휴먼 컴퓨터 인터페이스

과제#3 구현 완성도 개선

제출일 : 2018 년 5 월 27 일

컴퓨터소프트웨어학과

2016726031 어지예

개요

-모든 기능적 요구조건에 대한 구현 완성도 요약

수식 입력	정수, 실수, 복소수의 표현	O
	산술연산 (+, -, *, /, %, ^)	O
	비교연산 (==, !=, >, <, >=, <=)	O
	벡터 표현과 기본연산	O
	행렬 표현과 기본연산	O
	자주 사용되는 상수(pi, e)	O
	자주 사용되는 함수 (sin, cos, tan, exp, log, sqrt)	O
결과 출력	수식의 결과 값	O
	오류메시지	O
변수, 함수 정의 및 사용	변수 최소 3 개	O
	함수 최소 2 개	O
함수 그래프 기능 구현	사용자 정의 함수에 대한 그래프 그리기	O
	둘 이상의 함수에 대한 그래프 중첩 지원	O
	원점, 좌표축, 눈금 표시	O
	주요 좌표, 눈금 값 등 텍스트라벨로 표시	O
	그래프 표시 영역 조절 가능	O
	원점 위치, 정의역 구간 제어	△

-오픈소스 라이브러리 의존성 요약

저번과 마찬가지로 계산을 위한 math.js 라이브러리를 사용하였으며, 함수 그래프 구현을 위해 D3 라이브러리를 사용하였다.

-사용성 향상에 기여하는 핵심적인 상호작용 방식 요약

(기존)

입력한 식과 답을 동시에 볼 수 있다.

이전 계산 기록이 저장된다.

한 단계 전으로 돌아갈 수 있다.

(추가)

사용자 정의 함수에 대한 그래프를 그릴 수 있다.

도움말 모드를 이용하여 버튼들의 도움말을 얻을 수 있다.

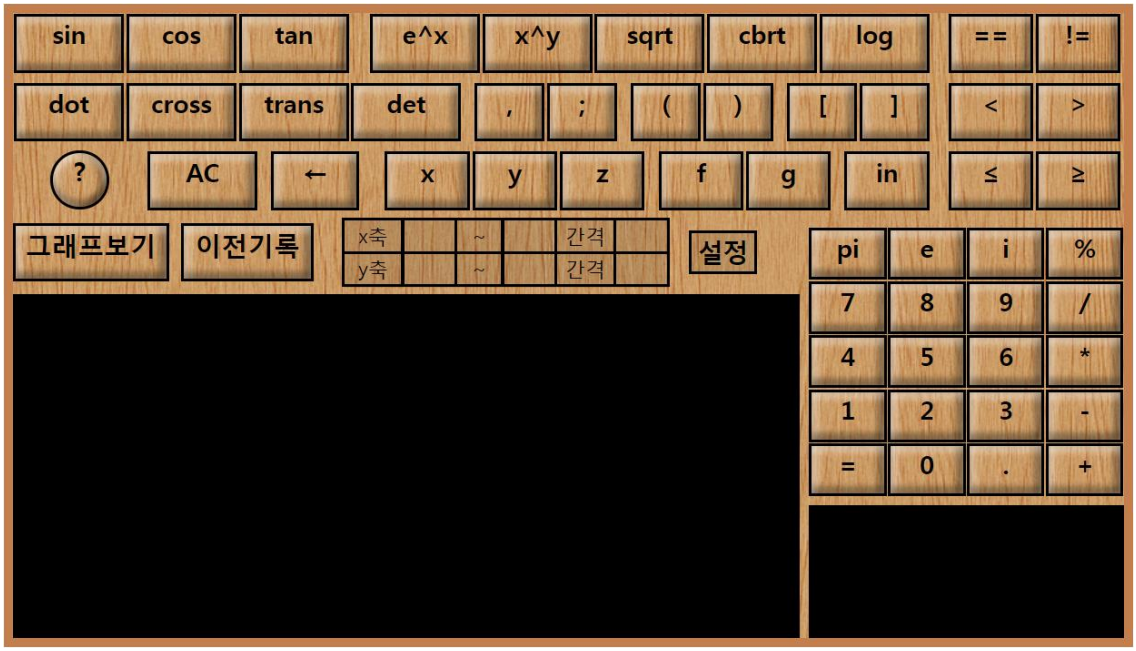
본문

-저번 과제와 변경된 부분과 그 이유

저번에 제출한 과제의 계산기에는 특수버튼이 이전기록을 보여주는 버튼 밖에 없었다.

