실습 보고서

과목	휴먼컴퓨터인터페이스
과제	기본계산기 구현
제출일	2020-05-31
소속`	컴퓨터소프트웨어학부
학번	2015726012
이름	최세일

목차

1. 요구조건 만족도 표

내용		구현 여부
1단계: WebUI 라이브러리	레이아웃 기능 구현을 위한 코드 제시 및 의미 설명	o
	로그인 페이지 재구성을 위한 코드 제시 및 결과 분석	o
2단계: 기본 계산기	구체적인 구현 방법(MyPushButton 클래스, initWidgets() 함수 등)	o
	실행 과정 제시 및 결과 분석	o
3단계: 확장 계산기	새로운 인터페이스/기능 개요	o
	추가된 상호작용 위젯	o
	추가된 레이아웃 위젯	x
	실행 과정 제시 및 결과 분석 (기본 계산기로부터 향상된 측면 강조)	o

2. 1 단계

2.1. 레이아웃 기능 구현을 위한 코드

```
if(properties != undefined) {   //# 위젯의 기본속성 지정. 부모자식간을 연결 시켜줌
    for(let name in properties) {
        let value = properties[name];
        if(name == 'children') {
            value.forEach(child => {
                child.parent = this;
                this.children.push(child);
            });
        }
        else {
            this[name] = value;
        }
    }
}
```

▲: 계층적 레이아웃을 구현하기 위해 부모 자식이 무엇인지에 대한 정보를 위젯이 가져야 하는데 Widget 생성자에 이 부분을 넣어줌

```
WebUI.Widget.prototype.layout = function() {
    //#(1)Measure size of widget in bottom-up order
    this.measure();

    //(2)Arrange each widget in top-down order
    this.arrange(this.position);
}
```

▲: 위젯을 배치해주는 함수. Measure 함수로 먼저 위젯의 사이즈를 측정하고 arrange 함수를 통해 배치한다.

```
WebUI.Widget.prototype.measure = function() {
    if(this.children.length > 0) {
        this.size_children = {width: 0, height: 0};
        this.children.forEach(child => {
            let size_child = child.measure();
            this.size_children = this.extendSizeChildren(this.size_children, size_child);
        });
        this.size = WebUI.maxSize(this.desired_size, this.size_children);
    }
    else {
        this.size.width += this.padding * 2;
        this.size.height += this.padding * 2;
    }
    return this.size;
}
```

▲: measure 함수 구현. 재귀를 통해 Bottom-up 방식으로 위젯의 크기를 잰다.

```
//#최댓값 크기 조정.
WebUI.maxSize = function(size1, size2) {
    let max_size = {width: 0, height: 0};

    max_size.width = (size1.width > size2.width) ? size1.width : size2.width;
    max_size.height  = (size1.height > size2.height) ? size1.height : size2.height;
    return max_size;
}
//#최솟값 크기 지정
WebUI.minSize = function(size1, size2) {
    let min_size = {width: 0, height: 0};

    min_size.width = (size1.width > size2.width) ? size1.width : size2.width;
    min_size.height = (size1.height > size2.height) ? size1.height : size2.height;
    return min_size;
}
```

