情報学群実験第 2 レポート

学籍番号 1280391
細川 夏風

2026 年 2 月 15 日

1　背景

近年，VR などの発展によってバーチャル空間が身近なものとなっている．それ加えて，Web が最も身近に自身とインターネットなどのバーチャル空間を取り繋いでいると言える．そのため，バーチャル空間についてと Web についての両者について学びを得ることができる HTML(Hyper Text Markup Langage)，CSS(Cascading Style Sheets)，Javascript がサイバー空間の理解と発展に寄与すると考えられる．これらを目的に本実験を行った．

2　方法

課題についてはすべて以下付録に記載している．

環境

HTML，CSS，Javascript はブラウザ上で動作するマークアップ言語やスタイルシート言語，スクリプト言語であるためそれぞれに環境が設定されているというケースはあまり多くない．大きな変化として HTML5 等は存在するがそのような形式のみが基本的な形式となっているため，特定の環境を表記する必要性は少ない．

第 1 回

HTML

HTML はタグを用いて文章の構造を記述する言語である．<何かしらのタグ>で始まり，何かしらの文章や形式を記述し，</何かしらのタグ>で閉じるという形式で多くは記述される．

CSS

CSS は文章の見栄えや体裁を整える言語である．HTML によって記述された何かしらのタグについて表記するケースがほとんどである．また，HTML のタグ内に記述された class について記述することも可能である．.HTML のタグまたは class 体裁 のような記述で用いられる．

また，HTML は表のようなテーブル構造を作成することができる．それぞれの行と列に対してヘッダー等を設定することができる．課題の表を作る HTML ファイルは以下付録に記載している．

第 2 回

Javascript

Javascript は Web で用いられるスクリプト言語の一種である．Web 上で動きをつけるなどの動作を伴った処理に用いられることが多い．また，スクリプト言語であるため少し複雑性を持っている処理についても記述することが可能となっている．HTML のタグに加えられた id などを取得することによって単一に処理を追記することも可能．

HTML は DOM(Document Object Model) と呼ばれるデータ構造を用いて記述されている．これは木構造に近い概念を持っている．いずれかのオブジェクトの下に何かしらのオブジェクトを持つことができる．また，それらについてそれがどのような形式のオブジェクトであるかも記述することができる．この構造を活用し，それに追記や削除，作成などをすることで Web ページを自由にカスタマイズすることができる

また，Javascript は動的型付けを採用しており型にリソースを割く必要がないことやクリック，リロードどの個々のイベントについての動作を記述することができる．

課題では，本来はそれぞれ個々に id を準備してそれぞれ追加の処理を行うことによって可能であるが，この方法はあまりにも冗長である．よって一度のすべての td タグを取得し，それぞれに追加の処理を行うことによって実現している．

HTML の id と class の違いは，id は単一のものでなければならないが class は単一でなくても良い．

第 3 回

WebGL

WebGL とはブラウザ上で動作するグラフィックを描画する仕組みのことである．HTML 内に Canvas を配置することにより，その中にグラフィックを記述することができる．WebGL では三角

形ポリゴンのみを用いる事ができる．この三角形を組み合わせることによってあらゆる形状を表現できる．

今課題については，四角形と三角形，円を描くというものだ．基本的に三角形のみしか描けない WebGL について四角形は三角形を 2 つ逆に並べることによって実現可能である．また，円についてはその三角形を回転させることによって実現している．time による for 文の部分がその処理を担っている．

第 4 回

アニメーション

WebGL の図形はそれぞれ，配置の際に座標を指定している．その座標について値を変えることによって図形を移動させることなどができる．この際，その移動が平行移動なのか回転移動なのかなどによって処理が少し異なる．アニメーションについては図形の配置座標についてのみで無くカメラ座標を移動させることによって実現することができる．今課題については図形を回転させるという方法をとっている．

第 5 回

図形のスタック

WebGL にはスタックの概念が存在し，これにより図形を複製が簡易なものになる．今課題については初期の行列についてスタックに保存し，図形の位置を変え，スタックからポップした初期行列から先ほどとは逆方向に移動させる．

