embeded system report

Deadline 2020

[201601639 컴퓨터공학부 홍승현] e-mail: cx\_nf@inu.ac.kr Contact : 010 – 4019 - 2514

목차

[중간 과제 1](#_Toc40809451)

[서론 2](#_Toc40809452)

[보고서 설명 2](#_Toc40809453)

[수행단계 3](#_Toc40809454)

[setting 3](#_Toc40809455)

[INPUT OUTPUT 설정 3](#_Toc40809456)

[Input PIN 설정 3](#_Toc40809457)

[Output PIN 설정 3](#_Toc40809458)

[BREADBOARD 4](#_Toc40809459)

[구현 방법 5](#_Toc40809460)

[초기 실행단계 6](#_Toc40809461)

[가감산기 동작 과정 6](#_Toc40809462)

[예외처리 13](#_Toc40809463)

[추가 구현 14](#_Toc40809464)

[디버그 모드 14](#_Toc40809465)

# 중간 과제

CLCD를 이용한 가감산기 프로그램을 구현

* 입력키 중 ‘+’와 ‘-‘는 CLCD에 내제되어있는 버튼으로 한다.
  + ‘-‘연산자가 최초에 입력되면 minus 기호로 인식한다.
* 버튼을 통해 입력할 때마다 CLCD에 입력되어야 하며, 등호 연산버튼을 누를 경우 계산된 결과가 출력되어야 한다.
* 첫째 줄부터 출력하되, 공간이 부족하면 다음 줄에서 출력한다.
* 결과가 출력된 후 새로운 숫자를 누르면 화면이 지워지고, 다시 눌러진 숫자부터 CLCD에 출력되게 한다.
* 입력, 출력 중 CLCD의 화면이 가득차게 될 경우 “Overflow” 문구, 올바르지 않는 수식일 경우 “Invalid operation” 문구를 2초간 출력 후 화면을 clear한다.
* wiringPi에서 lcd library를 사용하지 말아야 한다.

# 서론

## 보고서 설명

가감산기 구현설명을 목적으로 둔 보고서이다. 가감산기 세팅, 구현방법순으로 열거하여 작성되었으며, 후에lab에 해당하는 모든 요구조건들을 세세히 기술하여 설명한다.

또한, 구현방식이 어떻게 되어있는지를 보이기 위해 코드를 최소화하고, 순서도를 중점적으로 사용하여 이해도를 높였다.

# 수행단계

## setting

가감산기를 사용하기 위한 세팅방법에 대한 설명

### INPUT OUTPUT 설정

가감산기 제작에 필요한 CLCD INPUT과 OUTPUT을 각각 밑 설명과 같이 세팅한다. 그 중 OUTPUT에 해당하는 Target은 점프선을 통해 CLCD와 연결한다.

### Input PIN 설정

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BTN\_PLUS | BTN\_MINUS | BTN\_EQUAL | BTN0 | BTN1 | BTN2 | BTN3 | BTN4 | BTN5 | BTN6 | BTN7 | BTN8 | BTN9 |
| #102 | #103 | #97 | #98 | #100 | #108 | #99 | SCLK | #101 | #118 | MOSI | MISO | CEO |

#### WiringPi for Odroid C1 에서의 GPIO 값

#102 = 5  
#103 = 6  
#97 = 24  
#98 = 27  
#100 = 22  
#108 = 23  
#99 = 26  
SCLK = 14  
#101 = 21  
#118 = 11  
MOSI = 12  
MISO = 13  
CEO = 10

### Output PIN 설정

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| LCD\_RS | LCD\_E | LCD\_D4 | LCD\_D5 | LCD\_D6 | LCD\_D7 |
| #83 | #88 | #116 | #115 | #87 | #104 |