도움말은 버튼위에 커서를 올려놓았을 때 나타나게 하였었다.

그렇지만, 도움말이 나타나면서 다른 버튼들을 가리고, 누르는데 불편함이 생긴다는 것을 깨닫고 도움말 버튼을 추가하여, 도움말 모드일 때 식부분에 도움말이 출력되게 하였다. 그리고 그래프를 그리기 위해 그래프 보기 버튼을 추가하였으며, 그래프보기 버튼을 누르면 그래프가 보이고, 다시 누르면 식출력화면으로 돌아온다. 그래프를 조금 더 크게 그리기 위해 결과값이 나오는 부분은 숫자 버튼들 밑으로 위치를 조정하였다. 또한 함수 키 부분을 비슷한 것끼리 묶어 간격을 조정하여줌으로써 한눈에 들어오게 하였다. 또한 코드 / 그래프 출력 창과 결과 출력 창의 배경을 검정색으로 하였다.



-사용자 인터페이스의 구성 요소 및 사용 방법

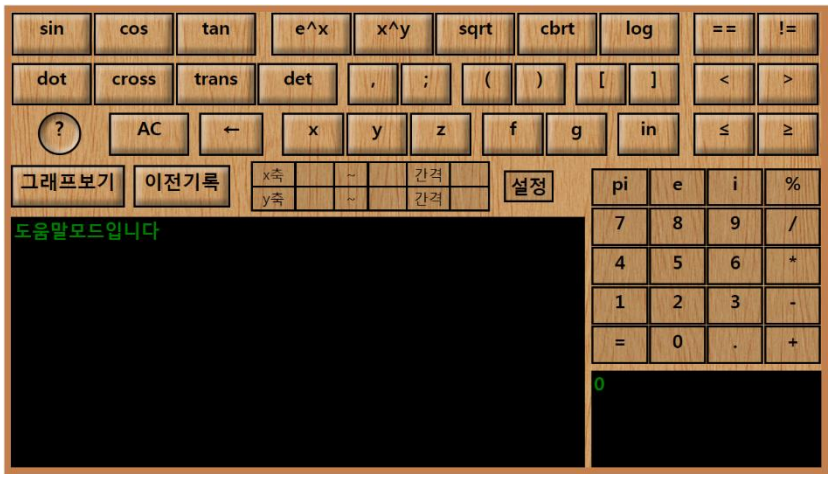
▷구성요소

<table>
<tr><td colspan="4">함수 키 부분
도움말</td></tr>
<tr><td colspan="3">그래프보기 | 이전기록 | 축 재설정</td><td rowspan="2">숫자 키 부분</td></tr>
<tr><td colspan="3" rowspan="2">코드 / 그래프출력 창</td></tr>
<tr><td>결과 출력 창</td></tr>
</table>

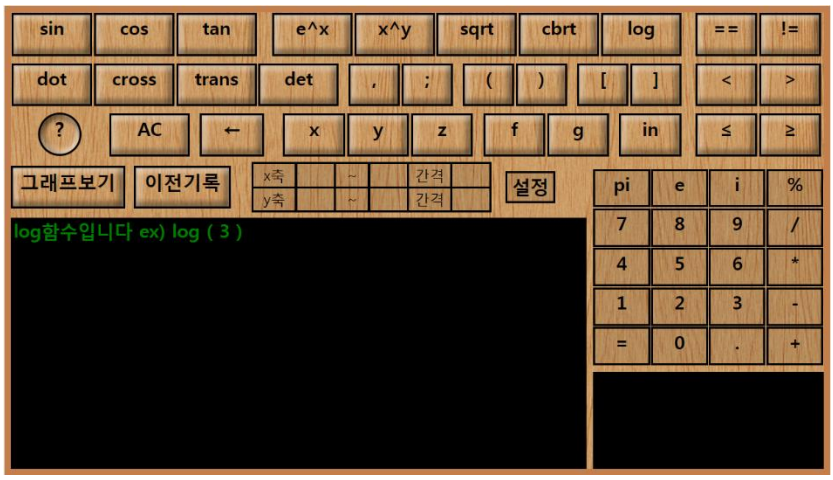
계산기는 크게 함수 키 부분과 숫자 키 부분, 코드 /그래프 출력 창과 결과 출력 창, 그리고 이전 기록 버튼, 그래프보기 버튼, 도움말 버튼, 그리고 축 재설정 부분으로 구성되어 있다.

