

보고서

- 공학용 계산기 -

과목명 | 휴먼컴퓨터인터페이스

담당교수 | 이강훈

제출일 | 2019년 4월 14일

소속 | 소프트웨어학부

학번 | 2015202065

이름 | 윤홍찬



< 목 차 >

I . 요구조건에 대한 구현 완성도 요약	1
II . 사용자 인터페이스 구성 요소 및 사용 방법	2
III. 특징적인 상호작용 방식들에 대한 세부 구현 방법	9
IV. 실제 문제에 대한 사용 예시	17
V. 구현 측면에서 성공적인 부분과 실패한 부분	22
VI . 사용성 측면에서 긍정적인 측면과 부정적인 측면	23
VII . 과제 결과에 대한 전반적인 자체 평가 및 향후 개선 계획	24

1 - 1. 기능적 요구조건에 대한 구현 완성도 요약

분류	기능	구현
요구 조건	정수, 실수, 복소수의 표현과 그 기본 연산 - 산술연산(+, -, *, /, %, ^), 비교연산(==, !=, >, <, >=, <=)	○
	벡터, 행렬의 표현과 그 기본 연산 - 벡터: 내적(n 차원), 외적(3 차원) / 행렬: 곱셈, 역행렬, 행렬식	○
	자주 사용되는 상수 및 함수 지원 - 상수: pi, e / 함수: sin, cos, tan, exp, log, sqrt	○
	결과 출력 - 올바른 입력 -> 수식의 결과 값 / 잘못된 입력 -> 오류 메시지	○
	변수, 함수 정의 및 사용 - 변수 : x, y, z / 함수 : f(), g()	○
추가 기능	자주 사용되는 연산 추가 - x^y , ln 연산 등 추가 구현	○

1 - 2. 인터페이스 요구조건에 대한 구현 완성도 요약

분류	기능	구현
기본	팝업 메뉴 - 기능의 그룹화, 단계별 선택지의 최소화	○
	백 스페이스 - 입력 수식의 마지막 문자 지우기	○
	도움말 - 인터페이스 사용법 설명	○
고급	벡터, 행렬 입력 간소화 - 괄호 쌍 입력의 불편함 최소화	○
	복사 & 붙여넣기 - 일부 영역 선택하여 보관 및 재사용	○
	히스토리 - 과거 입출력 내역의 확인 및 재사용	○
추가	계산기의 현재 상태를 표시 - 사용자가 계산기의 상태를 파악하고 상호작용 함	○
	Next 버튼 구현 - 행렬, 벡터 뿐만 아니라 각종 연산 또한 입력 간편화	○

1 – 3. 오픈소스 라이브러리 의존성 요약

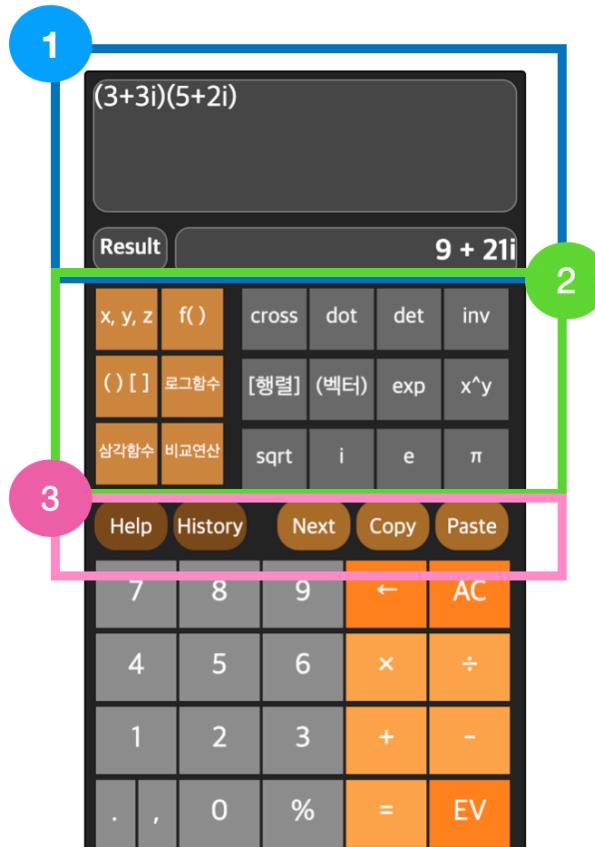
[jQuery popModal 플러그인]

(데모 및 다운로드 링크 : <https://www.jqueryscript.net/lightbox/jQuery-Multi-Purpose-Popup-Modal-Plugin-popModal.html>)
(github 링크 : <https://github.com/vadimsva/popModal>)

popModal 플러그인은 MIT 라이선스로 공개되고 있으며, Tooltips, Title, Modal Dialog 등을 위한 플러그인이다. 이중 팝업 창을 표시하는 **popModal** 과 마우스를 오버하면 **ToolTip** 과 비슷한 형태로 정보를 제공하는 **hintModal** 을 사용하였다.

2 사용자 인터페이스의 구성 요소 및 사용 방법

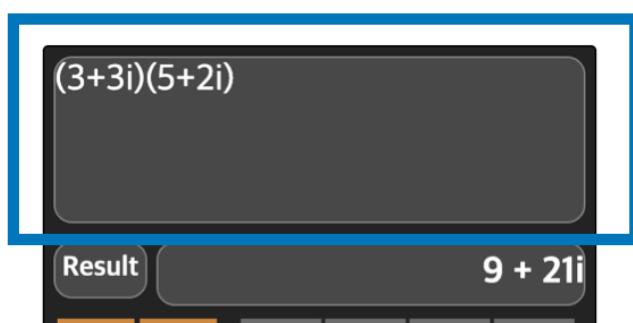
다음 사진은 최종적으로 구현한 계산기를 크게 세 부분으로 나눈 것이다.