3　考察

本実験では，Web とバーチャル空間について学ぶために HTML，CSS，Javascript，WebGL を扱った．まず DOM 操作についてだが，課題ではテーブルのセルに文字を追加する処理を行った．本来なら個々に ID を振って処理することも可能だが，それはあまりにも冗長である．そのため，一度すべての td タグを取得し，まとめて処理する方法をとった．DOM の木構造のような概念を理解すれば，Javascript から Web ページを自由にカスタマイズすることができると言える．次に WebGL について．WebGL では三角形ポリゴンのみを用いる事ができるという制約があるが，四角形は三角形を 2 つ逆に並べることによって実現可能である．円についてもその三角形を回転させることによって実現している．単純な図形の組み合わせであらゆる形状を表現できるという点は興味深い．アニメーションに関しては，今課題については図形を回転させるという方法で実現した．座標の値を変えることで図形は移動する．また，行列のスタックを使うと図形を複製が簡易なものになるという

のも重要な点だ．Web は最も身近に自身とインターネットなどのバーチャル空間を取り繋いでいる存在である．今回の実験を通して，普段見ている Web ページやグラフィックがどのように作られているのか，その裏側の仕組みについて少し理解が深まったと考えられる．

参考文献

[1]

A　ソースコード

Day1-2

Listing 1　Day1-2.html

```
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <title>1週間予定表</title>
    <link rel="stylesheet" href="Day1-2.css">
    <script type"text/javascript" src="Day1-2.js"></script>
</head>
<body>
  <center>
    <table border="1">
        <caption>1週間の予定表</caption>

        <tr>
            <th>1/20</th>
            <th>1/21</th>
            <th>1/22</th>
            <th>1/23</th>
            <th>1/24</th>
            <th class="sat">1/25</th>
            <th class="sun">1/26</th>
        </tr>

        <tr>
            <td>CR概論</td>
            <td>実験第2</td>
            <td id="trip">出張</td>
            <td>CR概論</td>
            <td>実験第2</td>
            <td class="sat"></td>
```

```
            <td class="sun"></td>
        </tr>
    </table>
    <input type="text" id="input_text" placecholder="入力">
  </center>
</body>
</html>
```

Listing 2　Day1-2.css

```
/* 出張（オレンジ） - IDで指定 */
#trip {
    background-color: orange;
}

/* 土曜（水色） */
.sat {
    background-color: lightblue;
}

/* 日曜（ピンク） */
.sun {
    background-color: pink;
}
```

Listing 3　Day1-2.js

```
window.onload = function() {
    // 全てのtdタグを取得
    var cells = document.getElementsByTagName("td");

    // ループでイベントを設定
    for (var i = 0; i < cells.length; i++) {
        cells[i].onclick = function() {
            var val = document.getElementById("input_text").value;
            if (val == "") return;

            // div要素を作り、テキストを入れて、セルに追加(appendChild)す
                る
```

```
            var div = document.createElement("div");
            div.appendChild(document.createTextNode(val));
            this.appendChild(div);
        };
    }
};
```

Day3

Listing 4　Day3_start.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>WebGL入門</title>
    <script src="./Day3_start.js"></script> <!-- main.jsを読み込む -->

        <!-- create vertex shader -->
    <script id="vertexShader" type="x-shader/x-vertex">
      attribute vec3 position;
      void main(void){
        gl_Position = vec4(position, 1.0);
      }
    </script>

    <!-- create fragment shader -->
    <script id="fragmentShader" type="x-shader/x-fragment">
      precision mediump float;
          uniform vec4 uGlobalColor;
      void main(void){
        gl_FragColor = uGlobalColor;
      }
    </script>
  </head>
  <body>
    <!-- bodyタグの中にCanvasを配置する -->
    <canvas id="canvas" width = 200, height = 200></canvas>
  </body>
```

