목차

1. 수행한 과제

IR Sensor

- IR 센서를 아두이노에 연결한다.
- 다음 조건을 만족하는 계산기를 설계한다.

● 키트에 포함된 리모콘을 계산을 위한 키패드로 활용한다.

- 필요한 경우, 숫자를 제외한 다른 키들은 기능(연산자)들을 재정의할 수 있다.
- 예를 들면, CH- 키는 곱셈, CH+ 키는 나눗셈...

● 입력된 숫자와 계산 결과 값은 시리얼 모니터에 표시한다.

- 시리얼 모니터에 표시 방법은 자유 선택.
- 예를 들면, '4', '+', '3', '=' 이 입력된 경우, 시리얼 모니터상에 '4 + 3 = 7'과 같이 표현할 수 있다.

● 리모콘을 통해 입력되는 숫자와 연산자를 사용하여 계산하고, 그 결과 값을 시리얼 모니터에 출력한다.

2. 부품 리스트

핵심 사용 부품

아두이노 우노 R3(Arduino Uno R3) x1

IR 리모컨 수신 센서(IR Receiver Sensor) x1

아두이노 IR 리모콘(IR Remote Control) x1

16x2 문자형 LCD(16x2 Text LCD with Soldered Header Pin) x1

수동 부저(Passive Buzzer) x1

10K 가변 저항(10K Variable Resistance) x1

기타 사용 부품

830 포인트 브레드보드(830 Point Breadboard) x1

220Ω 저항(220Ω Resistance) x2

330Ω 저항(330Ω Resistance) x1

10kΩ 저항(10kΩ Resistance) x1

점퍼 케이블(Jumper Cable)

3. 구현하고자 하는 기능 및 설명

Core – 1. IR Sensor와 IR 리모콘을 이용한 계산기 기능

IR Sensor를 통해 리모콘을 계산을 위한 키패드로 사용하는 기본적인 기능(리모콘 입력, 직렬 통신 출력)을 가진 계산기를 구현하였다.

Idea – 1. 16x2 문자형 LCD 모듈을 이용한 계산기의 물리적 디스플레이 추가

16x2 문자열 LCD 모듈을 아두이노에 연결하여 입력 중인 정보를 확인하고, 계산 디스플레이로 사용할 수 있도록 구현하였다.

Idea – 2. 범용 사칙 연산 파싱 지원

단순히 덧셈, 뺄셈만 수행하는 것이 아니라, 중위 표현식인 사용자의 입력을 후위 표현식으로 변환한 뒤 유사 스택(배열 활용)을 이용하여 계산하는 알고리즘을 이용하여 사용자의 사칙연산 계산식의 결과를 계산하는 기능을 추가하였다.

Idea – 3. 계산기에 부가 기능 추가

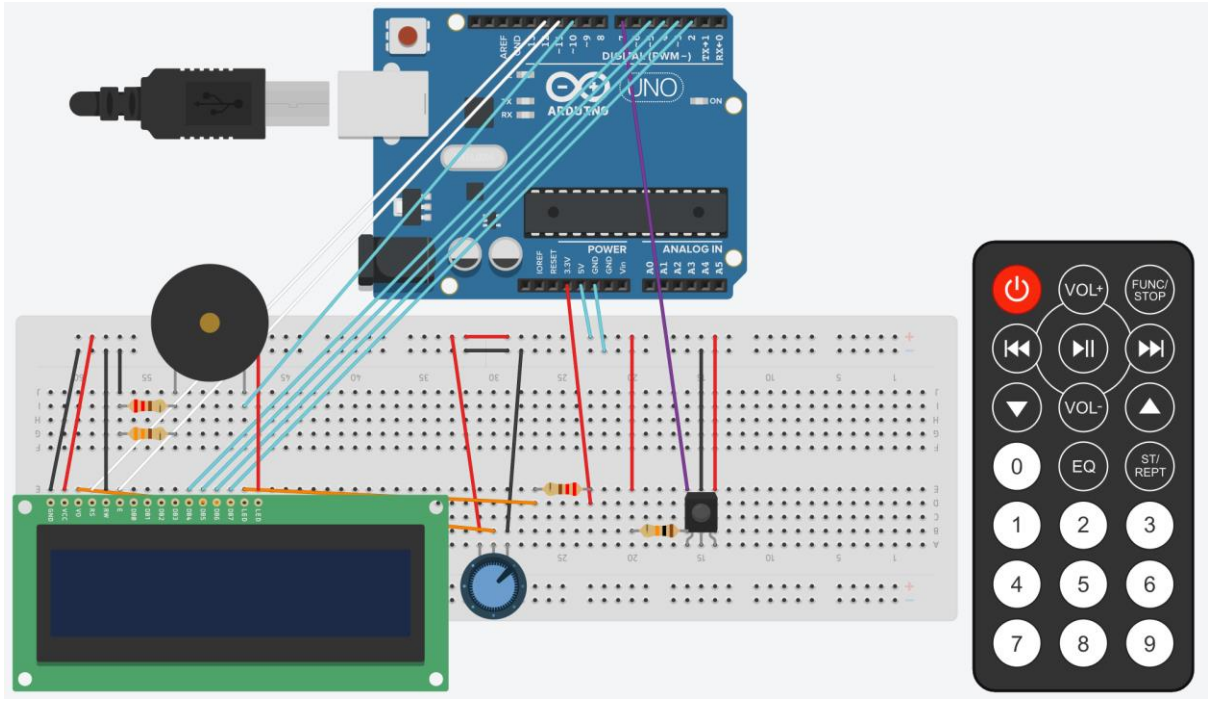
100+: '<' (Cursor left)
200+: '>' (Cursor right)
<<: 'B' (Backspace),
>>: 'C' (Clear),
CH: '(' (Left parentheses)
CH+: ')' (Right parentheses)
Play: '/' (Divide),
EQ: '*' (Multiply)

