                wlength--;
                transmit(' ', 1);
                count[key -1] = 0;

                isBackspacePressed = 1;
            }
            else
            {
                character = tchar[i] + (count[i] - 1)%4;
                transmit(character, 1);
            }
        }
    }

    TCCR0 = 0x05;
    TCNT0 = 4;
}
```

```
void marquee()
{
    lcd_clrscr();
    lcd_gotoxy(0,1);
    lcd_puts("Poojan Shah");
    _delay(3);
    lcd_clrscr();
    lcd_gotoxy(0,0);
    lcd_puts("Poojan Shah");

    lcd_gotoxy(0,1);
    lcd_puts("Aditya Bhatt");
    _delay(3);
    lcd_clrscr();
    lcd_gotoxy(0,0);
    lcd_puts("Aditya Bhatt");

    lcd_gotoxy(0,1);
    lcd_puts("Athatrva Patel");
    _delay(3);
    lcd_clrscr();
    lcd_gotoxy(0,0);
    lcd_puts("Atharva Patel");

    lcd_gotoxy(0,1);
    lcd_puts("Pratik Pandey");
    _delay(3);
    lcd_clrscr();

    lcd_gotoxy(0,0);
    lcd_puts("Pratik Pandey");

    lcd_gotoxy(0,1);
    lcd_puts("Varun Vyas");
    _delay(3);
    lcd_clrscr();
}

void _delay(int time)
{
    for(i = 0; i < time; i++)
        _delay_ms(250);
}
```