```
</html>
```

Listing 5　Day3_start.js

```
document.addEventListener('DOMContentLoaded', function () {
  // HTMLからcanvas要素を取得する

  const r = 0.15; // 半径
  // canvas要素からwebglコンテキストを取得する
  const gl = canvas.getContext('webgl');

  // WebGLコンテキストが取得できたかどうか調べる
if (!gl) {
    alert('webgl not supported!');
    return;
  }

  // canvasを初期化する色を設定する
  gl.clearColor(0.0, 0.0, 0.0, 1.0);//RGBAの順で0--1の範囲

  // canvasを初期化する
  gl.clear(gl.COLOR_BUFFER_BIT);
  const program = gl.createProgram();

  // シェーダのソースを取得する
  const vertexShaderSource = document.getElementById('vertexShader').
      textContent;
  const fragmentShaderSource = document.getElementById('fragmentShader').
      textContent;

  // シェーダをコンパイルして、プログラムオブジェクトにシェーダを割り当て
      る
  const vertexShader = gl.createShader(gl.VERTEX_SHADER);
  gl.shaderSource(vertexShader, vertexShaderSource);
  gl.compileShader(vertexShader);
  gl.attachShader(program, vertexShader);
  const fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
  gl.shaderSource(fragmentShader, fragmentShaderSource);
```

```
32  gl.compileShader(fragmentShader);
33  gl.attachShader(program, fragmentShader);
34
35  // シェーダをリンクする
36  gl.linkProgram(program);
37  // プログラムオブジェクトを有効にする
38  gl.useProgram(program);
39
40  // 3つの頂点の座標を定義する
41  const triangleVertexPosition = [
42    0.0, 0.2, 0.0,
43    -0.3, -0.2, 0.0,
44    0.3, -0.2, 0.0,
45    0.025, -0.2, 0.0,
46    -0.025, -0.2, 0.0,
47    0.025, -0.7, 0.0,
48    -0.025, -0.2, 0.0,
49    -0.025, -0.7, 0.0,
50    0.025, -0.7, 0.0
51  ];
52
53  const circleVertexPosition = [
54   0 , 0.35, 0.0
55  ]
56  const time = 60;
57  for (let i = 0; i <= time; i++) {
58    var rad = (i / time) * 2 * Math.PI;
59    let x = Math.cos(rad) * r / (canvas.width / canvas.height);
60    y = Math.sin(rad) * r + 0.35;
61    circleVertexPosition.push(x, y , 0.0);
62  }
63  // 頂点バッファを作成する
64  const triangleVertexBuffer = gl.createBuffer();
65  // 頂点バッファをバインドする
66  gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexBuffer);
67  // 頂点バッファに頂点データをセットする
```

```
68   gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(triangleVertexPosition
         ), gl.STATIC_DRAW);
69
70   // Positionのロケーションを取得し、バッファを割り当てる
71   const positionLocation = gl.getAttribLocation(program, 'position');
72   gl.enableVertexAttribArray(positionLocation);
73   gl.vertexAttribPointer(positionLocation, 3, gl.FLOAT, false, 0, 0);
74
75   uGoldColor = gl.getUniformLocation(program, "uGlobalColor");
76   gl.uniform4fv(uGoldColor, [0.0, 0.0, 1.0, 1.0]);
77
78   //三角形
79   gl.drawArrays(gl.TRIANGLES, 0, 3);
80   gl.uniform4fv(uGoldColor, [0.0, 1.0, 0.0, 1.0]);
81   gl.drawArrays(gl.TRIANGLES, 3, 3);
82   gl.uniform4fv(uGoldColor, [0.0, 1.0, 0.0, 1.0]);
83   gl.drawArrays(gl.TRIANGLES, 6, 3);
84
85   //円
86   gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(circleVertexPosition),
         gl.STATIC_DRAW);
87   gl.uniform4fv(uGoldColor, [1.0, 0.0, 0.0, 1.0]);
88   gl.drawArrays(gl.TRIANGLE_FAN, 0, circleVertexPosition.length / 3);
89   gl.flush();
90
91   });
```

Day4

Listing 6　Day4_sample4.html

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>WebGL入門</title>
5      <script type="text/javascript" src="./gl-matrix-min.js"></script>
6      <script src="./Day4_sample4.js"></script> <!-- main.jsを読み込む -->
7
```

```
8       <!-- ここから追記 -->
9       <!-- create vertex shader -->
10      <script id="vertexShader" type="x-shader/x-vertex">
11        attribute vec3 position;
12        uniform mat4 uMVMatrix;
13        uniform mat4 uPrMatrix;
14        void main(void){
15          gl_Position = uPrMatrix * uMVMatrix * vec4(position, 1.0);
16          //gl_Position = vec4(position, 1.0);
17        }
18      </script>
19      <!-- create fragment shader -->
20      <script id="fragmentShader" type="x-shader/x-fragment">
21        precision mediump float;
22        void main(void){
23          gl_FragColor = vec4(0.0, 0.0, 1.0, 1.0);
24        }
25      </script>
26      <!-- ここまで追記 -->
27    </head>
28
29    <body>
30      <!-- bodyタグの中にCanvasを配置する -->
31      <canvas id="canvas" width=256 height=256></canvas>
32    </body>
33  </html>
```