▲: 처음 예제코드를 그대로 실행할 경우 에러 발생. maxSize, minSize 함수에 return 문을 적어줌

자식영역이 지정한 영역보다 클 경우 maxSize 를 통해 더 큰 값으로 지정해준다.

```
WebUI.Widget.prototype.arrange = function(position) {
    //arrange this
    this.moveTo(position);
    this.visual_items.forEach(item => {WebUI.canvas.add(item);});

    //arrange children
    if(this.children.length > 0) {
        let left_spacing = 0, top_spacing = 0;

        if(this.size.width > this.size_children.width) {
            let room_width = this.size.width - this.size_children.width;

            if(this.horizontal_alignment == WebUI.Alignment.LEFT)
                left_spacing = this.padding;
            else if(this.horizontal_alignment == WebUI.Alignment.CENTER)
                left_spacing = this.padding + room_width / 2.0;
            else if(this.horizontal_alignment == WebUI.Alignment.RIGHT)
                left_spacing = this.padding + room_widthh;
        }
        if(this.size.height > this.size_children.height) {
            let room_height = this.size.height - this.size_children.height;

            if(this.vertical_alignment == WebUI.Alignment.TOP)
                top_spacing = this.padding;
            else if(this.vertical_alignment == WebUI.Alignment.CENTER)
                top_spacing = this.padding + room_height / 2.0;
            else if(this.vertical_alignment == WebUI.Alignment.BOTTOM)
                top_spacing = this.padding + room_height;
        }

         let next_position = {left: position.left + left_spacing,
                              top: position.top + top_spacing};
         this.children.forEach(child => {
             child.arrange(next_position);
             next_position = this.calcNextPosition(next_position, child.size);
         })
     }

 }
```

▲: arrange 함수 구현. Top-Down 방식으로 위젯을 배치한다.

```
//#위젯의 크기를 결정하는 메소드
WebUI.Container.prototype.extendSizeChildren = function(size, child_size) {
    if(size.width < child_size.width) size.width = child_size.width;
    if(size.height < child_size.height) size.height = child_size.height;
    return size;
}
```

▲: Container 레이아웃 위젯이 자식의 사이즈를 측정하는 부분

```
//#위젯위치 결정하는 메소드
WebUI.Container.prototype.calcNextPosition = function(position, size) {
    let next_left = position.left;
    let next_top = position.top;
    return {left: next_left, top: next_top};
}
```

▲: Container 내부 위젯의 위치를 정하는 부분

```
//# Column에서 자식의 크기 결정
WebUI.Column.prototype.extendSizeChildren = function(size, child_size) {
    size.width += child_size.width;
    if(size.height < child_size.height) size.height = child_size.height;
    return size;
}
```

▲: Column 에서 자식들의 총 사이즈를 측정하는 부분

```
//# Column에서 다음위치 결정
WebUI.Column.prototype.calcNextPosition = function(position, size) {
    let next_left = position.left + size.width;
    let next_top = position.top;
    return {left: next_left, top: next_top};
}
```

▲: Column 에서 자식들의 위치를 측정하는 부분

```
//#Row의 자식사이즈 결정
WebUI.Row.prototype.extendSizeChildren = function(size, child_size) {
    if(size.width < child_size.width) size.width = child_size.width;
    size.height += child_size.height;
    return size;
}
```

▲:Row 내부에서 자식사이즈를 정하는 부분

```
//#Row에서 다음위치 결정
WebUI.Row.prototype.calcNextPosition = function(position, size) {
    let next_left = position.left;
    let next_top = position.top + size.height;
    return {left: next_left, top: next_top};
}
```