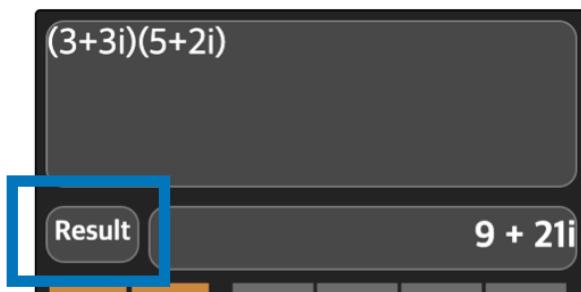
첫 번째, 파란색 네모 부분은 사용자로부터 입력 받은 값을 표시하고, 계산기의 현재 상태와 연산 결과를 출력하는 사용자와 상호작용 하는 부분이다. 두 번째, 초

록색 부분은 각종 연산들이 위치해있으며, 일련의 연관된 연산들은 묶어서 사용자에게 표시하고 있다. 마지막으로 세 번째 부분은, 도움말, 메모리, 팔호 연산에 도움을 주는 버튼, 복사, 붙여 넣기가 위치하여 사용자를 위한 편의를 제공하는 기능을 모아놓았다. 이제 1 번 파란색부터 3 번 분홍색까지 차례대로 살펴보겠다.

1) 사용자와 상호작용



1 번 부분은 다시 세 부분으로 나눌 수 있다. 제일 먼저 가장 큰 비중을 차지하는 이 영역은 사용자로부터 입력 받은 값을 표시한다.



제일 작은 비중을 차지하는 이 영역은 현재 계산기의 상태를 표시한다. 다음은 각 상황에 대한 상태이다.

사용자 입력 : Input

결과 출력 : Result

지우기 : Erase

초기화 : Clear

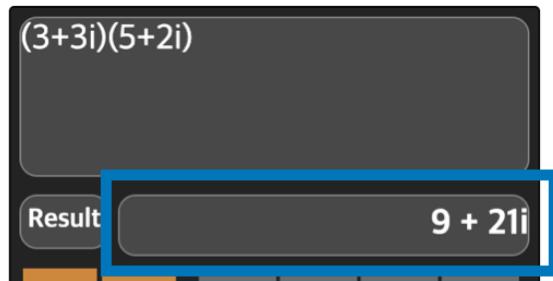
복사 : Copy

붙여 넣기 : Paste

설명 모드 : Help

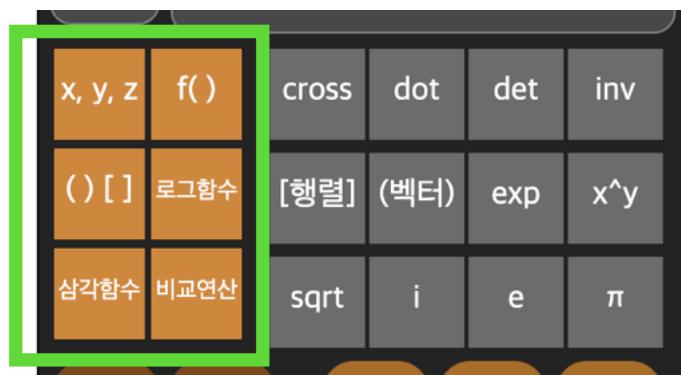
행렬 및 벡터 간편 입력 모드 : M / V

사용자는 계산기를 사용하면서 입력에 따라 어떻게 계산기가 반응하는지 확인하게 되고, 이는 계산기를 더 잘 활용할 수 있을 것이라 기대된다.

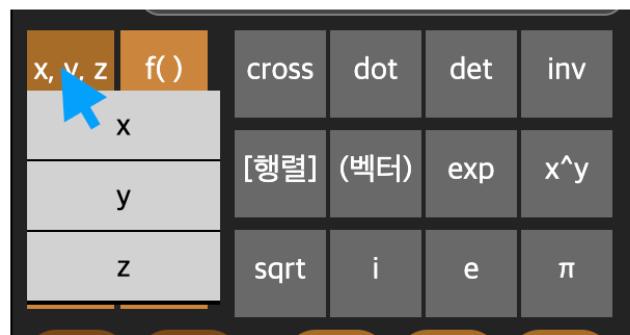


첫 번째 부분 중 마지막인 **수식의 결과가 출력되는 부분**이다. 입력된 수식이 올바른 값이면 결과가 출력되고, 올바르지 않다면 에러를 출력하게 된다.

2) 각종 연산 기능을 제공



2 번 부분은 크게 두 가지 부분으로 나눌 수 있다. 위의 사진은 **일련의 연관된 연산들을 모아 놓은 곳**이다. 기존 공학용 계산기는 복잡한 연산들이 뭉쳐 없이 제공되고 있어 사용자로부터 불편함을 초래하고 있다. 따라서 연관된 연산을 뭉어서 제공함으로서 이를 해결하고자 하였다.



