

# R/ E/ P/ O/ R/ T

제출일: 2019-04-14

과제명: 공학용 계산기 #1

과목 : 휴먼컴퓨터인터페이스

학과 : 소프트웨어학부

학번 : 2015726076

이름 : 김현구

# 개요

## 구현 완성도 요약표

### 1. 기능적 요구조건

구분	기능	구현 여부
최소 기능적 조건	<b>1. 수식 입력</b>	○
	1. 정수, 실수, 복소수의 표현과 그 기본 연산	
	산술연산 (+, -, *, /, %, ^)	○
	비교연산 (==, !=, >, <, >=, <=)	○
	2. 벡터, 행렬의 표현과 그 기본연산	
	벡터: 내적(n 차원), 외적(3 차원)	○
	행렬: 곱셈, 역행렬(inverse), 행렬식(determinant)	○
	3. 자주 사용되는 상수 및 함수 지원	
	상수: pi, e, i	○
	함수: sin, cos, tan, exp, log, sqrt	○
	<b>2. 결과 출력</b>	○
	올바른 입력 → 수식의 결과 값	○
	잘못된 입력 → 오류 메시지	○
	<b>3. 변수, 함수 정의 및 사용</b>	○
	변수: 최소 3 개 (예. x, y, z)	○
	함수: 최소 2 개 (예. f, g)	○

## 2. 인터페이스 요구조건

구분	기능	구현 방법	구현 여부
기본 인터페이스	팝업 메뉴	행렬, 벡터에 관한 함수와 사용자 정의 함수, 변수에 대해 팝업 메뉴로 그룹화	○
	백 스페이스	숫자, 연산자 등 하나의 문자로 된 것 뿐 아니라 함수처럼 여러 문자로 되어있을 경우 그 함수 전체를 지움	○
	도움말	? 모양의 녹색 버튼을 누를 경우 팝업을 이용해서 전체적인 도움말을 출력	○
고급 인터페이스	벡터, 행렬 입력 간소화	벡터함수나 행렬에 관한 함수, 혹은 default 버튼으로 행렬 괄호쌍을 사용자가 버튼으로 원하는 크기로 조절 후, 결과창에 출력 가능	○
	복사, 붙여넣기	드래그 후 copy 을 누르면 드래그 한 부분이, 드래그 하지 않은 상태에서 누르면 문자열 전체가 복사되어 붙여넣기 가능	○
	히스토리	Select 태그에 기록되는 과거의 계산 식들을 더블 클릭으로 다시 사용가능	○

## 3. 오픈소스 라이브러리 의존성 요약

부트스트랩 프레임 워크 사용 – CSS, JS

bootstrap.css, bootstrap.js

Jquery 사용 – JS

<https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js>

Math.js 사용 – JS

Math.js

# 본문

## 사용자 인터페이스의 구성 요소 및 사용 방법



# 특징적인 상호작용 방식들에 대한 세부 구현 방법

## 행렬, 벡터 괄호 쌍 입력 기능

### 1) Toggle open

```
$(document).on("click",".procession",function(k) //토글 창 on,off , 생성된 html 팝업요소의 접근을 위한 on()사용
{
    // slice(-3,-1)은 마지막에서 세번째, 두번째 문자로 cross와 dot의 중복 클릭을 했을 때 입력을 막기 위한
    if(($$(this).val() == 'DO' || $(this).val() == 'CR' || $(this).val() == 'DE' || $(this).val() == 'IN' || $(this).val() == 'DF')
    && (displayValue.slice(-3,-1) != "do" || displayValue.slice(-3,-1) != "os" || displayValue.slice(-3,-1) != "de"
    || displayValue.slice(-3,-1) != "in" || displayValue.slice(-3,-1) != "ul"))
    {
        displayValue += $(this).text();
    }

    if($("#plustab").is(':visible'))
    {
        $('#plustab').hide();
    }
    else
    {
        $('#plustab').show();
        $(".row-"+rows+"-col-"+cols).focus(); // 정해진 칸에 포커스
    }

    k.stopImmediatePropagation();
})
```

#### 1-1) 설명

행렬을 표현하는 3 버튼 (default, det, inv) 와

벡터를 표현하는 2 버튼 (dot, cross) 를 누르면 procession 클래스를 가졌기 때문에 id = plustab 에 해당하는 추가 탭이 토글이 된다.