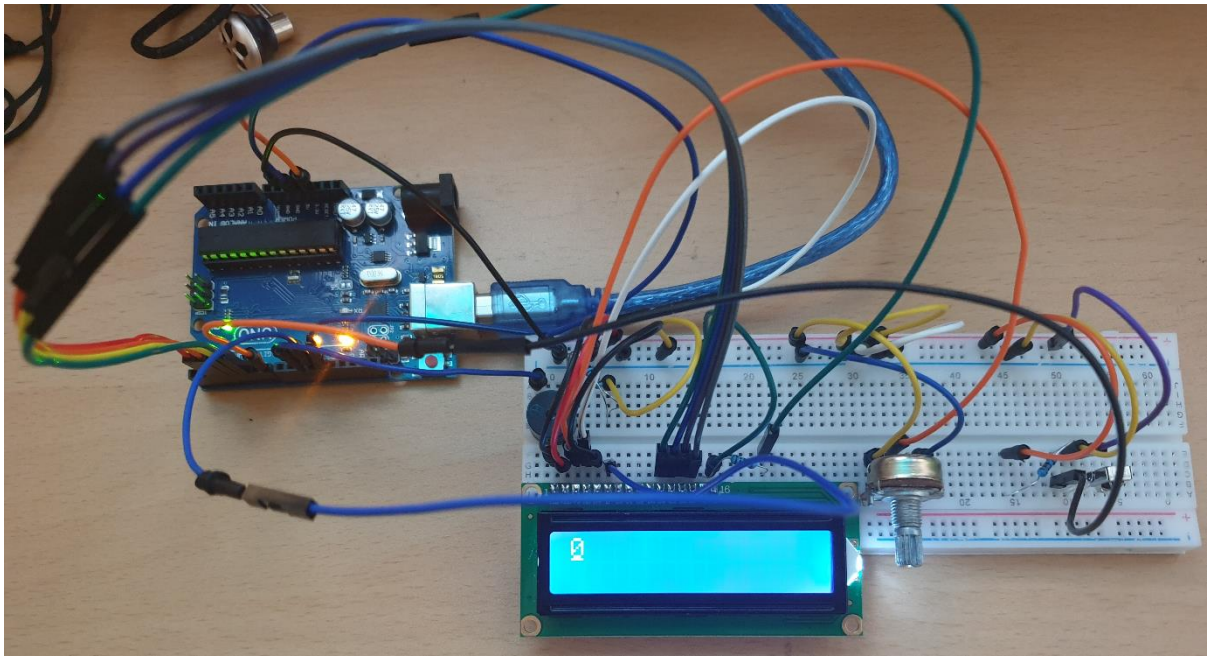
IR 리모콘의 숫자 입력 버튼을 제외하고 남는 버튼을 계산기의 다른 기능을 위한 버튼으로 처리하여 다양한 기능을 구현하였다.