[변수를 모아 놓은 x, y, z 버튼을 누른 예]

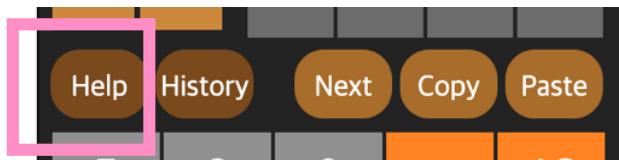
위의 사진과 같이 각 버튼을 누르게 되면 일련의 연산, 기호들이 애니메이션 효과와 함께 드롭박스 형태로 제공된다. 사용자는 이 중 원하는 것을 선택하여 사용하면 된다.



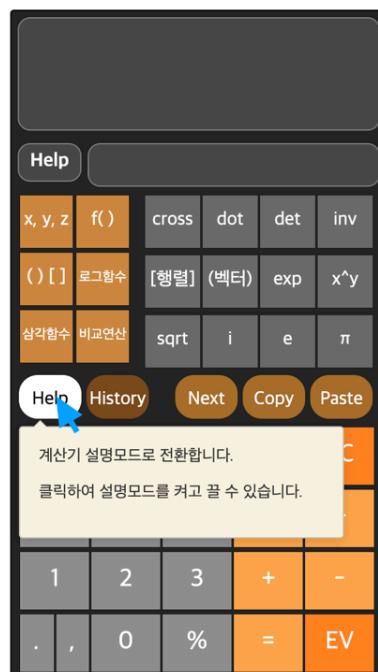
다음 오른쪽에 위치한 것은 자주 사용되는 각종 연산들이다. 사용자는 여기 있는 것을 사용해 제곱근, 지수함수, 거듭제곱, 벡터의 내적, 외적, 행렬식, 역행렬을 구할 수 있다. 여기 있는 연산들은 대부분 괄호, 행렬, 벡터를 사용하는데, 이에 대해 간단하게 입력할 수 있게 구현하였다. 이것은 이후 3 번 영역에서 Next 버튼과 함께 설명하겠다.

3) 사용자 편의를 위한 각종 기능

3 - 1) 도움말 기능



각종 기능 중 첫 번째로 **Help**, 즉 도움말 기능이다. 복잡한 공학용 계산기를 처음 사용하는 사용자는 쓰는 방법을 잘 몰라 혼란스러워 할 것이다. 따라서 Help 버튼을 두어 계산기를 사용하는 방법을 제공하였다. 다음은 이에 대한 예시이다.



위와 같이 Help 버튼에 마우스를 오버하게 되면 위와 같은 툴팁이 제공된다. Help 버튼을 클릭하면 설명 모드로 전환이 되어 각종 연산이나, 버튼, 인터페이스에 마우스를 오버하면 아래 사진과 같이 툴팁 형태로 설명문이 보여진다. 설명 모드 시 Help 버튼의 색과 글자 색이 바뀌어 현재 설명 모드인 것을 나타낸다.



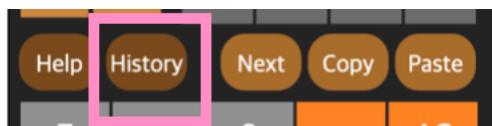
[연산의 경우 사용 예시와 과정 또한 도움말로 제공된다]



[연산 이외의 버튼, 인터페이스 등에도 도움말이 제공된다]

설명 모드를 사용하고 있는 동안에도 기존 계산기의 연산 등 모든 기능을 사용할 수 있어 계산기를 처음 사용하는 사용자들은 계산기가 익숙해질 때까지 설명 모드를 켜고 계산기를 사용할 수 있다. (이후 명시된 YouTube 링크에서 첫 번째 예제로 설명 모드를 켜고 연산을 수행하는 모습을 볼 수 있다.)

3 - 2) 메모리 기능

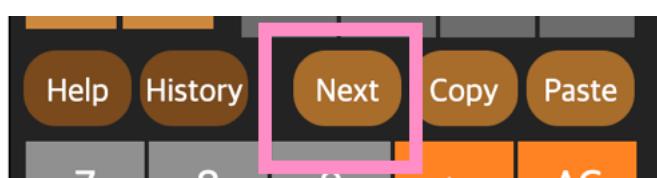


다음으로 살펴볼 기능은 **History**, 메모리 기능이다. 사용자는 이전에 계산을 수행했던 수식의 결과를 확인하고 싶어할 수 있다. 따라서 이에 대한 기능을 제공한다.



History 버튼을 클릭하면 다음과 같이 팝업 창이 하나 생겨, 이전 수식의 결과를 확인할 수 있다.