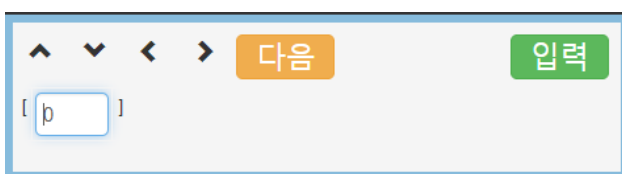
그리고 행렬식을 다 받고 해당 함수의 이름을 같이 출력해 주기 위해서 함수가 아닌 default 버튼을 제외하고 다른 값을 텍스트에 저장한다.

show 함수로 탭을 열었을 때는 행렬 첫번째 칸에 접근하는 focus 함수를 사용했다.

이후 stopImmediatePropagation() 함수로 같은 계층의 다른 이벤트 버블링을 막는다.

#### 1-2) 이미지

웹페이지를 띄우고 처음 plustab 을 열었을 때 모습



## 2) 크기 조절

```
if($(this).val() == 'DN' && mxrows < 5) // 아래로 늘리기
{
    mxrows += 1;
    $("#myTable").append("<tr id='row-" + mxrows + "'></tr>");
    $("#row-" + mxrows).append("<td>[</td>");
    $("#row-" + mxrows).append("<td><input type='text' class='form-control key vectorInsert row-" + mxrows + " col-1' value='0'></td>");
    for(i = 2; i <= mxcols; i++)
    {
        $("#row-" + mxrows).append("<td>,</td>");
        $("#row-" + mxrows).append("<td><input type='text' class='form-control key vectorInsert row-" + mxrows + " col-" + i + "' value='0'></td>");
    }
    $("#row-" + mxrows).append("<td class='lastcol'>]</td>");
}
else if($(this).val() == 'RI' && mxcols < 5) // 오른쪽으로 늘리기
{
    $(".lastcol").remove();
    mxcols += 1;
    for(i = 1; i <= mxrows; i++)
    {
        $("#row-" + i).append("<td>,</td>");
        $("#row-" + i).append("<td><input type='text' class='form-control key vectorInsert row-" + i + " col-" + mxcols + "' value='0'></td>");
        $("#row-" + i).append("<td class='lastcol'>]</td>");
    }
}
else if($(this).val() == "LE" && mxcols > 1)
{
    for(i = 1; i <= mxrows; i++)
    {
        $("#row-" + i).children('td').slice(-3,-1).remove();
    }
    mxcols -= 1;
}
else if($(this).val() == "UP" && mxrows > 1)
{
    $("#row-" + mxrows).remove();
    mxrows -= 1;
}
```

### 2-1) 설명

화살표 모양의 버튼으로 행렬 괄호 쌍을 조절할 수 있음

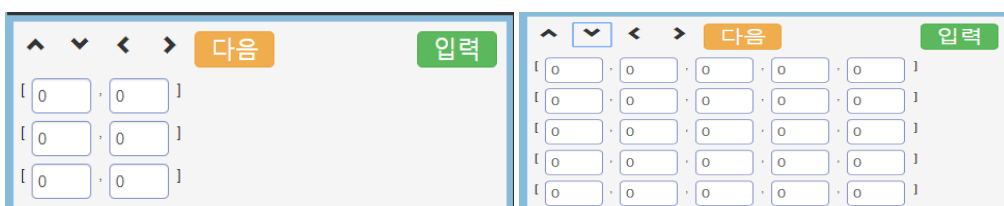
DN = 아래, RI = 오른쪽, LE = 왼쪽, UP = 위쪽 으로 조절한다.