※저번과 바뀌지 않은 기본 기능, 한단계 돌아가기, 초기화하기, 이전기록 보기는 쓰지 않았습니다.

▷도움말 모드

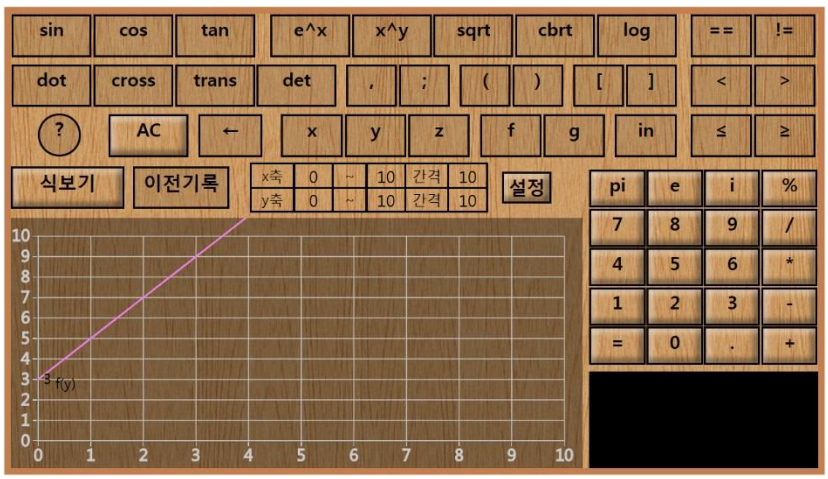


도움말 버튼을 클릭하면 도움말 모드로 바뀌며 코드 / 그래프 출력 창에 출력되어 도움말 모드라는 것을 알 수 있다. 또한 도움말이 필요없는 숫자 키 부분등은 누르지 못하게 설정이 바뀐다.



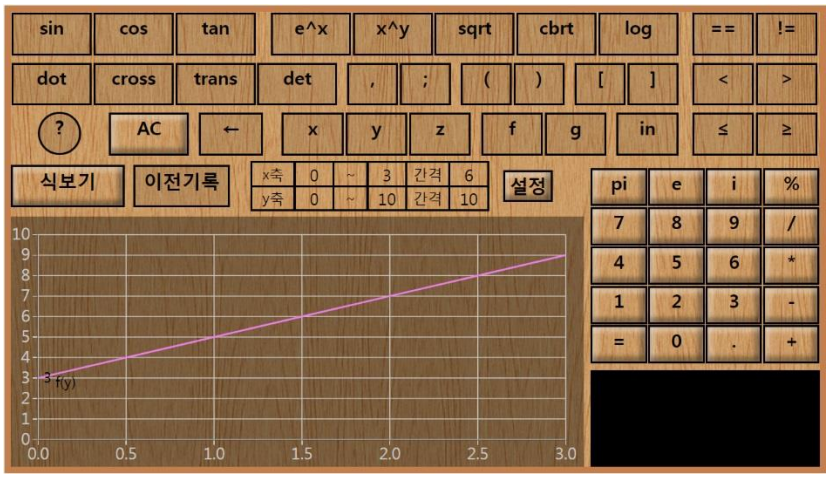
도움말을 얻고 싶은 버튼을 누르면 코드 / 그래프 출력 창에 도움말이 출력된다.