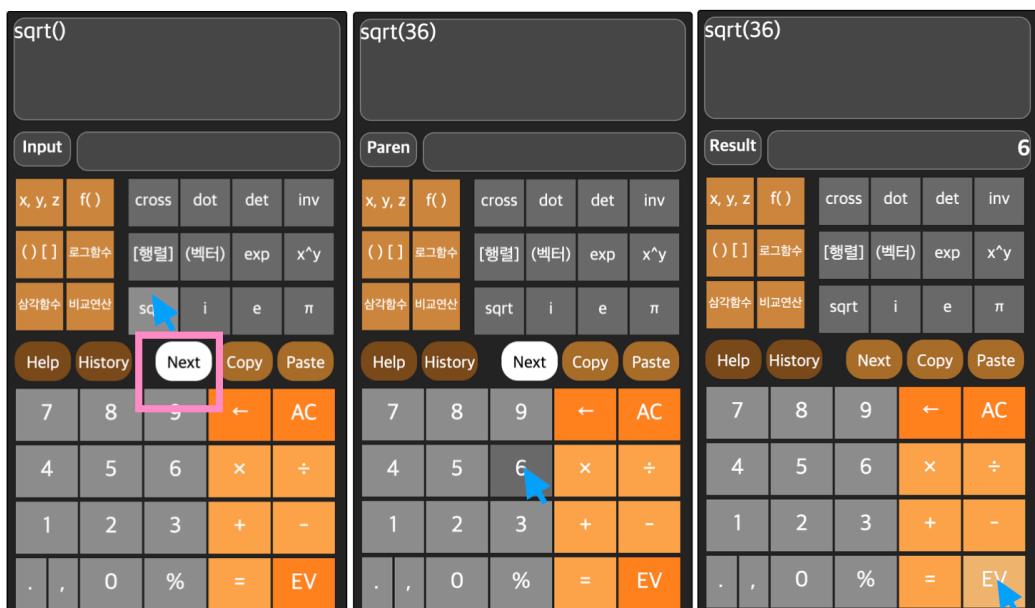
3 - 3) Next 버튼(괄호, 벡터, 행렬 연산의 단순화)



세 번째로 살펴볼 것은 **Next** 버튼이다. 위에서 연산에 대해 언급할 때 이를 사용해서 **괄호, 벡터, 행렬의 연산을 단순화** 할 수 있다고 언급했었다. 어떻게 사용하는지 **sqrt** 연산으로 설명하겠다. (**벡터 및 행렬 입력 간편화는 이후 실제 문제에서 설명하겠다.**)

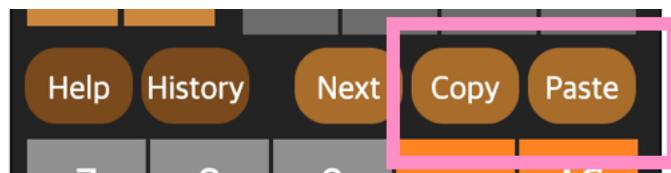


sqrt 연산을 사용하기 위해 위에서 설명한 설명 모드로 확인을 하고 있는 사진이다. 이 연산은 설명을 보듯이 괄호 연산을 필요로 한다. 위의 연산 입력 순서와 똑같이 따라해 보겠다.

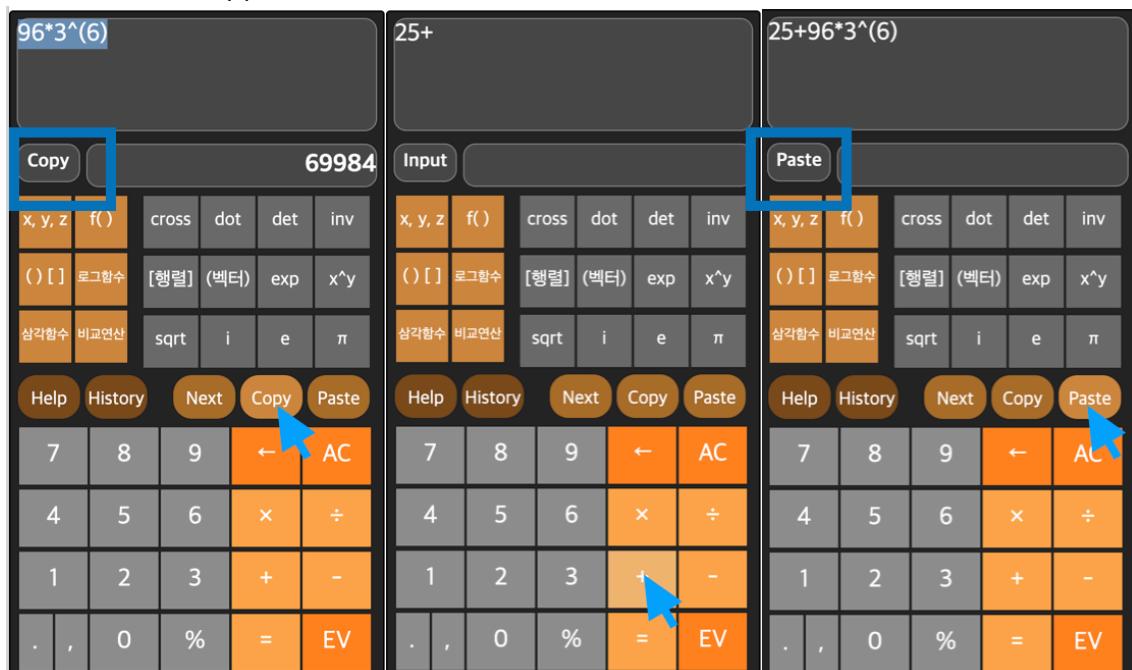


맨 왼쪽 사진을 보면 **sqrt** 연산을 클릭하고 있는데, 밑의 **Next** 버튼이 하얀색으로 활성화 된 것을 확인 할 수 있다. 이를 통해 사용자에게 괄호 간편 입력이 시작되었다고 알리게 된다. 위의 경우는 추가적인 연산이 불 필요하기 때문에 **Next** 버튼을 직접적으로 클릭하지는 않지만, 추가적으로 연산을 입력할 때는 반드시 **EV** 버튼이 아닌 **Next** 버튼을 클릭하여야 한다.