아래와 위 방향은 mxrows 라는 최대 괄호 쌍 크기를 조절 후에 열을 하나 추가하거나 삭제한다.

왼쪽과 오른쪽은 mxcols 변수와 class='lastcol'이 붙어있는 닫는 대괄호를 이용해서 오른쪽으로 늘릴 때는 ']' 를 하나 지우고 ',' 와 input 태그를 넣고 다시 ']'를 추가한다. 거꾸로 왼쪽으로 지울 때는 tr 태그의 자식 태그중 td 를 slice(-3,-1)을 이용해 input 태그 하나와 ',' 하나를 지운다

최대 크기는 5\*5 로 그 이상 늘릴 수 없다.

### 2-2) 이미지



### 3) 값 접근하기

```
if($(this).hasClass("pc")) // 숫자+변수+기본함수 버튼을 눌렀을 때
{
    displayProcess += $(this).text()
    $(".row-"+rows+" .col-"+cols).val(displayProcess);
}
```

```
if($(this).hasClass("arrow")) // 행렬 칸 내에서의 지우기
{
    displayProcess = remove(displayProcess);
    $(".row-"+rows+" .col-"+cols).val(displayProcess);
}
```

```
function remove(display)
{
    let context = display.slice(-3,length.display);
    // 여러 함수에 대한 한번에 지우기 처리
    if(context == "sin" || context == "cos" || context == "tan" || context == "exp" || context == "log"
    || context == "qrt" || context == "dot" || context == "oss" || context == "inv" || context == "det")
    {
        if(context == "qrt")
        {
            display = display.slice(0,-4);
        }
        else if(context == "oss")
        {
            display = display.slice(0,-5);
        }
        else
            display = display.slice(0,-3);
    }
    else // 일반 숫자나 한자리의 연산에 대한 지우기 처리
    {
        display = display.slice(0,-1);
    }

    return display;
}
```

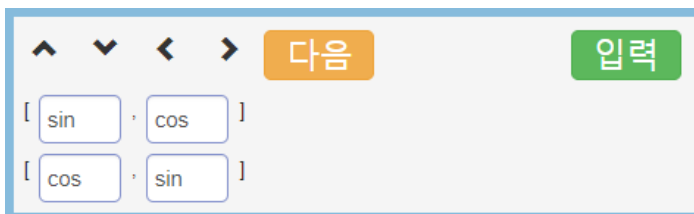
#### 3-1) 설명

처음 이미지는 pc 클래스를 통해서 숫자, 삼각함수 등 원래 사용 가능한 모든 값을 받을 수 있다.

두번째 이미지는 백스페이스 기능을 함수로 저장하고 행렬 칸 안에서도 실행 가능하게 만들었다.

백스페이스 기능은 현재 표시된 display 값을 slice 로 접근해서 함수의 경우 이름을 한번에 지울 수 있도록 만들었다.

#### 3-2) 이미지



#### 4) 칸 이동

```
let mxrows = 1;
let mxcols = 1;
let rows = 1;
let cols = 1;
```

```
if($(this).hasClass("next")) // 칸 이동
{
    displayProcess = '';

    if(cols < mxcols) // 행 이동
    {
        cols += 1;
    }
    else if(rows < mxrows) //같은 행의 숫자가 다 찾으면 한 열 아래로
    {
        rows += 1;
        cols = 1;
    }
    else // 모든 칸이 다 찾으면 1열 1칸으로 이동
    {
        cols = 1
        rows = 1
    }
}
```

##### 4-1) 설명

열과 행의 최댓값을 저장한 mxrows, mxcols 변수와 현재 focus 로 가리키는 rows, cols 변수를 통해서 다음 버튼을 눌렀을 때 next 클래스를 확인하고 기능을 수행한다.