Listing 7　Day4_sample4.js

```
1   document.addEventListener('DOMContentLoaded', function () {
2     // HTMLからcanvas要素を取得する
3     const canvas = document.getElementById('canvas');
4
5     // canvas要素からwebglコンテキストを取得する
6     const gl = canvas.getContext('webgl');
7
8     // WebGLコンテキストが取得できたかどうか調べる
9     if (!gl) {
```

```
10      alert('webgl not supported!');
11      return;
12  }
13
14  setTimeout(render, 30);
15  var frame = 0;
16  function render() {
17  // canvasを初期化する色を設定する
18  gl.clearColor(0.0, 0.0, 0.0, 1.0);
19
20  // canvasを初期化する
21  gl.clear(gl.COLOR_BUFFER_BIT);
22  frame++;
23  // ------------ ここから追記 ------------
24  // プログラムオブジェクトを作成する
25  const program = gl.createProgram();
26
27  // シェーダのソースを取得する
28  const vertexShaderSource = document.getElementById('vertexShader').
        textContent;
29  const fragmentShaderSource = document.getElementById('fragmentShader').
        textContent;
30
31  // シェーダをコンパイルして、プログラムオブジェクトにシェーダを割り当て
        る
32  const vertexShader = gl.createShader(gl.VERTEX_SHADER);
33  gl.shaderSource(vertexShader, vertexShaderSource);
34  gl.compileShader(vertexShader);
35  gl.attachShader(program, vertexShader);
36  const fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
37  gl.shaderSource(fragmentShader, fragmentShaderSource);
38  gl.compileShader(fragmentShader);
39  gl.attachShader(program, fragmentShader);
40
41  // シェーダをリンクする
42  gl.linkProgram(program);
```

```
43  // プログラムオブジェクトを有効にする
44  gl.useProgram(program);
45
46  // 3つの頂点の座標を定義する
47  const triangleVertexPosition = [
48    0.0, 0.4, 0.0,
49    -0.4, -0.4, 0.0,
50    0.4, -0.4, 0.0
51  ];
52
53  // 頂点バッファを作成する
54  const triangleVertexBuffer = gl.createBuffer();
55  // 頂点バッファをバインドする
56  gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexBuffer);
57  // 頂点バッファに頂点データをセットする
58  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(triangleVertexPosition
        ), gl.STATIC_DRAW);
59
60  // Positionのロケーションを取得し、バッファを割り当てる
61  const positionLocation = gl.getAttribLocation(program, 'position');
62  gl.enableVertexAttribArray(positionLocation);
63  gl.vertexAttribPointer(positionLocation, 3, gl.FLOAT, false, 0, 0);
64
65  MVMat = glMatrix.mat4.create();
66  glMatrix.mat4.identity(MVMat);
67  glMatrix.mat4.rotateX(MVMat, MVMat, 3.14/180.*frame);
68  glMatrix.mat4.translate(MVMat, MVMat, [0.4, 0.2, -.2]);
69  let eye=glMatrix.vec3.fromValues(0,0,3);
70  let center=glMatrix.vec3.fromValues(0,0,0);
71  let up=glMatrix.vec3.fromValues(0,1,0);
72  var lookAtMatrix = glMatrix.mat4.create();
73  glMatrix.mat4.lookAt(lookAtMatrix,eye,center,up);
74  console.log(lookAtMatrix)
75  glMatrix.mat4.multiply(MVMat,lookAtMatrix,MVMat);
76
77        //uPrLocationにMVMatの値を格納
```

```
78    PrMat = glMatrix.mat4.create();
79    //glMatrix.mat4.identity(PrMat);
80    glMatrix.mat4.perspective(PrMat,.4*Math.PI,1,0.1,3);
81
82
83    //glMatrix.mat4.frustum(PrMat,-.5,.5,-.5,.5,0.05,2);
84    //console.log(PrMat)
85    var uPrLocation = gl.getUniformLocation(program, 'uPrMatrix');
86          //シェーダプログラム中のuPrMatrix変数の場所を受けとる
87    gl.uniformMatrix4fv(uPrLocation, false, PrMat)
88          //uPrLocationにPrMatの値を格納
89    var uMVLocation = gl.getUniformLocation(program, 'uMVMatrix');
90          //シェーダプログラム中のuMVMatrix変数の場所を受けとる
91    gl.uniformMatrix4fv(uMVLocation, false, MVMat)
92
93
94    // 描画する
95    gl.drawArrays(gl.TRIANGLES, 0, 3);
96
97    gl.flush();
98    setTimeout(render, 30);
99    }
100   // ------------ ここまで追記 ------------
101 });
```