3 - 4) 복사 및 붙여넣기 기능



마지막으로 살펴 볼 사용자 편의 기능은 **복사와 붙여넣기** 기능이다. 사용자는 복사 버튼과 붙여넣기 버튼을 순서대로 클릭하여 원하는 수식을 복사하고 붙여넣을 수 있다. 다음은 예시이다. 이때, 복사된 값은 계산기 내에서 쓸 수 있을 뿐만 아니라 **클립보드에 저장되어 외부에서도 사용할 수 있다**. 추가적으로 붙여넣기 기능은 현재 이어쓰기(Append)기능만 지원하고 있다.



맨 왼쪽 사진부터 보면 수식을 입력하고 Copy를 누르면 수식이 입력되는 부분이 선택되고, 파란색 박스 안의 계산기 상태로 보아 복사가 되었음을 알 수 있다 이후 다시 Paste 버튼을 클릭하면 수식이 붙여넣기(Append) 된다.

3. 특징적인 상호작용 방식들에 대한 세부 구현 방법

1) 팝업 메뉴 – 연관된 일련의 연산들의 모음

```

758 <table>
759   <tr>
760     <td class="selectBox">x, y, z
761       <div class="dropDown">
762         <span class="dropKey">x</span>
763         <span class="dropKey">y</span>
764         <span class="dropKey">z</span>
765       </div>
766     </td>
767     <td class="selectBox">f( )
768       <div class="dropDown">
769         <span class="dropKey">f( )</span>
770         <span class="dropKey">g( )</span>
771       </div>
772     </td>
773   </tr>
774   <tr>

```

- html 코드 -

```

359 /* Install Dropdown Event Handler */
360 $('.selectBox').click(function(){
361   $('#status').text("");
362   var $dropDown = $(this);
363   var toggle = $dropDown.children().hasClass('show');
364   if(toggle == true){
365     $dropDown.children().removeClass('show');
366     $dropDown.removeClass('selected');
367     $dropDown.children().slideUp(400);
368   }else{
369     $dropDown.children().addClass('show');
370     $dropDown.addClass('selected');
371     $dropDown.children().slideDown(400);
372   }
373 });

```

- jQuery 코드

- html 태그에서 처음에 보여지는 요소 밑의 자식 태그를 작성한 후 보이지 않게 속성처리를 함.
- 보여지는 요소를 클릭할 때마다 'show'라는 클래스가 지정되어있지 않으면 지정하고, 지정되어있으면 없앰.
- 따라서, 클릭할 때마다 보여지고 사라지는 것을 반복함

+ 'show' 클래스와 같이 적용하거나 해제하는 'selected' 클래스는 부모의 색상을 바꾸어 선택된 것처럼 효과를 줌

+ toggle 메소드를 사용하지 않은 이유는 slideUp, slideDown 애니메이션 효과 적용을 위함

x, y, z	f()	cross	dot	det	inv
() []	로그함수	[행렬]	(벡터)	exp	x^y
삼각함수	비교연산	sqrt	i	e	π

`$(객체).addClass('show')`

x, y, z	f()	cross	dot	det	inv
x		[행렬]	(벡터)	exp	x^y
y		sqrt	i	e	π

`$(객체).removeClass('show')`

2) 백 스페이스 – 입력 수식의 마지막 문자 지우기

```

/* Backspace */
else if(clickedText == '←'){
  displayValue = displayValue.substring(0, displayValue.length - 1);
  $('#userInput').text(displayValue);
  $('#status').text("Erase");
}

```

입력된 수식에서 substring 메소드를 호출하여 맨 마지막에 입력된 값을 제외한 문자열을 반환 받아 저장하였다. 다음은 이에 대한 화면이다.



3) 도움말 – 인터페이스에 대한 설명 제공

```
<td class="operationKey hintModal">sqrt
    <div class="hintHide thisIsHint">
        표시된 값의 제곱근을 계산합니다.
        <p>
            사용 예시 : sqrt(36)<br><br>
            1. sqrt 클릭<br>
            2. 36 입력<br>
            3. 수식을 더 작성할 경우 Next<br>
                &nbsp;&nbsp;&nbsp;연산할 경우 EV 클릭
        </p>
    </div>
</td>
```

- html 코드 -

```
/* Install Help Event Handler */
$('#help').click(function(){
    /* Original Mode (Hide ToolTip) */
    if(help){
        help = false;
        $(this).removeClass('helpSelected');

        $('.hintModal > .thisIsHint').removeClass('hintModal_container');
        $('.hintModal > .thisIsHint').addClass('hintHide');
        $('#status').text("");
    }
    /* Explain Mode (Show ToolTip) */
    else{
        help = true;
        // add result, status tag
        $(this).addClass('helpSelected');

        $('.hintModal > .thisIsHint').removeClass('hintHide');
        $('.hintModal > .thisIsHint').addClass('hintModal_container');
        $('#status').text("Help");
    }
});
```

- jQuery 코드 -

1. html 코드 작성 - hintModal 클래스를 가지는 요소 하위에 div태그를 만들고 숨김 속성 지정
2. Help 버튼을 클릭하면 help = true 대입
3. help 변수가 true면 툴팁 기능을 제공하기 위해 hintModal_container 클래스 적용, false일 경우는 툴팁 기능을 없애기 위해 위의 클래스 적용 해제

+ 위의 팝업 메뉴와 같이 선택되어 있는 효과를 주기 위해 helpSelected 클래스 사용



\$(객체).removeClass('hintModal_container')



\$(객체).addClass('hintModal_container')