Idea – 4. 수동 부저를 이용한 계산기 작동음 추가

수동 부저를 아두이노 우노 보드에 연결하고, IR 리모콘의 버튼을 눌렀을 때 정상 입력과 비정상 입력시 각각의 상황을 인지할 수 있는 알림음을 출력하여 계산기 사용간 청각적 피드백을 받을 수 있도록 구현하였다.

4. 부품과 아두이노 우노 보드와의 연결도





5. 프로그램 소스

GitHub: https://github.com/refracta/koreatech-assignment/tree/master/MicroprocessorNPractice/Assignment3

Arduino-IRremote: https://github.com/Arduino-IRremote/Arduino-IRremote

IRSensor.ino

```
/**
 * refracta - 마이크로프로세서및실습 (CSE124-02)
 * 과제 3 IR Sensor 소스 코드
 */
#include <IRremote.h>
#include <LiquidCrystal.h>

#define BUZZER_PIN 10
#define RECEIVE_PIN 7

#define LC_D0_PIN 5
#define LC_D1_PIN 4
#define LC_D2_PIN 3
#define LC_D3_PIN 2
#define LC_ENABLE_PIN 11
#define LC_RS_PIN 12

#define IR_CH_MINUS 0xFFA25D
#define IR_CH 0xFF629D
#define IR_CH_PLUS 0xFFE21D
#define IR_PREV 0xFF22DD
#define IR_NEXT 0xFF02FD
#define IR_PAUSE 0xFFC23D
#define IR_MINUS 0xFFE01F
#define IR_PLUS 0xFFA857
#define IR_EQ 0xFF906F
#define IR_0 0XFF6897
#define IR_100 0xFF9867
#define IR_200 0xFFB04F
#define IR_1 0xFF30CF
#define IR_2 0xFF18E7
#define IR_3 0xFF7A85
#define IR_4 0xFF10EF
#define IR_5 0xFF38C7
```

```
#define IR_6 0xFF5AA5
#define IR_7 0xFF42BD
#define IR_8 0xFF4AB5
#define IR_9 0xFF52AD
#define IR_NUM_OF_BUTTONS 21

int BUTTON_VALUES[] = {IR_CH_MINUS, IR_CH, IR_CH_PLUS, IR_PREV,
IR_NEXT, IR_PAUSE, IR_MINUS, IR_PLUS, IR_EQ, IR_0,
                       IR_100, IR_200, IR_1, IR_2, IR_3, IR_4, IR_5,
IR_6, IR_7, IR_8, IR_9};
char CALCULATOR_SYMBOLS[] = {'E', '(', ')', 'B', 'C', '/', '-', '+',
'*', '0', '<', '>', '1', '2', '3', '4', '5', '6',
                             '7', '8', '9'};

char toChar(int signal) {
    for (int i = 0; i < IR_NUM_OF_BUTTONS; i++) {
        if (BUTTON_VALUES[i] == signal) {
            return CALCULATOR_SYMBOLS[i];
        }
    }
    return 'W0';
}

LiquidCrystal lcd(LC_RS_PIN, LC_ENABLE_PIN, LC_D0_PIN, LC_D1_PIN,
LC_D2_PIN, LC_D3_PIN);
IRrecv irr(RECEIVE_PIN);
decode_results results;

#define SUCCESS_TUNE() playTone('g', 5, 200);
#define FAIL_TUNE() playTone('F', 6, 100); playTone('F', 6, 100);

/*
 * data from
https://www.arduino.cc/en/Tutorial/BuiltInExamples/toneMelody
 * 아두이노 튜토리얼 문서의 헤더 파일을 배열 형태로 정리한 것, 각
음계의 진동수의 상수 정의이다.
 */
const int frequencies[] =
        {
                31,
// octave 0
                33, 35, 37, 39, 41, 44, 46, 49, 52, 55, 58, 62,
// octave 1
```

```
78                  65, 69, 73, 78, 82, 87, 93, 98, 104, 110, 117, 123,
79 // octave 2
80                  131, 139, 147, 156, 165, 175, 185, 196, 208, 220, 233,
81 247,               // octave 3
82                  262, 277, 294, 311, 330, 349, 370, 392, 415, 440, 466,
83 494,               // octave 4
84                  523, 554, 587, 622, 659, 698, 740, 784, 831, 880, 932,
85 988,               // octave 5
86                  1047, 1109, 1175, 1245, 1319, 1397, 1480, 1568, 1661,
87 1760, 1865, 1976,    // octave 6
88                  2093, 2217, 2349, 2489, 2637, 2794, 2960, 3136, 3322,
89 3520, 3729, 3951,    // octave 7
90                  4186, 4435, 4699, 4978
91 // octave 8
92         };
93
94
95 // "도 도# 레 레# 미 파 파# 솔 솔# 라 라# 시" 순의 출력 처리용
96 character 배열
97 const char notes[] = {'c', 'C', 'd', 'D', 'e', 'f', 'F', 'g', 'G',
98 'a', 'A', 'b'}; // length = 12
99
100 // note 에 해당하는 notes 의 index 를 반환한다.
101 int getNoteIndex(char note) {
102     for (int i = 0; i < 12; i++) {
103         if (notes[i] == note) {
104             return i;
105         }
106     }
107     return -1;
108 }
109
110 // note, octave 의 출력으로 duration 동안 부저에 소리를 출력하는 함수
111 void playTone(char note, int octave, int duration) {
112     int frequency = frequencies[getNoteIndex(note) - 11 + octave *
113 12];
114     tone(BUZZER_PIN, frequency, duration);
115     delay(duration);
116     noTone(BUZZER_PIN);
117 }
118
119 int getReceiveSignal() {
120     int value = 0;
```

```
121     if (irr.decode(&results)) {
122         if (results.decode_type == NEC) {
123             value = results.value;