Day5

Listing 8　Day5_sample1.html

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>WebGL入門</title>
5      <script type="text/javascript" src="./gl-matrix.js"></script>
6      <script src="./Day5_sample1.js"></script> <!-- main.jsを読み込む -->
7
8      <!-- ここから追記 -->
9      <!-- create vertex shader -->
```

```
    <script id="vertexShader" type="x-shader/x-vertex">
      attribute vec3 position;
          uniform mat4 uMVMatrix;
          uniform mat4 uPrMatrix;
      void main(void){
        gl_Position = uPrMatrix * uMVMatrix * vec4(position, 1.0);
                //gl_Position = vec4(position, 1.0);
      }
    </script>
    <!-- create fragment shader -->
    <script id="fragmentShader" type="x-shader/x-fragment">
      precision mediump float;
      void main(void){
        gl_FragColor = vec4(0.0, 0.0, 1.0, 1.0);
      }
    </script>
    <!-- ここまで追記 -->
  </head>

  <body>
    <!-- bodyタグの中にCanvasを配置する -->
    <canvas id="canvas" width=256 height=256></canvas>
  </body>
</html>
```

Listing 9　Day5_sample1.js

```
document.addEventListener('DOMContentLoaded', function () {
  // HTMLからcanvas要素を取得する
  const canvas = document.getElementById('canvas');
  canvas.width = 512;
  canvas.height = 512;

  // canvas要素からwebglコンテキストを取得する
  const gl = canvas.getContext('webgl');
  // WebGLコンテキストが取得できたかどうか調べる
  if (!gl) {
      alert('webgl not supported!');
```