## 5) 값 입력하기

```
else if($(this).val() == "EN") //완성된 행렬 출력하기
{
    let state = '';
    console.log(displayValue);
    if(displayValue.slice(-3,-1) == "de" || displayValue.slice(-3,-1) == "in") // 행렬을 입력할 때 괄호쌍 설정을 위함
        state = "procession";
    if(displayValue.slice(-3,-1) == "ul") // default라는 텍스트를 제거해줌
    {
        state = "default";
        displayValue = displayValue.slice(0,-7);
    }

    if(state != "default") // 기본행렬일 경우 '(' 삭제
        displayValue += "(";
    if(state == "procession" || state == "default") // 행렬에 관한 함수일 경우 '[' 추가
        displayValue += "[";
    for(i = 1; i <= mxrows; i++)
    {
        displayValue += "[";
        displayValue += $(".row-"+i+" .col-1").val();
        for(j = 2; j <= mxcols; j++)
        {
            displayValue += ",";
            displayValue += $(".row-"+i+" .col-"+j).val();
        }
        displayValue += "];";
        displayValue += ",";
    }
    displayValue = displayValue.slice(0,-1); //하나 지우기(,)

    if(state == "procession" || state == "default") // 행렬에 관한 함수일 경우 ']' 추가
        displayValue += "];";
    if(state != "default")
        displayValue += ")";

    $('#result').text(displayValue);
    $('#plustab').hide();
}
```

### 5-1) 설명

Default 는  $[[x,x],[x,x]]$  처럼 괄호쌍을

det, inv 는 행렬함수로  $\text{func}([[x,x],[x,x]])$  2 중 대괄호 쌍을

cross, dot 은 벡터함수로  $\text{func}([x,x],[x,x])$  1 중 대괄호 쌍을 설정하기 위해서

조건문으로 따로 바깥의 괄호쌍을 출력해주고, 안에 내용은 반복문을 통해서  
rows- cols- 클래스 값으로 input 태그에 접근해서 값을 출력한다.

### 5-2) 이미지



## 실제 문제에 대한 사용 예시

1) 행렬식  $\begin{bmatrix} -2 & 2 & 3 \\ -1 & 1 & 3 \\ 2 & 0 & -1 \end{bmatrix}$  을 풀어보자



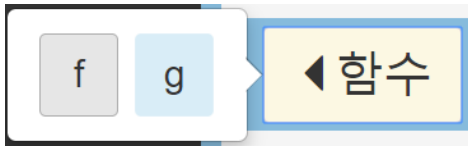
행렬식을 구하기 위해 det 함수를 누른다

값을 입력하고 입력버튼을 누른다.

결과창에 완벽히 출력된 것을 확인하고 EV 버튼으로 계산한다.

계산이 완료되고 히스토리에 수식이 추가가 된다.

2) 함수를 이용해서  $n$  을 입력하면  $n*(n+1)$ 을 출력한다.



함수 버튼을 누르고 f 함수를 누른다.

**$f(x)=x*(x+1)$**

변수  $x$  를 이용해서 수식을 만든다.

**function**

$f(x)=x*(x+1)$   
 $\det([[-2,2,3],[-1,1,3],[2,0,-1]])$

function 이라는 문구가 뜨며 함수가 정의된 것을 확인할 수 있다.

**$f(5)$**

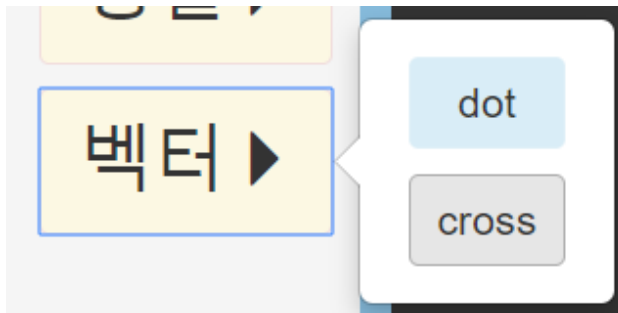
함수의 인자로 5 를 넣어보고 EV 버튼을 누른다.