4) 괄호 입력 간소화

```

case 'sqrt\n':
    clickedText = 'sqrt()';
    parenNum = 1;
    parenIndex = displayValue.length + 5;
    break;
case 'exp\n':
    clickedText = 'exp()';
    parenNum = 1;
    parenIndex = displayValue.length + 4;
    break;
case 'sin':
    clickedText = 'sin()';
    parenNum = 1;
    parenIndex = displayValue.length + 4;
    break;

```

```

/* Active Next Button */
if(parenNum){
    $('#next').addClass('nextSelected');
}

```

```

/* Process Parenthesis */
if(parenNum){
    var tempStr = displayValue.substring(0, parenIndex);
    tempStr += clickedText;
    tempStr += ')';
    displayValue = tempStr;
    parenIndex++;

    // Print
    $('#userInput').text(displayValue);
    $('#status').text("Paren");
    return;
}

```

```

/* Install Next Event Handler */
$('#next').click(function(){
    if(parenNum){
        parenNum--;
        if(parenNum == 0){
            parenIndex = 0;
            $('#next').removeClass('nextSelected');
        }
    }
})

```

1. 괄호 연산이 필요한 연산자들은 입력될 때 parenNum(괄호 개수), parenIndex(값이 입력 되어야 하는 곳의 index)를 지정한다.



2. Next 버튼에 nextSelected 클래스를 추가 해 활성화 시킨다.

3. 위에서 parenNum 값을 지정 했으므로, if 문에 들어온다. 기존 문자열에 지정한 인덱스만큼 자르고 새로 입력 받은 값을 넣은 다음에 인덱스를 1 증가시키고 출력한다.

4. 사용자가 입력을 마치고 Next 버튼을 클릭하면, 괄호의 개수(parenNum)를 한 개 줄인다. 만약 모든 괄호에 대해 값이 입력되었다면, Next 에 nextSelected 클래스를 제거해 비활성화한다.

5) 행렬, 벡터 입력 간소화

행렬, 벡터는 비슷하게 작동해, 보고서에는 행렬을 예시로 들겠다.

[행렬] 버튼을 클릭하여 행렬을 바로 간단히 입력할 수 있고, det(행렬식), inv(역행렬)를 누르면 자동으로 [행렬] 버튼이 클릭 되어 이어서 간단히 입력할 수 있다.

```

case 'det\n':
    clickedText = 'det';
    operationValue = 'matrix';
    break;
case 'inv\n':
    clickedText = 'inv';
    operationValue = 'matrix';
    break;

else if(clickedText == "det" || clickedText == "inv"){
    // Matrix
    displayValue += "(";
    matrixBool = true;
    tempMatrixStrBool = true;
    $('.matrix').addClass('matrixSelected');
}

if(matrixBool){
    if(clickedText == 'AC' || clickedText == '←' || clickedText == 'EV')
        return;
    if(tempMatrixStrBool){
        // Only first time
        tempMatrixStr = displayValue;
        tempMatrixStrBool = false;
    }

    if(clickedText == 'x'){
        // Input Value is x
        colStatus = true;
        tempMatrixStr += 'x';
    }
    else if(colStatus){
        // Input Value is Col
        col += clickedText;
        tempMatrixStr += col;
    }
    else{
        // Input Value is Row
        row += clickedText;
        tempMatrixStr += row;
    }
    $('#status').text("M / V");
    $('#userInput').text(tempMatrixStr);

    return;
}

```

1. 행렬 입력이 필요한 det, inv 연산 입력 시 operationValue에 행렬을 뜻하는 'matrix'를 대입한다.

x, y, z	f()	cross	dot	det	inv
() []	로그함수	[행렬]	(벡터)	exp	x^y

2. det, inv 연산이 들어오면 matrixBool, tempMatrixStrBool 변수에 true를 넣고, [행렬] 버튼에 클래스를 추가하여 버튼을 활성화한다.



3. 이곳에 처음 들어왔다면 현재 수식을 복사한다. 이후 사용자는 2x3(2 행 3 열) 방식으로 입력을 하게 되는데, 왼쪽은 이에 대한 처리이다.

```

/* Install Matrix Event Handler */
$('.matrix').click(function(){
    if(matrixBool){
        $(this).removeClass('matrixSelected');
        matrixBool = false;
        colStatus = false;

        if(row > 0 && col > 0){
            var printValue = "[";
            for(var i = 0; i < row; i++){
                printValue += "[";
                for(var j = 0; j < col - 1; j++){
                    printValue += ",";
                }
                printValue += "],";
            }
            printValue = printValue.substring(0, printValue.length - 1);
            printValue += "]";
        }

        matrixRowNum = row;
        matrixColNum = col;
        matrixOriginalColNum = col;
        matrixIndex = displayValue.length + 2;
    }
})

```

```

$('#next').addClass('nextSelected');
$('#userInput').text(displayValue);
$('#result').text('');
$('#status').text("Input");

```

```

/* Process Matrix */
if(matrixRowNum || matrixColNum){
    if(matrixColNum){
        var tempStr = displayValue.substring(0, matrixIndex);
        var tempRightStr = displayValue.substring(matrixIndex, displayValue.length);

        tempStr += clickedText;
        tempStr += tempRightStr;

        displayValue = tempStr;
        matrixIndex += 1;

        // Print
        $('#userInput').text(displayValue);
        $('#status').text("Matrix");
        return;
    }
}

```

3. 사용자로부터 $m \times n$ 입력을 받아 행과 열을 특정하게 되면, 사용자는 [행렬] 버튼을 클릭한다.