#### WiringPi for Odroid C1 에서의 GPIO 값

#83 = 7  
#88 = 0  
#116 = 2  
#115 = 3  
#87 = 1  
#104 = 4

### BREADBOARD

빵판타겟 중 가장 윗칸의 3V3, 5.0를 기본적으로 CLCD에 점프선으로 끼워 놓아야 하고, 그라운드에 연결해야하는 부분이 여러 곳이 있는데 그 중 필자는 파란색 원에 해당하는 곳에 끼워놓았다(Figure 1-1 참조).

|  |
| --- |
|  |
| 빵판 연결 부분<Figure 1-1> |

|  |
| --- |
|  |
| 빵판 연결 형태<Figure 1-2> |

## 구현 방법

가감산기를 CLCD에 출력하기 위해서는 출력할 문자열을 반복문을 통해 문자들을 일일히 출력해야하지만, 계산을 하기위해서는 정수형 형태로 계산을 하여야 제대로 된 계산결과가 나타난다.

입력된 값을 계산 후 출력하는 형태로 하기 위해서는 문자열로 입력된 값을 정수형으로 변환하여 계산하여야 하고, 계산된 값을 CLCD에 출력하려면 문자열 형태로 바꾸어 CLCD에 출력하는 방식으로 설계하였다.

### 초기 실행단계

|  |
| --- |
|  |
| 초기실행 과정<Figure 2-1> |

* 초기에 파일실행할 때를 입력으로 본다.
* 디버그 모드에 관련해서는 후에 살펴보도록 한다.
* 각 pinMode에서의 input과 Output을 wiringPi 번호에 맞게 세팅해준다.
* 그 후 가감산기가 동작되게 된다.

부가적으로, Input과 Output Setting에 관련해서는 Lab에서의 Hint와 CLCD 세팅에서 그대로 추출하여 작성하였다.

입력 부분에서 매개변수는 총 하나만 받을 수 있는데(실행할 파일의 이름은 생각하지 않는다)  
“./실행될 파일 이름”, 또는 “./실행될 파일 이름 1”을 통해 유저모드나 디버그 모드로 가감산기를 실행할 수 있다. 이는 후에 디버그 모드와 함께 기술하도록 하겠다.

### 가감산기 동작 과정

* 가감산기 동작 과정은 거시적으로 보자면 입력->계산->출력의 한 형태로 볼 수 있다.
* 입력은 문자열 형태의 한 수식으로(등호까지 포함) 받아오며, 계산은 수식을 정수형 형태로 변환하여 각각의 연산자와 피연산자, 등호를 구분하여 계산하고, 출력은 계산된 정수 값을 문자열 형태로 바꾸어 보내는 과정으로 자세한 설명은 다음 그림에 있다.

#### 가감산기 입력처리 과정

|  |
| --- |
|  |
| 입력처리과정1<Figure 2-2> |

* 초기에 유저로부터 입력을 받아야 처리할 수 있기 때문에 계속해서 입력을 기다리는 형태로 되어있다.
* 이를 처리하는 과정은 소스코드 중 Input이라는 함수를 통해 구현되었는데(Figure 2-3 참조) flag에 값을 저장하는 형식으로 되어있다. Flag에 대해 좀더 자세히 설명하면 다음과 같다.

|  |
| --- |
| ------------------------------------------------------------------------------------------------------    ------------------------------------------------------------------------------------------------------ |
| 입력처리과정 소스코드1(상)  Input함수 도식화(중)  Flag를 이용한 입력과정(하)<Figure 2-3> |