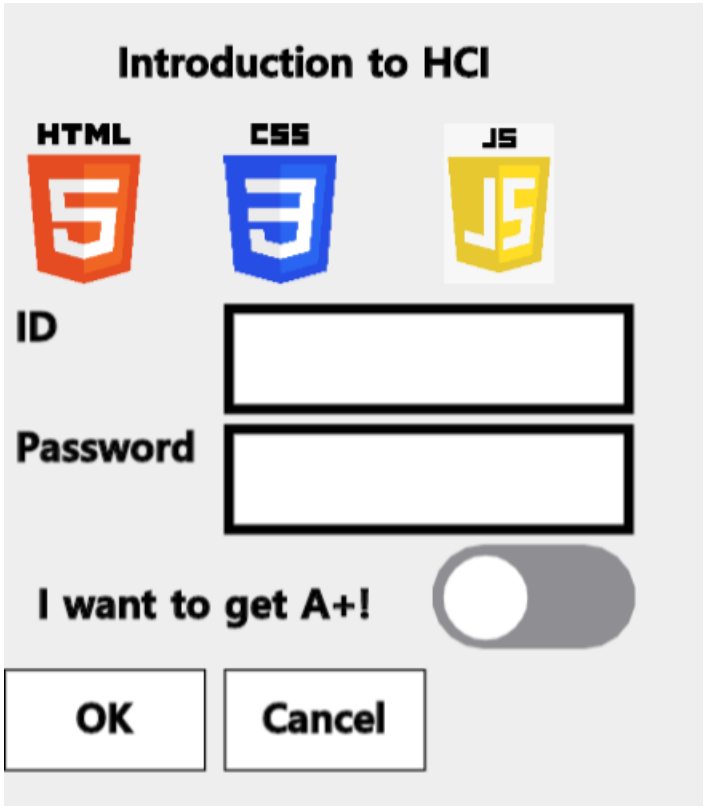
▲: Row 내부 자식의 위치를 정하는 부분

```
//# 위젯 활성화
WebUI.initWidgets = function() {
    WebUI.app = new WebUI.Row({
        children: [
            new WebUI.Container({
                desired_size: {width: 300, height: 60},
                horizontal_alignment: WebUI.Alignment.CENTER,
                vertical_alignment: WebUI.Alignment.CENTER,
                children: [new WebUI.Text("Introduction to HCI"),]
            }),
            new WebUI.Column({
                children: [
                    new WebUI.Image("resources/HTML5.png", {width: 100, height: 80}),
                    new WebUI.Image("resources/CSS3.png", {width: 100, height: 80}),
                    new WebUI.Image("resources/JS.png", {width: 100, height: 80})
                ]
            }),
            new WebUI.Column({
                children: [
                    new WebUI.Container({
                        desired_size: {width: 110, height: 60},
                        horizontal_alignment: WebUI.Alignment.LEFT,
                        vertical_alignment: WebUI.Alignment.LEFT,
                        children: [
                            new WebUI.Text("ID"),
                        ]
                    }),
                    new WebUI.TextField("", {width: 200, height: 50}),
                ]
            }),
```

▲ 최종 initWidgets 함수 부분. 새로운 계층 구조의 위젯들을 생성해준다.

```
new WebUI.Column({
    children: [
        new WebUI.Container({
            desired_size: {width: 110, height: 60},
            horizontal_alignment: WebUI.Alignment.LEFT,
            vertical_alignment: WebUI.Alignment.LEFT,
            children: [
                new WebUI.Text("Password"),
            ]
        }),
        new WebUI.TextField("", {width: 200, height: 50}),
    ]
}),
new WebUI.Column({
    children: [
        new WebUI.Container({
            desired_size: {width: 200, height: 60},
            horizontal_alignment: WebUI.Alignment.CENTER,
            vertical_alignment: WebUI.Alignment.CENTER,
            children: [
                new WebUI.Text("I want to get A+!")
            ]
        }),
        new WebUI.Container({
            desired_size: {width: 130, height: 60},
            horizontal_alignment: WebUI.Alignment.CENTER,
            vertical_alignment: WebUI.Alignment.CENTER,
            children: [
                new WebUI.Switch(false, {width: 100, height: 50})
            ]
        })
    ]
}),
new WebUI.Column({
    children: [
            new WebUI.PushButton("OK", {width: 100, height: 50}),
            new WebUI.PushButton("Cancel", {width: 100, height: 50})
    ]
}),
```

▲ initWidget 함수 나머지 부분



▲실행결과 위젯들의 배치가 원하는 대로 배치되었음을 확인할 수 있다.

3. 2 단계

3.1. 구현 방법(MyPushButton, initWidgets 등...)

```
WebUI.WidgetTypes = {
    UNDEFINED:      "undefind",
    TEXT:           "text",
    IMAGE:          "image",
    PUSH_BUTTON:    "push_button",
    MY_PUSH_BUTTON: "my_push_button",    //#타입 지정
    TEXT_FIELD:     "text_field",
    SWITCH:         "switch",
    CONTAINER:      "container",
    ROW:            "row",
    COLUMN:         "column"
};
```