왼쪽은 이에 대한 처리인데, 특정한 행, 열 정보를 가지고 행렬의 틀을 만들고, 행렬에 값을 입력하기 위해 필요한 변수의 값을 대입한다.



4. 행렬에 값을 입력 받기 위해 Next 버튼에 클래스를 추가해 활성화 시킨다.



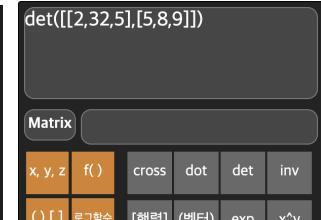
5. 위에서 괄호 처리를 하듯이 index 변수와 Next 버튼을 사용하여 행렬에 입력된 값을 추가 한다.

```

if(matrixRowNum == 0){
    // Visit All index
    matrixIndex = 0;
    matrixColNum = 0;
    $('#next').removeClass('nextSelected');

    if(operationValue == "matrix" || operationValue == "vector"){
        displayValue += ")";
        $('#userInput').text(displayValue);
        operationValue = "";
    }
}

```



6. 최종적으로 행렬에 모든 곳에 값을 입력하고 Next 버튼을 클릭하면 마무리 된다.

+ 여기서 함수의 경우(det, inv)에는 ')' 수식이 포함되어 입력된다.

6) 복사 & 붙여넣기

```

373 /* Install Copy, Paste Event Handler */
374 $('.copyPaste').click(function(e){
375     var key = $(this).text();
376     var sp = key.split(' ');
377     key = sp[0];
378
379     if(key == "Copy\n"){
380         var text = $('#userInput').get(0);
381         var selection = window.getSelection();
382         var range = document.createRange();
383
384         /* Copy to Clip Board */
385         range.selectNodeContents(text);
386         selection.removeAllRanges();
387         selection.addRange(range);
388         document.execCommand('Copy');
389
390         /* Copy to Local Variable */
391         copied = $('#userInput').text();
392         $('#status').text("Copy");
393     }
}

```



1. 사용자가 Copy 버튼을 클릭하면, 왼쪽의 이벤트 핸들러가 호출되며, 384~388 행은 클립보드로 복사를 진행하고, 391 행은 전역 변수에 저장한다.

```

else if(key == "Paste\n"){
    /* Append Paste */
    displayValue += copied;
    $('#userInput').text(displayValue);
    $('#status').text("Paste");
}

```



2. 이후 Paste 를 하면 기존 수식에 이어 붙이게 된다.

7) 히스토리 기능

```
/* Push to Susic, Result Array */
if(rightSusic){
    susic.push(temp);
    result.push(displayValue);
}
```

1. 계산을 할 때마다 유효한 수식이면 수식은 susic, 결과는 result 배열에 각각 값을 넣는다.

```
/* Install History Event Handler */
$('#history').click(function(){
    /* Add New Susic And Result */
    for(var i = 0; i < susic.length; i++){
        var content1 = '<tr><td class="susic">' +susic[i] + '</td>';
        var content2 = '<td class="result">' + result[i] + '</td></tr>';

        var $div = $(content1 + content2);
        $('#historyPrint').append($div);
    }

    /* Clear Array */
    susic.splice(0, susic.length);
    result.splice(0, result.length);

    /* Show Popup Screen */
    $(this).popModal({
        html : $('#historyContent').html()
    });
});
```

2. History 버튼을 클릭하면, 위에서 저장한 susic, result 배열에서 값을 가져와 새로운 html 태그를 만들어 낸다. 그리고 난 후 배열을 초기화하고, jQuery 플러그인을 사용해 화면에 표시한다.



-결과 화면-

4. 실제 문제에 대한 사용 예시

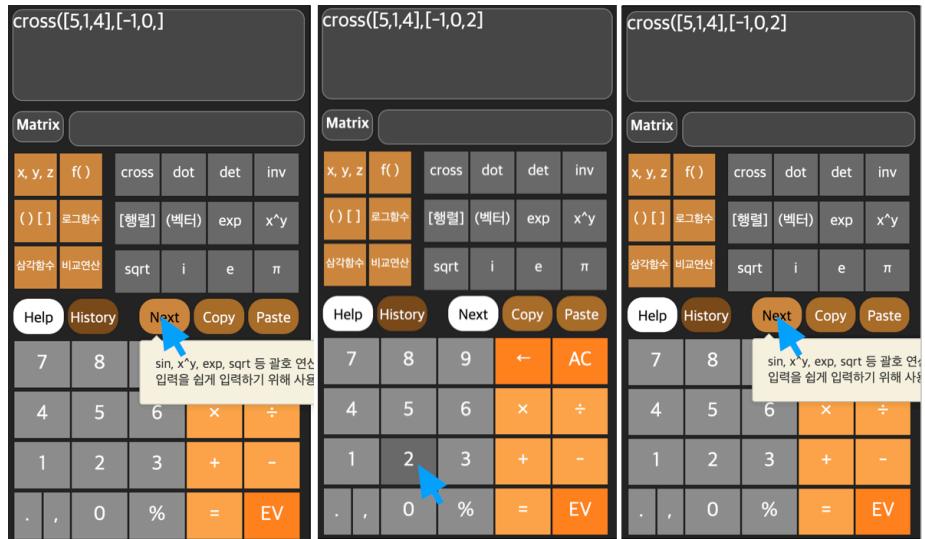
(링크 : <https://youtu.be/GxthLSBSnFQ>)

1. 두 벡터 $A = (5, 1, 4), B = (-1, 0, 2)$ 일 경우 외적 벡터 $A * B$ 를 구하여라. ↪

(첫 번째 문제이므로 설명 모드를 활성화하고 문제를 풀겠다)







cross([5,1,4],[-1,0,2])

Result [2, -14, 1]

$$2 \cdot (3 + 3i)(5 + 2i) = ?$$







3. 행렬 $A = \begin{pmatrix} 1 & -2 \\ 3 & 4 \end{pmatrix}$ 일 때, A 의 행렬식 $|A|$ 를 구하여라.