* + flag비트는 int형으로 선언되어 있으며 0으로 초기화 되어있다. 그 중 하위 13비트를 사용하게 된다.
  + 그림을 보면 각각의 비트 위치가 무엇을 나타내는지 표기해 두었다. 하위 비트 0에 해당하는 부분은 9를 입력 받을 때 사용하는 부분이라고 생각하면 되며, 13가지 중 하나의 입력이 들어올 경우 해당되는 비트를 1로 변환시키어 저장하게 된다. 이 경우 입력 값을 받았으므로 반복문을 탈출할 수 있게 되는 구조이다.
* 입력을 받았을 때 연산자, 등호, 피연산자에 따라 처리가 달라지도록 설계된다.
  + 피연산자의 경우, 연속적으로 값을 할당해줄 수 있는 반면, 연산자의 경우, 연속된 연산자를 작성하는 것은 Invalid한 수식이기 때문에 이를 처리해줄 필요가 있다.
  + 등호의 경우, 모든 수식에 대한 입력을 마치고 계산을 하겠다는 표시이기 때문에 계속되는 유저로부터의 입력을 마치고 계산을 처리하는 과정으로 넘어가게 해주어야 한다. 물론 이 과정에서도 여러 조건들을 확인하여 처리해야 한다. 등호의 입력을 받기 전 연산자가 입력이 되어있는 상태라면 수식이 성립되지 않으므로 예외처리를 해주어야 한다.
* 피연산자와 연산자간 입력을 유연하게 설계한다.
  + 연속된 연산자를 막기위해 피연산자가 입력되었을 경우 연속된 연산자인지 처리해주는 변수를 미리 선언해두어 값을 조절한다. 이를 통해 연산자가 연속적으로 입력되는지를 파악할 수 있어 원활한 동작이 가능하다. 또한 처음으로 입력된 연산자 ‘-‘에 경우 빼기 연산자가 아닌 앞으로 오게 될 피연산자의 음수 표기로 설정 해두어야한다.
* CLCD에서 나타내는 공간을 제어해야 한다.
  + 입력받은 것을 그대로 CLCD에 출력할 때 화면이 넘어서는 오버플로우 문제가 발생할 수 있다(물론 오버플로우가 직접적으로 발생하는 것은 아니지만 Lab에서 설정한 것을 기준으로 생각한다). 이를 제어하기위해 첫 번째 줄에 최대 16개의 문자가 입력되는데 이럴 경우 두 번째 줄로 커서를 옮기는 식으로 해결할 수 있다. 허나 두 번째 줄까지 꽉 찬 경우에는 오버플로우가 발생하게 된 것이므로 에러 처리를 하여야 한다. 에러처리와 관련된 것에 대해서는 후에 순서도로 표현하도록 하겠다.

#### 가감산기 계산처리 과정

|  |
| --- |
|  |
| 계산처리과정2(상), 최초 연산자 입력 조건문(하)<Figure 2-4> |

* 유저로부터 수식을 입력 받았으면, 이를 정수 형태로 변환하여 계산하여야 한다.
  + 혹시 모를 접근에 대비하여 수식이 존재하지 않을 때 종료를 시키게 두었다.
* 문자열 형태로 되어있는 수식을 하나의 문자형태로 처음부터 끝까지 하나씩 접근하여 처리한다.
  + 피연산자의 정보를 알아내기 위해서 start라는 char\*형 변수를 두어 초기에 수식의 첫 번째 주소를 넣었다. 그리고 각각의 요소에 접근할 때 피연산자의 경우는 처리를 따로 두지 않았다(피연산자의 위치가 어디까지 인지를 파악해야 하기 때문).
  + 연산자를 만났을 경우 뺄셈인지 덧셈인지 확인하는 변수에 상응하는 값을 넣어준 후 계산한 다음, start의 주소 위치를 다음 피연산자로 지정한다(연산자 다음에는 무조건적으로 피연산자가 들어가 있다).
    - * 만약 연산자가 처음에 나오는 경우 계산과정을 무시하고 반복문을 돈다(이는 계산처리과정 순서도에 반영되어 있지 않아 코드로 대체하였다).
  + 등호일 경우 가감연산 계산만 실행한다(빨간색 괄호를 표시한 이유이기도 하다).
* 가감연산 계산형태는 아래와 같다.