▲일단 위젯타입을 추가로 지정해 줌

```
WebUI.parser = math.parser();    //#수식계산을 위한 객체 정의
```

▲수식계산을 위한 객체도 새로 정의한다.

```
//# MuPushBUtton 생성
WebUI.MyPushButton = function(label, desired_size, properties) {
    WebUI.PushButton.call(this, label, desired_size, properties);
    this.type = WebUI.WidgetTypes.MY_PUSH_BUTTON;

    this.onPushed = WebUI.MyPushButton.handleButtonPushed;
}

WebUI.MyPushButton.prototype = Object.create(WebUI.PushButton.prototype);
WebUI.MyPushButton.prototype.constructor = WebUI.MyPushButton;
```

▲MyPushButton 을 생성해준다. PushButton 을 상속받기 위해 내부에 PushButton.call 메소드를 이용하고 있다.

```
WebUI.PushButton.prototype.handleMouseDown = function() {
    if (!this.is_pushed) {
        this.translate({x:0, y:5});
        this.is_pushed = true;

        if (this.onPushed != undefined) {
            this.onPushed.call(this);
        }
        else{   //#함수 호출
            this.handleButtonPushed();
        }

        return true;
    }
    else {
        return false;
    }
}
```

▲ PushButton 을 상속받은 MyPushButton 에서 별도의 이벤트 처리를 위해 handleMouseDown 이벤트에 함수호출을 만들어 줌

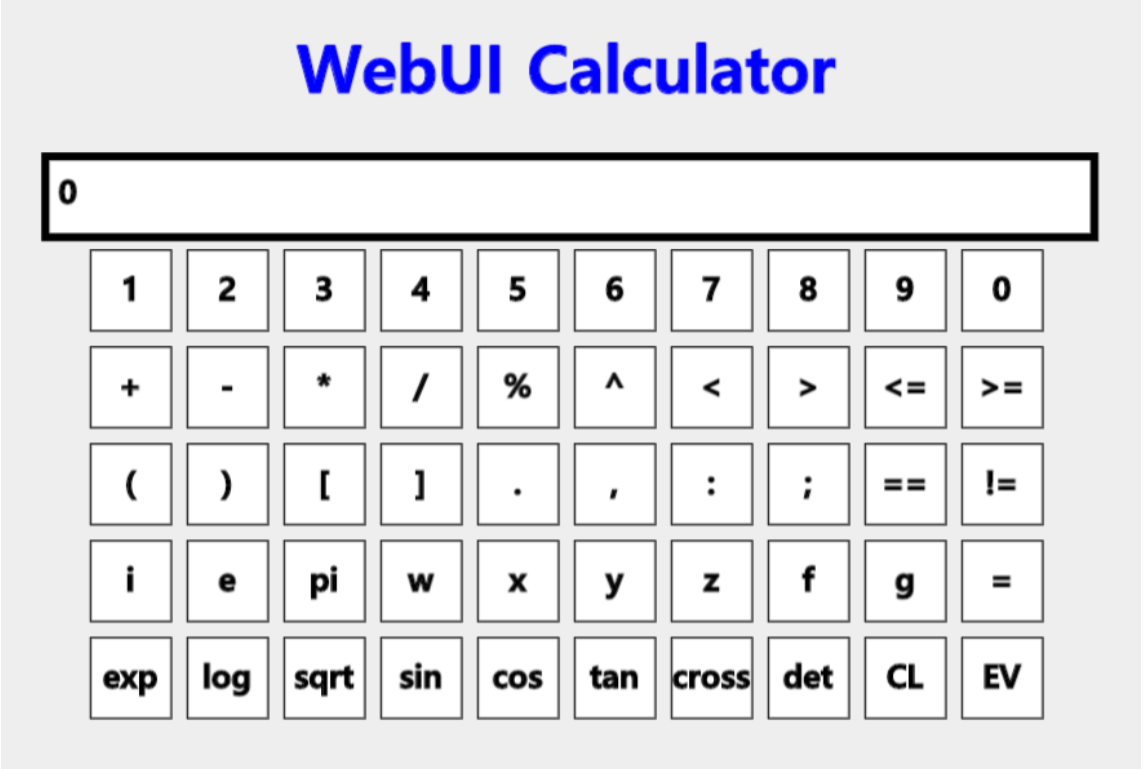
```
var displayValue = "0";          //# displayValue 생성
//#handleButtonPuhed 메소드 생성
WebUI.MyPushButton.prototype.handleButtonPushed = function() {
    let textF = WebUI.widgets[2];
    let textBox = textF.visual_items[1];
    console.log(textBox.text);

    if(displayValue == '0') displayValue = '';
```