124         }
125         irr.resume();
126     }
127     return value;
128 }
129
130 char screen[32] = {0,};
131 int cursor = 0;
132
133 void clear(bool withZero) {
134     for (int i = 0; i < 32; i++) {
135         screen[i] = 0;
136     }
137     lcd.clear();
138     if (withZero) {
139         lcd.print('0');
140     }
141     lcd.home();
142     cursor = 0;
143 }
144
145 #define IS_ADD_TYPE(c) ((c) == '+' || (c) == '-')
146 #define IS_MULTI_TYPE(c) ((c) == '*' || (c) == '/')
147 #define IS_OPERATOR(c) (IS_ADD_TYPE(c) || IS_MULTI_TYPE(c))
148 #define IS_PARENTHESES(c) ((c) == ')' || (c) == ')')
149 #define IS_DIGIT(c) ('0' <= (c) && (c) <= '9')
150 #define STACK_SIZE 64
151
152 // 주어진 표현식의 소괄호 쌍이 닫혀있는지 검사한다.
153 bool isValidParentheses(char *target) {
154     int length = strlen(target);
155     if (length == 0) { return true; }
156     // if (length % 2 == 1) { return false; }
157     char stack[32] = {0,};
158     int stackCursor = 0;
159     for (int i = 0; i < length; i++) {
160         char c = target[i];
161         if (c == '(') {
162             stack[stackCursor++] = c;
163         } else if (c == ')') {
```

```
164             if (stackCursor == 0) {
165                 return false;
166             } else {
167                 stack[--stackCursor] = 'W0';
168             }
169         }
170     }
171     return stackCursor == 0;
172 }
173
174 // 주어진 표현식을 검증한다. (소괄호 쌍 검사, 연산자와 피연산자가
175 올바른 형태로 나타났는지 검사)
176 bool isValidExpression(char *target) {
177     if (!isValidParentheses(target)) {
178         return false;
179     }
180     int operatorCount = 0;
181     int operandCount = 0;
182     char prev = '(';
183     int length = strlen(target);
184     for (int i = 0; i < length; i++) {
185         char c = target[i];
186         if (IS_OPERATOR(c)) {
187             if (operandCount == 0) {
188                 return false;
189             }
190             operandCount = 0;
191             ++operatorCount;
192             if (operatorCount > 1) {
193                 return false;
194             }
195         } else if (!IS_DIGIT(prev) && IS_DIGIT(c)) {
196             operatorCount = 0;
197             ++operandCount;
198             if (operandCount > 1) {
199                 return false;
200             }
201         }
202         prev = c;
203     }
204     return operatorCount == 0;
205 }
206
```

```
207 // 함수 포인터로 이용되는 표현식 계산 소기능 함수들
208 bool isOperatorPredicate(char c) {
209     return IS_OPERATOR(c);
210 }
211
212 bool isLPPredicate(char c) {
213     return c == '(';
214 }
215
216 bool isMultiTypePredicate(char c) {
217     return IS_MULTI_TYPE(c);
218 }
219
220 bool isLPAndAddTypePredicate(char c) {
221     return c == '(' || IS_ADD_TYPE(c);
222 }
223
224 bool alwaysFalsePredicate(char c) {
225     return false;
226 }
227
228 String postfix;
229
230 // stack 에서 postfix String 에 덧붙일 요소들을 이동시킨다.
231 int moveOperatorsToPostfix(char *stack, int stackCursor, bool
232 (*pushPredicate)(char), bool (*stopPredicate)(char)) {
233     int originalStackCursor = stackCursor;
234     while (stackCursor != 0) {
235         char top = stack[stackCursor - 1];
236         if (pushPredicate(top)) {
237             postfix += top;
238             postfix += ' ';
239             stackCursor--;
240         } else if (stopPredicate(top)) {
241             break;
242         }
243     }
244     return stackCursor - originalStackCursor;
245 }
246
247 // infix 표현식을 postfix 형태로 변환시킨다.
248 String toPostfix(char *infix) {
249     postfix = "";
```

```
250     char stack[STACK_SIZE];
251     int stackCursor = 0;
252     int length = strlen(infix);
253     int n = 0;
254     bool processingDigit = false;
255     for (int i = 0; i < length; i++) {
256         char c = infix[i];
257         if (IS_DIGIT(c)) {
258             processingDigit = true;
259             n = (c - '0') + n * 10;
260         } else if (c == '(') {