|  |
| --- |
|  |
| 가감산기 동작과정(계산 형태)(상), 가감산기 동작과정 코드(하)<Figure 2-5> |

* + 수식을 계산하여 저장할 변수를 sum으로 지정하였다.
  + 현재 피연산자의 시작 위치(start)와 음수계산인지 양수계산인지(plusOrMinus), 계산된 결과를 저장할 변수(sum)을 매개변수로 가져온다.
  + strtol이라는 라이브러리 내 함수를 사용하여 start부터 정수로 인식되는 곳까지를 정수형으로 바꾸어 sum에다가 더하거나 뺀다.
  + 그 후 결과값인 sum을 리턴하여 저장한다.
* 좀 더 자세한 설명을 위해 주소가 어떻게 접근되는지 간단하게 정리하였다(아래 그림 참조).

|  |
| --- |
|  |
| 가감산기 동작과정(계산 형태)<Figure 2-6> |

1. 각 수식들의 요소들을 하나씩 접근하여 처리한다(Figure 2-4를 참조).
2. 연산자를 만났을 경우 strtol 함수를 통해 start 위치부터 정수로 인식되는 곳까지(‘+’부분까지, 즉 현재 부분까지)를 설정하여 정수형태로 반환된다.
3. 계산된 후 start의 위치를 해당 인덱스의 다음 주소로 설정해둔다.
4. 예외로 등호의 경우에는 2 까지만 실행이 되고 그 후로 start의 위치를 지정하지 않고 넘어가는 식으로 구현되었다.

#### 가감산기 출력처리 과정

지금까지 입력과 계산처리 부분 까지를 알아보았다.

CLCD 출력에 대해서는 이렇다할 자료구조들이 필요로 하지 않기 때문에 간단하게 구현되는 것을 아래 그림을 통해 확인할 수 있다.

|  |
| --- |
|  |
| 출력처리과정<Figure 2-7> |

* 지금까지 입력을 위해 사용되었던 수식문자열을 출력으로 재사용하기 위해서 null로 초기화를 해준다.
* 정수형을 문자열로 저장시키기 위해 sprintf 함수를 사용하여 저장한다(해당 메소드 설명은 넣지 않음).
* 저장된 문자열 각각의 요소를 접근하여 CLCD에 출력해준다.
  + 만약 CLCD 출력 중 첫 번째 줄을 출력하고 있을 때 칸이 모자라는 경우 두 번째 줄로 이동시키는 커맨드를 입력한다(0xC0, 이는 예제 형식에 나와있으므로 설명 생략).
  + 두 번째 줄에서 출력 중일 때 칸 수가 모자라는 경우 오버플로우가 발생된 형태이므로 예외처리를 해주어야 한다.
* 모든 값이 정상적으로 출력이 되면 다시 가감산기가 동작하여 Figure 2-2로 돌아간다.

### 예외처리

해당 과제 내에서 구현해야하는 에러는 Invalid와 Overflow 두 가지이다.

이는 가감산기 처리 과정 내에서 예외처리를 해야할 때가 발생하는데, 이 때 매개변수 하나가 주어지게 된다. 매개변수를 통해 이 처리값이 Overflow인지 Invalid인지를 구분하여 처리해줄 수 있다.

|  |
| --- |
|  |
| 에러 처리<Figure 2-8> |

## 추가 구현

### 디버그 모드

추가적으로 어떻게 동작하는지 알아보기위해 디버그모드를 추가하였다.

|  |
| --- |
|  |
| 디버그 모드(좌)와 유저 모드(우)<Figure 2-1> |

해당 파일을 실행할 때 두 번째 매개변수에 1이라는 값을 입력해주면 디버그모드로 바뀌어지므로 가감산기가 도중에 어떻게 실행되는지는 이를 통해 확인해 볼 수 있다. 유저모드(일반)일 경우에는 초기화, 에러를 제외한 나머지는 결과가 출력되지 않는다.

랩에 해당되지는 않지만 사용자의 편의성을 위해 구현되었다.