▷그래프 보기

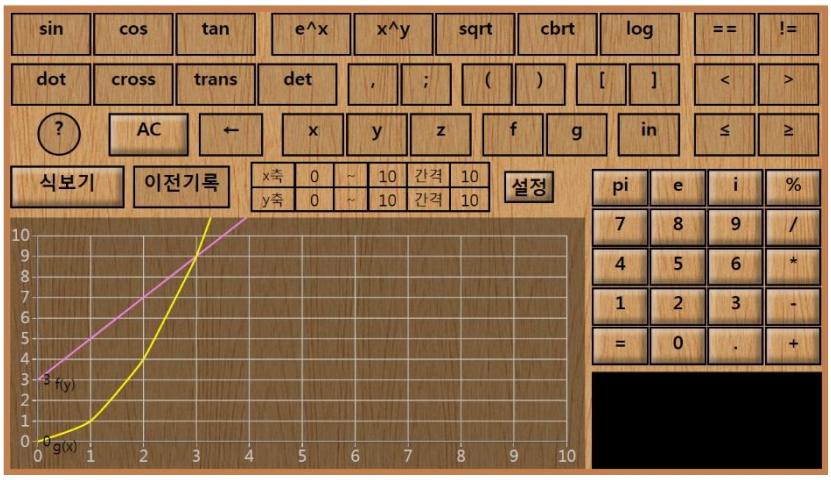


사용자 정의 함수를 입력하고, 그래프보기 버튼을 누르면 코드 / 그래프 출력 창에 그래프가 나타난다. 축 재설정 부분이 활성화되고, 쓰지 않을 버튼인 함수 키들과 이전기록 버튼은 비활성화되어 누를 수 없다.

그래프보기는 식보기로 바뀐다.



축 재설정 부분을 클릭하여 x, y 축의 범위와 간격을 숫자 키를 이용하여 입력할 수 있고, 설정 버튼을 누르면 재설정 된 축으로 다시 그려진다.



그래프는 최대 2 개까지 그릴 수 있으며 함수 f 는 보라색, 함수 g 는 노란색으로 출력된다.

-특징적인 상호작용 방식들에 대한 세부 구현 방법

▷계산기의 시작

```
<script type="text/javascript">
  $(document).ready(function() { //시작되었다
    $('.display').addClass("code"); //식 화면을 보여주며 시작
    d3.select("svg").style("opacity", 0); //안보이게
    $('.draw, .box').addClass("disable");
```

계산기의 코드 / 그래프 출력 창은 기본이 display 로 되어있다. 따라서 시작할 때 code 클래스를 추가하여 코드 출력 창이 바로 보이도록 해주었고, 그래프의 배경은 보이지 않게 설정하였다. 또한 축 재설정 부분은 비활성화 시키며 시작하였다.

▷도움말 모드

```
var h = false;
$(".help").click(function() {
  if ($('.show').text() == "식보기")
    return; //그래프 보일때 도움말모드x
  if (h == false) { //도움말모드
    h = true;
    $(this).addClass("sselected");
    $('.key').addClass("disable");
    $('.code').text("도움말모드입니다");
    $(".key_f, .prev, .show, .ac").addClass("helpmode");
  } else {
    h = false;
    $(this).removeClass("sselected");
    $('.key').removeClass("disable");
    $('.code').text("");
    $(".key_f, .prev, .show, .ac").removeClass("helpmode");
  }
});
```

도움말 모드가 켜져있는지 아닌지를 확인하는 변수 h 를 false 로 초기화하고, 도움말버튼(.help)이 눌리면 계속 눌러져 있는 것 같은 효과를 주는 클래스 sselected 를 추가하였다.

쓰지 않을 버튼인 숫자 키(key)에는 비활성화를 의미하는 disabled 클래스를 추가하였고, 도움말을 출력할 그래프보기 버튼(show), 이전기록 버튼(prev), 함수 키(key_f), AC 버튼(ac)에는 도움말 모드임을 의미하는 helpmode 클래스를 추가하였다. 아무일도 일어나지는 않지만 각각의 키가 도움말 모드를 가졌다는 것을 알려준다.

도움말 버튼을 다시한번 눌렀을 때는 도움말 모드가 해제되며, 위와 반대로 클래스를 제거해준다.