(중략)

5. 구현 측면에서 성공적인 부분과 실패한 부분

가장 성공적인 기능은 **도움말을 통한 설명 기능**이라고 생각한다. 사실, 이 부분은 언뜻 보면 매우 간단한 구현이라고 생각할 수 있다. 그러나 요소가 겹쳐 우선순위를 부여하는 과정에서 매우 많은 시간이 걸렸다. 밑의 사진과 같은 경우이다.



이는 초기 구현을 할 때 많이 해 맸던 부분이다. 오른쪽 연산들을 테이블로 구성했는데, 각 요소들이 형제, 자매가 되어 z-index로 우선순위를 아무리 주어도 구현하기 어려웠다. 따라서 다시 처음부터 테이블로 구현했던 모든 요소들을 잘게 쪼개며 전체 레이아웃을 재설계하여 문제를 해결하였다.

행렬 및 벡터 입력의 단순화 기능 또한 성공적이라고 생각한다. 이 부분은 정말 복잡하게 구현을 하여서 구현되는 것을 확인했을 때 너무 기뻤던 기억이 난다.

이제 실패, 포기한 부분에 대해서 언급하겠다. 초기 계산기를 구현할 때 다크 테마에 주황색 계열의 색으로 강조를 하기로 결정하였다. 그리고 사용자 경험을 넘어 서서 즐거운 인터페이스를 제공하기 위해 **화이트 테마 또한 제공하여**, 두 개의 테

마 중 사용자가 원하는 테마를 사용하도록 할 계획이었다. 그러나 많은 이유가 있지만 가장 큰 시간적인 이유로 구현하지 못하였다.

두 번째로, **즐겨찾기 기능**이다. 초기 계산기를 구현할 때 사용자가 자주 사용하는 수식을 등록하여 사용할 수 있도록 해 사용성을 극대화 할 계획이었다. 이도 마찬가지로 여러 이유로 구현하지 못하였다.

6. 사용성 측면에서 긍정적인 측면과 부정적인 측면

제일 먼저 긍정적인 측면으로 **도움말을 통한 설명 기능**이 있다. 이는 복잡한 과정없이 Help 버튼을 누르기만 하면, 간단하지만 매우 자세한 정보가 툴팁 형태로 제공되기 때문이다. 툴팁 형태로 제공되기 때문에 가장 좋은 점은 시연에서도 보았듯이, 각 연산이나 인터페이스의 설명을 보면서 동시에 계산기의 이용이 가능한 것이다.

두 번째로 긍정적인 측면은 **행렬 및 벡터 입력의 단순화 기능**이다. 계산기를 구현하는데 있어, 전체적인 통일감을 유지하려고 노력하였다. 괄호 연산 단순화 입력과 행렬 및 벡터 입력 방식이 매우 유사한 것도 이와 같은 이유다. 또한 행렬 및 벡터 입력의 단순화 과정에서 불필요한 창을 띄워 depth 를 늘리는 대신 기존의 입력 방식을 사용하여, 더욱 더 단순화 하였다.

마지막으로 **계산기의 현재 상태를 표시하는 기능**이다. 사용자는 계산기를 사용하면서 입력에 따라 어떻게 계산기가 반응하는지 확인하게 되고, 이는 계산기를 더 잘 활용할 수 있을 것이라 기대된다.

사용성에 있어 부정적인 면은 **Paste 기능**이라고 생각한다. 현재 구현한 붙여넣기 기능은 이어 붙이기(Append)기능만을 제공하고 있다. 그러나 사용자는 때로 덮어쓰기를 원할 수도 있지만 이에 대한 기능을 제공하고 있지 않다.

```
/* Original Paste */
/*
    displayValue = copied;
    $('#userInput').text(displayValue);
    $('#status').text("Paste");
*/
```

위의 사진은 덮어쓰기를 하는 코드이다. 코드로의 구현은 쉬우나 이를 어떤 방식으로 제공해야 할지 고민하다 결국 해답을 찾지 못한 채로 첫 번째 과제를 마무리 하였다.

7. 과제 결과에 대한 전반적인 자체 평가 및 향후 개선 계획

지금까지 과제를 하면서 사용자의 입장은 크게 생각해 본적이 없는데, 이번에 공학용 계산기를 구현하면서 항상 사용자의 입장에서 쉽고, 효율적인 인터페이스를 설계하기 위해 큰 노력을 한 것 같다. 사용자의 경험을 위한 조건으로 유용성, 사용성, 감성이 있다면, 유용성과 사용성은 위의 성공적인 부분, 긍정적인 측면에서 보듯이 어느정도 만족한 것 같다. 그러나 사용자에게 즐거움을 선사하는 점은 아직 조금 부족한 것 같다. 따라서 감성적인 측면을 보완하기 위해 위에서 언급한 다양한 테마를 제공하고 즐겨찾기 같은 기능을 제공해 사용자에게 즐거움을 주는 요소를 추가해야 겠다고 생각했다.