▲handleButtonPushed 생성. 이 함수를 통해 누른 버튼을 확인하고 그 버튼에 맞는 이벤트 처리를 해준다. 결과값을 저장해줄 displayValue 변수도 따로 생성해 준다.

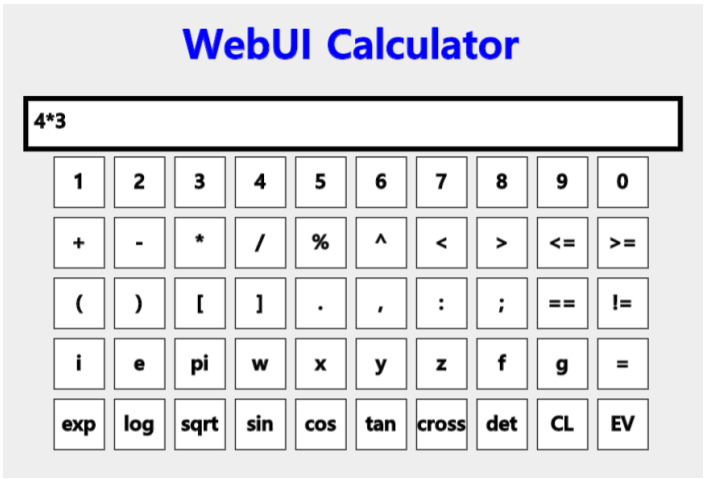
그리고 textF 에 텍스트 박스를 할당하고, textBox 에 Text 값을 조절할수 있도록 visual_items[1]를 할당해준다.

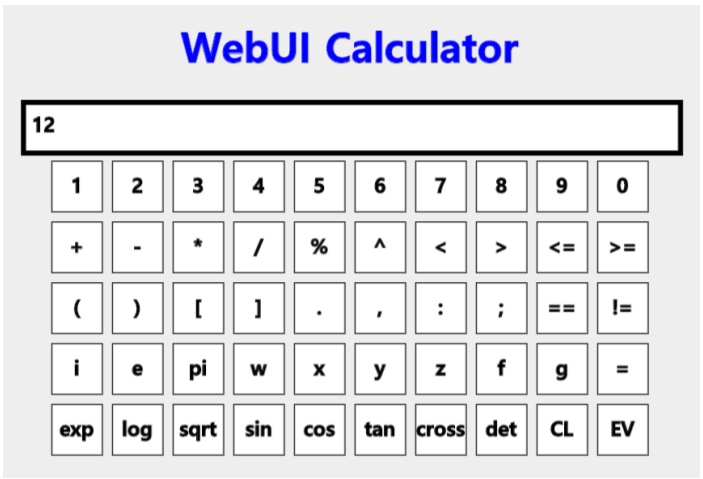
```
if(this.label == 'EV')
{
    try{
        displayValue = WebUI.parser.eval(displayValue).toString();
        var tokens = displayValue.split(' ');
        if(tokens[0] == 'function')
        {
            displayValue = tokens[0];
        }
        textBox.set('text', displayValue);
        displayValue = '0';
    }
    catch (ex)
    {
        displayValue = '0';
        if(displayValue != 'function')
        {
            textBox.set('text', ex.toString());
            console.log(ex.toString());
        }
    }
}
   else
   {
       if(this.label== 'CL')
       {
           displayValue = '0';
           textBox.set('text', displayValue);
       }
       else
       {
           displayValue += this.label;
           textBox.set('text', displayValue);
       }
   }

   console.log("displayValue: ", displayValue, ", 누른 버튼: ", this.label);
   WebUI.canvas.requestRenderAll();
}
```

▲눌린 버튼이 무엇인지에 따라 이벤트를 처리해준다. 마지막에는 디버깅을 위해 누른버튼의 정보를 콘솔창에 출력하고 있다.