단, 그래프 모드일때는 도움말 버튼을 선택할 수 없게 설정하였다.

```
var hf_name = new Array(22);
var hf_how = new Array(22);
hf_name[0] = "sin";
hf_name[1] = "cos";
hf_name[2] = "tan";
```

```
hf_how[0] = "sin함수입니다 ex) sin ( 3 )";
hf_how[1] = "cos함수입니다 ex) cos ( 3 )";
hf_how[2] = "tan함수입니다 ex) tan ( 3 )";
hf_how[3] = "e의 제곱입니다 ex) e^x 3 ≫ e^3";
```

도움말 모드를 받을 키의 이름을 저장한 배열 hf_name 과 각각 키의 설명을 저장한 hf_how 를 초기화하였다.

```
$(".key_f, .prev, .show, .ac").click(function() {
  if ($(this).hasClass("helpmode") == true) {
    var i = 0;
    for (i = 0; i < 22; i++) {
      if ($(this).text() == hf_name[i])
        $(".code").text(hf_how[i]);
    } //도움말 출력하기
  }
});
```

도움말을 출력할 버튼들이 눌렸을 때, helpmode 클래스를 갖고있다면, 그 버튼의 이름과 맞는 hf_name 의 인덱스를 찾아 hf_how 를 코드 출력 창에 출력해주었다.

▷그래프보기

```
<style type="text/css">
  svg {
    box-shadow: inset 640px 0px 0px rgba(0, 0, 0, 0.4);
  }

  svg g path.line {
    stroke: violet;
    stroke-width: 2px;
    stroke-opacity: 1;
    fill: none;
  }
```

우선 css 에 그래프가 그려질 svg 와 선의 속성을 설정한다.

```
case 'f':
  n++; //누른개수
  funcf_on = true; //켠다
  d_array[n] = $(this).text(); //방금누른게 배열d에 저장
  displayValue += d_array[n]; //이어붙인다
  $('.code').text(displayValue); //식 입력
  break;
```

사용자 정의 함수를 저장하는 방법은 함수를 의미하는 f 나 g 가 눌렸을 때 각각 funcf_on 과 funcg_on 을 true 로 설정한다.

```
if ($(this).text() == '=') //계산버튼 누르면
{
  n = 0; //다시0됨
  l++; //저장
  ll = l - 1;
  list[l] = displayValue;
  list[l] += ' = ';
  if (funcf_on == true) {
    funcf_array = displayValue; //계산식 저장
    var str = funcf_array.split('=');
    funcf_name = str[0]; //이름이되고
    var str1 = funcf_name.split('(');
    var str2 = str1[1].split(')');
    funcf_x = str2[0]; //변수가 되고
    funcf_y = str[1].replace(new RegExp(funcf_x, "gi"), 'z');
    funcf_on = false; //끈다
  }
```

함수를 저장하기 위해 =버튼을 누르면 코드 출력 창에 출력된 함수 식이 funcf_array 에 저장되고, 그 식을 =를 기준으로 분할한다. 왼쪽은 함수의 이름이 되어 funcf_name 에 저장되고, 이름 사이에 있는 변수는 (와)로 분리되어 funcf_x 에 저장된다. 오른쪽은 함수의 식이고, replace 를 이용해 함수 식 안에 있는 변수를 문자 z 로 변환한 후 funcf_y 에 저장한다. Funcf_on 은 다시 false 로 초기화해준다. 같은 방식으로 g 가 입력되었을 때도 함수의 이름, 변수, 식을 저장해준다. 이렇게 하면 함수 f 와 g 를 각각 저장할 수 있다.

```
$(".show").click(function() {
  if (h == true)
    return; //도움말모드 아닐때
  if (funcf_name == '' && funcg_name == '') {
    $('.code').text("사용자 함수가 정의되어있지 않습니다");
    return;
  }

  if ($(this).text() == "그래프보기") {
    $(this).text("식보기");
    $('.code').text("");
    $('.result').text("");
    $('.display').removeClass("code");
    $('.key_f, .prev, .help').addClass("disable");
    $('.draw, .box').removeClass("disable");
```

그래프 보기 버튼(show)를 눌렀을 때 도움말 모드가 켜져있다면, 즉 h 가 trure 라면 실행되지 않는다. 또한 아직 사용자 함수가 정의되어있지 않다면 그래프는 보여지지 않는다.