261             stack[stackCursor++] = c;
262         } else if (c == ')' || IS_OPERATOR(c)) {
263             if (processingDigit) {
264                 postfix += String(n) + " ";
265                 n = 0;
266                 processingDigit = false;
267             }
268             if (IS_OPERATOR(c)) {
269                 if (stackCursor == 0) {
270                     stack[stackCursor++] = c;
271                 } else {
272                     if (IS_ADD_TYPE(c)) {
273                         stackCursor += moveOperatorsToPostfix(stack,
274 stackCursor, isOperatorPredicate, isLPPredicate);
275                     } else {
276                         stackCursor += moveOperatorsToPostfix(stack,
277 stackCursor, isMultiTypePredicate,
278
279 isLPAndAddTypePredicate);
280                     }
281                     stack[stackCursor++] = c;
282                 }
283             } else {
284                 stackCursor += moveOperatorsToPostfix(stack,
285 stackCursor, isOperatorPredicate, isLPPredicate);
286                 int top = stack[stackCursor - 1];
287                 if (top == '(') { stackCursor--; }
288             }
289         }
290     }
291     if (processingDigit) {
292         postfix += String(n) + " ";
```

```
293     }
294     stackCursor += moveOperatorsToPostfix(stack, stackCursor,
295 isOperatorPredicate, alwaysFalsePredicate);
296     postfix = postfix.substring(0, postfix.length() - 1);
297     return postfix;
298 }
299
300 // postfix 형태를 가지는 표현식의 실제 연산 결과를 얻는다.
301 int evaluatePostfix(String postfix) {
302     int stack[STACK_SIZE];
303     int stackCursor = 0;
304     int n = 0;
305     bool processingDigit = false;
306     int length = postfix.length();
307     for (int i = 0; i < length; i++) {
308         char c = postfix[i];
309         if (IS_DIGIT(c)) {
310             processingDigit = true;
311             n = (c - '0') + n * 10;
312         } else if (c == ' ') {
313             if (processingDigit) {
314                 stack[stackCursor++] = n;
315                 n = 0;
316                 processingDigit = false;
317             }
318         } else if (IS_OPERATOR(c)) {
319             int n2 = stack[--stackCursor];
320             int n1 = stack[--stackCursor];
321             switch (c) {
322                 case '+':
323                     stack[stackCursor++] = n1 + n2;
324                     break;
325                 case '-':
326                     stack[stackCursor++] = n1 - n2;
327                     break;
328                 case '*':
329                     stack[stackCursor++] = n1 * n2;
330                     break;
331                 default:
332                     stack[stackCursor++] = n1 / n2;
333                     break;
334             }
335         }
```

```
336     }
337     return stack[--stackCursor];
338 }
339
340 bool evalStatus = true;
341
342 // 계산기 표현식을 매개변수로 받아 해당 표현식의 평가값을 반환한다.
343 구문, 문법 오류가 나지 않으면 evalStatus의 값을 false로, 오류가 나면
344 true로 변경한다.
345 int evaluate(char *target) {
346     if (!isValidExpression(target)) {
347         evalStatus = false;
348         return 0;
349     } else {
350         evalStatus = true;
351         return evaluatePostfix(toPostfix(target));
352     }
353 }
354
355 // 현재 계산기 화면을 직렬 통신 출력으로 내보낸다.
356 void printToSerial() {
357     Serial.println("<CalculatorScreen>");
358     Serial.print('[');
359     for (int i = 0; i < 16; i++) {
360         if (screen[i]) {
361             Serial.print(screen[i]);
362         } else {
363             Serial.print(' ');
364         }
365     }
366     Serial.println(']');
367     Serial.print('[');
368     for (int i = 16; i < 32; i++) {
369         if (screen[i]) {
370             Serial.print(screen[i]);
371         } else {
372             Serial.print(' ');
373         }
374     }
375     Serial.println(']');
376 }
377
378 // LCD, IR 수신부 초기화
```

```
379 void setup() {
380     lcd.begin(16, 2);