▲실행 결과 화면





▲ 실행 결과 수식계산이 정상적으로 이뤄지는걸 확인할 수 있다.

4. 3 단계

4.1. 새로운 인터페이스/기능 개요

1. 새롭게 기능키와 숫자키를 위아래로 분류
2. 삭제 버튼 (위에서 <-)을 추가하여 수식 입력중 문자를 하나씩 삭제 가능
3. History 창을 추가하여 계산결과를 저장
    A. History 창은 숨겨져있다가 History 버튼을 클릭하면 위에 수식화면을 대체함

4.2. 추가된 상호작용 위젯

```
WebUI.MyListView = function(label, desired_size, properties) {
    WebUI.Widget.call(this, properties);

    this.type = WebUI.WidgetTypes.MY_LIST_VIEW;
    this.label = [];
    this.label.push(label);
    this.desired_size = desired_size;

    this.stroke_width = 5;
    this.fill_color = '#D3D3D3';
}

WebUI.MyListView.prototype = Object.create(WebUI.Widget.prototype);
WebUI.MyListView.prototype.constructor = WebUI.MyListView;
```

▲ 우선 계산결과를 저장하여 목록으로 보여주는 위젯을 구현하였다. 위는 새롭게 구현한 MyListView 의 구현 부분

```
WebUI.MyListView.prototype.initVisualItems = function() {
    let background = new fabric.Rect({ ···
    });
    text1 = new fabric.Text(this.label[0], { ···
    });
    text2 = new fabric.Text(this.label[0], { ···
    });

    this.size = this.desired_size;

    text1.left = this.position.left + 10;
    text1.top = this.position.top + 10;

    text2.left = this.position.left + 10;
    text2.top = this.position.top + 50;

    this.visual_items.push(background);
    this.visual_items.push(text1);
    this.visual_items.push(text2);
    this.visible = false;   //# 일단 안보이는 상태로 지정
    this.is_resource_ready = true;
}
```

▲ MyListView 의 초기화 부분. fabric.js 의 text 를 두 개 넣어 구현하였다. 위치는 위젯의 왼편에 위치하도록 정해주었다. visible 멤버변수를 추가하여 위젯을 보이지 않는 위치에 둔 뒤 버튼 클릭시 나타나도록 하였다.

```
//# 리스트 위치 조정
WebUI.MyListView.prototype.switchVisible = function() {
    let t1 = this.visual_items[1].text;
    let t2 = this.visual_items[2].text;

    if(!this.visible) { //#보이게 하기
        this.translate({x:0, y:-750});
        this.visible = true;

        WebUI.widgets[69].visual_items[1].set('text', t1);
        WebUI.widgets[69].visual_items[2].set('text', t2);

        WebUI.widgets[69].visual_items[1]
    }
    else {
        this.translate({x:0, y:750});
        this.visible = false;

    }
    WebUI.canvas.requestRenderAll();
}
```

▲ MyListView 의 이벤트를 담당하는 메소드. 보일 때마다 위치이동을 한다.

```
let history = [];    //#계산수식 저장배열
```

▲일단 계산 결과를 저장하기위해 전역변수를 선언하였다.

```
$(document).ready(function() {
    WebUI.initialize();

    WebUI.widgets[69].translate({x:0, y:100});//#
});
```

▲MyListView 의 위치를 일단 보이지 않는곳으로 옮김

```
if(this.label == 'EV')
{
    if(displayValue == '')//# 수식 입력 안했으면 결과값 유지
        return;
    try{
        //# 수식 저장
        historyBox.visual_items[2].set('text', historyBox.visual_items[1].text);
        history.push(displayValue);
        historyBox.visual_items[1].set('text', displayValue);

        displayValue = WebUI.parser.eval(displayValue).toString();
        var tokens = displayValue.split(' ');
        if(tokens[0] == 'function')
        {
            displayValue = tokens[0];
        }
        resultBox.set('text', displayValue);
        //# 결과값 저장
        historyValue.push(displayValue);

        displayValue = '0';
    }
```