Display 는 code 클래스를 없애 코드 출력 창이 사라지게 하고, 그래프가 보여지는 동안 쓰지 않을 함수 키 부분, 이전기록 버튼, 도움말 버튼은 비활성화를 의미하는 disable 클래스를 추가하였고, 축 재설정 부분의 비활성화를 해제하였다.

```
var margin = {
  top: 20,
  right: 20,
  bottom: 30,
  left: 30
};
var width = 630 - margin.left - margin.right;
var height = 275 - margin.top - margin.bottom;

var svg = d3.select(".myChart").append("svg")
  .attr("width", width + margin.left + margin.right)
  .attr("height", height + margin.top + margin.bottom)
  .append("g")
  .attr("transform", "translate(" + margin.left + "," + margin.top + ")");
```

그래프가 그려질 부분을 초기화하였다. Svg 에 꽉차게 그려지면 숫자가 삐져나가는 일이 생기므로 여백을 두고 그 안에서만 그려지게 설정하였다.

```
if (funcf_name != '') {
  var xs = 0,
    xf = 10;
  var ys = 0,
    yf = 10;
  var xl = 10;
  var yl = 10;

  var data = d3.range(xf - xs + 1).map(function(funcf_x) {
    return {
      x: funcf_x,
      y: math.parser().eval(funcf_y.replace(/z/gi, funcf_x))
    };
  });
```

만약 f 함수가 저장되어 있다면, 즉 funcf_name 이 공백이 아니라면, 이제 f 함수의 data 를 저장한다. 아까 식의 변수를 z 로 변환하였으므로, z 를 다시 funcf_x 로 변환한 후 eval 을 통해 계산 값을 y 에 넣는다.

```
$(".xstart").text(xs);
$(".xfinish").text(xf);
$(".ystart").text(ys);
$(".yfinish").text(yf);
$(".xsplit").text(xl);
$(".ysplit").text(yl);

var x = d3.scale.linear().domain([xs, xf]).range([0, width]); //범위!
var y = d3.scale.linear().domain([ys, yf]).range([height, 0]); //범위!

var xAxis = d3.svg.axis().scale(x).orient("bottom").ticks(xl).outerTickSize(1);
var yAxis = d3.svg.axis().scale(y).orient("left").ticks(yl).outerTickSize(1);
```

```
svg.append('g')
  .attr('transform', 'translate(0, ' + height + ')')
  .style("stroke", "#ccc")
  .call(xAxis);

svg.append("g")…

svg.selectAll("line")
  .style({
    "stroke": "#ccc",
    "shape-rendering": "crispEdges"
  });
svg.selectAll("text")
  .attr("fill", "#ccc")
  .style("font-size", "18px");

svg.selectAll("line.y")
  .data(y.ticks(yl))
  .enter().append("line")
  .attr("class", "y")
  .attr("x1", 0)
  .attr("x2", width)
  .attr("y1", y)
  .attr("y2", y)
  .style("stroke", "#ccc");

svg.selectAll("line.x")…
```

축 재설정 부분에 초기 x 축 시작점(xs), 끝점(xf), y 축 시작점(ys), 끝점(yf), 간격(xl,yl)이 출력되며, 그 값으로 x 축과 y 축이 범위와 간격에 맞춰 그려지게하였다.

x 축과 y 축을 그리고, 눈금을 표시한다. 또한 각각 간격에 맞추어 격자를 그려 선이 어디에 있는지 잘 알 수 있도록 하였다.

```
var line = d3.svg.line()
  .y(function(d) {
    return y(d.y);
  }).x(function(d) {
    return x(d.x);
  }).interpolate('monotone');

svg.append("path")
  .datum(data)
  .attr("class", "line")
  .attr("d", line);
```

```
svg.selectAll("dot")
  .data(data)
  .enter()
  .append("text")
  .attr('text-anchor', 'middle')
  .attr("x", function(d) {
    return x(d.x);
  })
  .attr("y", function(d) {
    return y(d.y);
  })
  .attr('dx', '30')
  .attr('dy', '10')
  .text(function(d) {
    if (d.x == 0)
      return funcf_name;
  });
```

```
svg.selectAll("dot")
  .data(data)
  .enter()
  .append("text")
  .attr('text-anchor', 'middle')
  .attr("x", function(d) {
    return x(d.x);
  })
  .attr("y", function(d) {
    return y(d.y);
  })
  .attr('dx', '10')
  .attr('dy', '5')
  .text(function(d) {
    if (d.x == 0)
      return d.y;
  }); //y축과 만나는점 표시
```

data 의 x 와 y 를 이용해 선을 그리고, 그래프가 y 축과 만나는 부분에 함수의 이름과, 그 값을 출력하였다.