381     lcd.blink();
382     clear(true);
383     irr.enableIRIn();
384     Serial.begin(9600);
385 }
386
387 int eval = 0;
388 bool waitClear = false;
389
390 // 사용자의 계산기 입력이 처리되는 루프
391 void loop() {
392     lcd.cursor();
393     char input = toChar(getReceiveSignal());
394     if (input) {
395         if (waitClear) {
396             waitClear = false;
397             clear(true);
398         }
399         switch (input) {
400             case 'C':
401                 // Clear
402                 SUCCESS_TUNE();
403                 clear(true);
404                 break;
405             case 'B':
406                 // Backspace
407                 if (cursor > 0) {
408                     SUCCESS_TUNE();
409                     if (screen[cursor] && cursor < 32) {
410                         int c = cursor;
411                         for (; c < 32; c++) {
412                             if (screen[c]) {
413                                 screen[c - 1] = screen[c];
414                                 lcd.setCursor((c - 1) % 16, (c - 1) /
415 16);
416                                 lcd.print(screen[c - 1]);
417                             } else {
418                                 break;
419                             }
420                         }
421                         screen[c - 1] = '₩0';
```

```
422                     if (c - 2 == 15) {
423                         lcd.setCursor(0, 1);
424                     }
425                     lcd.print(' ');
426                     cursor--;
427                     lcd.setCursor(cursor % 16, cursor / 16);
428                 } else {
429                     cursor--;
430                     screen[cursor] = 'WO';
431                     lcd.setCursor(cursor % 16, cursor / 16);
432                     lcd.print(' ');
433                     lcd.setCursor(cursor % 16, cursor / 16);
434                 }
435             } else {
436                 FAIL_TUNE();
437             }
438             break;
439         case 'E':
440             // Enter
441             if (cursor == 0 && !screen[cursor]) {
442                 FAIL_TUNE();
443                 return;
444             }
445             eval = evaluate(screen);
446             clear(false);
447             lcd.noBlink();
448             if (evalStatus) {
449                 SUCCESS_TUNE();
450                 lcd.print("=");
451                 lcd.print(eval);
452                 Serial.print("EVAL: ");
453                 Serial.println(eval);
454             } else {
455                 FAIL_TUNE();
456                 lcd.print("SYNTAX ERROR!");
457                 Serial.println("EVAL: SYNTAX ERROR!");
458             }
459             waitClear = true;
460             break;
461         case '<':
462             // 커서 왼쪽으로 옮기기
463             if (0 < cursor) {
464                 SUCCESS_TUNE();
```

```
465                     cursor--;
466                     lcd.setCursor(cursor % 16, cursor / 16);
467                 } else {
468                     FAIL_TUNE();
469                 }
470                 break;
            case '>':
                // 커서 오른쪽으로 옮기기
                if (cursor < 32 - 1 && screen[cursor]) {
                    SUCCESS_TUNE();
                    cursor++;
                    lcd.setCursor(cursor % 16, cursor / 16);
                } else {
                    FAIL_TUNE();
                }
                break;
            default:
                // 비기능 키 입력 구현
                if (cursor < 32) {
                    SUCCESS_TUNE();
                    lcd.print(input);
                    if (cursor == 15) {
                        lcd.setCursor(0, 1);
                    }
                    screen[cursor] = input;
                    cursor++;
                } else {
                    FAIL_TUNE();
                }
                break;
        }
        printToSerial();
    }
}
```