**30**

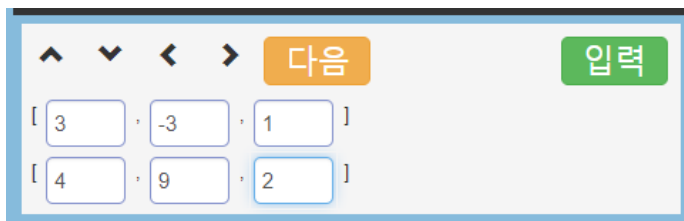
$f(5)$   
 $f(x)=x*(x+1)$   
 $\det([[-2,2,3],[-1,1,3],[2,0,-1]])$

결과가 정확히 출력된다.

3) 벡터의 외적  $\text{cross}([3, -3, 1], [4, 9, 2])$ 을 구한다.



벡터 버튼의 cross 를 누른다.



괄호 쌍의 크기를 맞추고 값을 입력한다.

```
cross([3,-3,1],[4,9,2])
```

결과를 확인하고 계산한다.

```
[-15, -2, 39]
```

답이 맞았음을 알 수있다.

# 논의

## 구현 측면에서 성공적인 부분

### 1. 행렬의 괄호 쌍 입력부분

이 것을 구현하면서 이벤트 버블링에 관한 문제를 제대로 알지 못해서 한번 코드를 갈아엎기도 하고 여러 우여곡절이 많았다. 결과적으로 직관적이고 좋은 UI로 괄호를 처리 할 수 있는 방법을 생각해 냈고, 잘 구현한 것 같다.

### 2. 히스토리, 복사 붙여넣기 부분

더블 클릭을 이용해서 히스토리를 사용하고, 그 안에서 복사 붙여넣기를 사용하면 마우스만을 사용하는 계산기 임에도 편의성이 상당히 올라가는 것을 느낄 수 있었다.

### 3. 백 스페이스 부분

함수 명을 인식해서 한번에 지우는 기능을 넣음 으로서, 수정된 수식에 대한 접근이 편리해졌다.

# 사용성 측면에서 긍정적인 측면과 부정적인 측면

## ㄱ.긍정적인 측면

### 1. 행렬에 대한 입력

괄호쌍을 편리하게 입력하면서 행렬과 벡터에 대한 사용이 편리해졌다.

### 2. 버튼

EV 를 제외한 모든 버튼을 한글로 만들어 직관적으로 이해할 수 있게 했고,  
버튼의 크기 또한 크고 누르기 편하게 만들었다.

## ㄴ.부정적인 측면

### 1. 다양한 함수의 부재

역삼각함수나, 피보나치 등 많은 함수가 있겠지만 기본적이고 자주 사용하는  
연산에 집중하기 위해 제외를 했다.

# 전반적인 평가와 향후 개선 계획

## ㄱ. 전반적인 평가

다양한 기능을 지원하기보다 자주 사용하는 기능을 더 편하게 하기위해서 일부 기능들이 다른 계산기에 부족할 수는 있으나 직관적이고 사용하기 편한 UI 로 잘 구현하는데 성공했다고 생각한다.

## ㄴ. 향후 개선 계획

여러 가지 아이디어가 많았는데, 나중에 추가했으면 좋겠는 기능을 모아두었다.

### 1. 팝업을 통해 계산 수식 구체화

제곱이나 log 와 같은 아래 첨자나 윗 첨자 등 일반적인 string 타입에서 볼 수 없는 식을 표시하는 기능

### 2. 식 계산 후 나온 답을 다음 식의 처음 값으로 사용

계산을 한번 완료하고 결과창에 뜬 결과값을 연산자를 바로 누르면 사용할 수 있게 하는 기능

### 3. 커서의 위치를 인식

커서의 위치를 인식해서 커서가 있는 지점에 식이 들어가게 하는 기능