같은 방법으로 함수 g 도 그린다. 대신, stroke 를 이용해 선의 색을 yellow 로 바꾸어 주었다.

```
  } else {
    $(this).text("그래프보기");
    $('.result').text("");
    $('.display').addClass("code");
    $('.key, .ac').removeClass("axis");
    $('.key_f, .key, .help, .prev').removeClass("disable");
    $('.draw, .box').addClass("disable");
    $(".xstart, .xfinish, .ystart, .yfinish, .xsplit, .ysplit").removeClass("axis")
    $(".xstart, .xfinish, .ystart, .yfinish, .xsplit, .ysplit").text("");
    d3.select("svg").style("opacity", 0);
  }
});
```

식보기 버튼을 누르면, 즉 그래프보기 버튼을 한번 더 눌렀을 때는 아까와 반대로 클래스들이 제거되고, 추가된다. 또한 svg 가 보이지 않게 설정된다.

▷축 재설정

```
$(".xstart, .xfinish, .ystart, .yfinish, .xsplit, .ysplit").click(function() {
  if ($('.show').text() == "그래프보기")
    return;
  $(".xstart, .xfinish, .ystart, .yfinish, .xsplit, .ysplit").removeClass("axis");
  $(this).addClass("axis");
  displayV = '';
});
var displayV = '';
```

축 재설정을 하기위해 x 축 시작점(xstart), 끝점(xfinish), y 축 시작점(ystart), 끝점(xfinish), x 축 간격(xsplit), y 축 간격(ysplit)부분을 눌렀을 때, 그래프보기모드라면, 누른 그 부분에 axis 클래스를 추가해준다. 누른 부분에만 추가하기위해 다른 모든 부분에 클래스를 없애고 누른부분에 추가하는 방식으로 하였다. axis 클래스는 조금 어두워짐으로써 이부분에 숫자가 입력될 것임을 보여준다.

```
$(".key, .ac").each(function(index, key) {
  $(this).click(function() {
    if (displayV == 0)
      displayV = '';
    if ($(this).text() == 'AC') {
      displayV = '0';
    } else {
      displayV += $(this).text();
    }
    if ($(".xstart").hasClass("axis"))
      $(".xstart").text(displayV);
```

숫자 키를 눌러 재설정 될 축의 숫자를 지정하는데, 숫자 키를 눌렀을 때 각각의 축 재설정 부분들이 axis 클래스를 갖고있다면 그 부분에 눌린 숫자들이 출력되도록 해주었다.

코드 출력 창에 누른 숫자를 출력할 때와 원리는 같으며 여기서도 AC 버튼을 누르면 그 부분이 초기화된다.

```
$(".draw").click(function() { //축 재설정
  if ($(this).hasClass("disable") != true) {
    $(".xstart, .xfinish, .ystart, .yfinish, .xsplit, .ysplit").removeClass("axis")

    xs = $(".xstart").text();
    xf = $(".xfinish").text();
    ys = $(".ystart").text();
    yf = $(".yfinish").text();
    xl = $(".xsplit").text();
    yl = $(".ysplit").text();

    d3.select("svg").remove(); //지우고 다시생성
```

이렇게 재설정된 축으로 그래프를 그리는 원리는 그래프를 처음그리는 것과 유사하다. 축 재설정 부분에 출력 되어있는 숫자로 각각 축의 시작점, 끝점, 그리고 간격을 초기화한 후 이미 나타나있는 svg 를 지운다. 그리고 아까와 마찬가지로 축, 눈금, 선을 그려주면 재설정된 축과 그에 따라 바뀐 그래프를 볼 수 있다.

-실제 문제에 대한 사용 예시

1) $\frac{\sin 2}{\cos 3+5}$

2) $\sqrt[3]{3} + 5^{\frac{1}{2}} - 1.2 \times 3^{\frac{2}{3}}$

3) 도움말 모드에서 dot 을 눌러본다.