▲버튼 내부에서 History 버튼 클릭시 처리해주기 위해 추가 작성한 코드부분. 계산 할 때마다 history 에 저장하고 MyListView 의 텍스트를 계산결과로 바꿔준다.

```
else
{
    if(this.label== 'CL')
    {
        displayValue = '0';
        textBox.set('text', displayValue);
    }
    else if(this.label == '<-') {   //# 텍스트 지우기 버튼
        if(displayValue.length <= 1) {
            displayValue = '0';
            textBox.set('text', displayValue);
        }
        else {
            displayValue = displayValue.substr(0, displayValue.length - 1);
            textBox.set('text', displayValue);
        }
    }
    else if(this.label == 'History') {  //# History 버튼

        historyBox.switchVisible();
    }
    else if(this.label == '◀' || this.label == '▶') {
        //#
    }
```

▲History 버튼 클릭시 switchVisible() 메소드를 호출하여 창을 보여준다. ◀,▶버튼을 이용하여 과거의 결과값을 선택할 예정이었으나 구현 실패

4.3. 추가된 레이아웃 위젯 (구현 실패)

- 우선 GridLayout 비슷하게 구현하고 싶었으나 실패함.

```
//열의 갯수를 지정하여 자동으로 위젯을 배치. 크기가 같은 위젯만 배치 가능
WebUI.MyGridLayout = function(gridCol, properties) {
    WebUI.Widget.call(this, properties);
    this.gridCol = gridCol; //# 갖게되는 열의 갯수
    this.gridRow = 0;       //# 갖게되는 행의 갯수
    this.currentCol = 0;    //현재 열
    this.currentRow = 1;    //현재 행
    this.type = WebUI.WidgetTypes.MY_GRID_LAYOUT;

    ////test
    console.log(properties.length);
}
```

▲원하는 열의 개수를 입력하여 열의 개수만큼씩 하위항목을 배치할 예정이었음

```
WebUI.MyGridLayout.prototype.extendSizeChildren = function(size, child_size) {
    if(this.currentRow == 0) {
        size.width += child_size.width;
        size.height = child_size.height;

        this.currentCol++;
        if(this.currentCol == this.gridCol) {
            this.currentCol = 0;
        }

        return size;
    }
    else {  //두번째 행 부터. 같은크기 위젯 상정했으므로 이제 높이만 변경해줌
        size.height = child_size.height * (this.gridRow + 1);   //갖는 행의 갯수만큼 높이 설정

        this.currentCol++;
        if(this.currentCol == this.gridCol) {
            this.currentCol = 0;
            this.gridRow++;
        }

        return size;
    }
}
```

▲MyGridLayout 에서 자식의 크기를 재는 부분.

4.4. 실행 과정 제시 및 결과 분석

▲ 위의 결과 실행 후, History 버튼 클릭시

▲ 위와같이 기존의 수식창에 추가된 상호작용 위젯이 나타나 계산수식이 저장된 모습을 보여줌

5. 논의

- 구현시 위젯을 원하는 위치에 보였다 다시 사라지는걸 반복할 수 있는 기능을 구현하고 싶었는데 그 위젯의 위치를 옮겨서 나타났다 사라지는 식으로 구현함
- 그러나 위젯의 위치를 옮기는 식으로 하여 앞으로 새롭게 위젯을 추가 구성시 이동위치를 매번 수동으로 정해주어야 하는 불편함이 존재
- 위의 불편함 때문에 새롭게 위젯을 만들시 처음부터 확장성에 염두를 두고 제작할 필요성을 느낌
- 수식의 계산을 저장하지만 계산결과를 불러들일수 없다는 한계점 존재
- 향후 위젯의 확장성과 재사용성을 염두에 두어 코드를 작성해봐야 하겠다.