4) $\mathrm{f(x)} = x^2 + 2x, \mathrm{g}(x) = 2\sin x$의 그래프를 그려보고, 축 재설정을 해본다.

수식이 길어 캡쳐 하지않고 동영상 링크 첨부합니다.

영상 링크 : https://www.youtube.com/watch?v=mY31JrYFAoA&t=7s

논의

-구현 측면에서 성공적인 부분과 실패한 부분

▷성공적인 부분

도움말 모드를 따로 만들어 버튼을 누르는데 지장이 없다. 도움말 모드라는 것을 나타내기 위해 helpmode 클래스를 추가하고 없애는 방식으로 한 것이 효과적이었다.

같은 방법으로 axis 클래스를 추가한 것으로 축 재설정도 가능했다.

비활성화되는 클래스를 이용해 각 모드마다 필요없는 버튼을 눌렀을 때 아무일도 일어나지 않아 방해가 되지않는다.

▷실패한 부분

그래프에서 원점의 위치를 바꾸는 법을 찾지 못하여 항상 왼쪽과 아래쪽에 그려지게 하였다.

축은 음수 부분도 표현이 되지만 그래프가 x 값이 음수인 부분에서는 그려지지 않는다.

그래프보기 버튼을 누르고 식보기 버튼을 눌렀을 때 버튼을 누르기 전의 식은 사라지고 초기 화면만 남는다.

출력한 내용이 길면 자동으로 다음줄로 넘어가게 하고 싶었지만 코드출력 칸과 결과 출력 칸을 넘어서 가로로 길게 써지는 문제를 해결하지 못하였다.

-사용성 측면에서 긍정적인 측면과 부정적인 측면

▷긍정적인 측면

마우스가 버튼 위에 올라갔을 때 시각효과를 주어 어떤 버튼을 선택하려고 하는지 쉽게 볼 수 있다. 또한 각각의 모드에 따라 누를 수 없는 버튼들은 비활성화를 시켜 어느 버튼을 누를 수 있고 없는지를 쉽게 볼 수 있다.

도움말모드에서 버튼을 누르면 도움말을 볼 수 있어 계산기를 처음 사용하는 사람도 버튼을 어떤 순서로 눌러야 하는지, 버튼을 어떤 용도로 사용할 수 있는지 쉽게 알 수 있다.

이전 기록을 확인할 수 있어 수식에 오류가 있었는지 확인할 수 있고 값을 다시 확인할 수 있다.

한단계 이전으로 돌아갈 수 있기 때문에 실수로 다른 버튼을 잘못 눌러도 언제든지 돌아갈 수 있으며 실수에 대한 부담감이 적어진다.

최대 두개의 그래프를 볼 수 있으며 그래프의 축의 시작점과 끝점을 변경하고, 나눠지는 구간도 설정할 수 있다.

▷부정적인 측면

X 의 음수부분 그래프가 그려지지 않기 때문에 그래프를 항상 양의 부분만 확인할 수 있다.

그래프 출력 창이 깔끔하지 않다.

이전 기록을 하나씩만 확인만 할 수 있고 활용도 할 수 없다.

-과제 #3 에 대한 전반적인 자체 평가

크게 그래프보기와 도움말 모드를 추가하였다. 그래프가 정상적으로 그려지고, 축 재설정에 따라 다시 그려지는 것도 만족스러웠다. 하지만 음수부분에서는 출력이 되지 않는 것은 끝내 해결하지 못하였다. 또한 그래프가 출력된 후에 알 수 있는 정보가 그래프와 이름, 그리고 y 축과 만나는 점 뿐이므로 그래프를 사용자가 잘 알아볼 수 있을지도 의문이다.

도움말 모드는 초기 계획했던 것에 맞게 동작하여 만족스러웠다.

시중에 파는 대나무 계산기를 목표로 삼았다. 따라서 계산기의 배경이 나무질감이고, 실제 버튼처럼 튀어나와있게 구현을 하였다. 결과가 출력되는 곳은 검은 배경으로 하고, 그래프도 밑에 배경이 있게 하긴 했지만 나무 질감 배경 때문인지 깔끔하지 못한 것은 어쩔 수 없었다.