```
12      return;
13  }
14
15  // ------------ ここから追記 ------------
16  program = glInitShader();
17
18  setTimeout(render, 30);
19  var frame = 0;
20  //uniform変数の場所を取得
21  var uMVLocation = gl.getUniformLocation(program, 'uMVMatrix');
        //シェーダプログラム中のuPrMatrix変数の場所を受けとる
22  var uPrLocation = gl.getUniformLocation(program, 'uPrMatrix');
        //シェーダプログラム中のuMVMatrix変数の場所を受けとる
23
24  MVMat = glMatrix.mat4.create();                 //モデルビュー行列
25  ModelMat = glMatrix.mat4.create();    //モデル行列
26  ViewMat =  glMatrix.mat4.create();    //カメラ行列
27  PrMat = glMatrix.mat4.create();                 //  投影行列
28  let MVMatStack = [];
29
30  let eye=glMatrix.vec3.fromValues(0,5,5);
31  let center=glMatrix.vec3.fromValues(0,0,0);
32  let up=glMatrix.vec3.fromValues(0,1,0);
33  glMatrix.mat4.lookAt(ViewMat,eye,center,up);
34
35  prMat = glMatrix.mat4.create();
36  //glMatrix.mat4.identity(PrMat);
37  glMatrix.mat4.perspective(PrMat,90.*Math.PI/180., canvas.width/canvas.
        height , 0.1, 10);
38  //glMatrix.mat4.frustum(PrMat,-.5,.5,-.5,.5,0.05,2);
39
40  // 6つの頂点の座標を定義する
41  const triangleVertexPos = [
42    0.0, 0.0, 0.0,
43    -0.4, -2.8, 0.0,
44    0.4, -2.8, 0.0,
```

```
45
46      0.0, -0.0, 0.0,
47      -0.4, 2.8, 0.0,
48      0.4, 2.8, 0.0
49
50  ];
51
52  // 頂点バッファを作成する
53  const triangleVertexBuffer = gl.createBuffer();
54  // 頂点バッファをバインドする
55  gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexBuffer);
56  // 頂点バッファに頂点データをセットする
57  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(triangleVertexPos), gl.
        STATIC_DRAW);
58
59  // Positionのロケーションを取得し、バッファを割り当てる
60  const positionLocation = gl.getAttribLocation(program, 'position');
61  gl.enableVertexAttribArray(positionLocation);
62  gl.vertexAttribPointer(positionLocation, 3, gl.FLOAT, false, 0, 0);
63
64
65  render();
66
67        function render(){
68          gl.clearColor(0.0, 0.0, 0.0, 1.0);
69          gl.clear(gl.COLOR_BUFFER_BIT);
70
71    glMatrix.mat4.identity(ModelMat);
72    pushMVMatrix();
73    glMatrix.mat4.translate(ModelMat, ModelMat, [2.0, 0.0, 0.0]);
74    drawPiller();
75    popMVMatrix();
76    glMatrix.mat4.identity(ModelMat);
77    pushMVMatrix();
78    glMatrix.mat4.translate(ModelMat, ModelMat, [-2.0, 0.0, 0.0]);
79    drawPiller();
```

```
80      popMVMatrix();
81    }
82
83    function drawPiller() {
84      for (let i = 0; i < 8; i++) {
85        let localMat = glMatrix.mat4.clone(ModelMat);
86        glMatrix.mat4.rotateY(localMat, localMat, (i * 360 / 8) * Math.PI /
              180);
87        glMatrix.mat4.translate(localMat, localMat, [0.0, 0.0, 1]);
88        glMatrix.mat4.multiply(MVMat,ViewMat,localMat);
89
90        gl.uniformMatrix4fv(uMVLocation, false, MVMat)
91        gl.uniformMatrix4fv(uPrLocation, false, PrMat)    //uPrLocationに
              PrMatの値を格納
92
93        // 描画する
94        gl.drawArrays(gl.TRIANGLES, 0, 6);
95      }
96           gl.flush();
97    }
98
99    function pushMVMatrix() {
100     var copyToPush = glMatrix.mat4.clone(MVMat);
101     MVMatStack.push(copyToPush);
102   }
103
104   function popMVMatrix() {
105     if (MVMatStack.length == 0) {
106       throw "Error popMVMatrix()--stack was empty";
107     }
108     MVMat = MVMatStack.pop();
109   }
110
111         function glInitShader(){
112           // シェーダのソースを取得する
113           const vertexShaderSource = document.getElementById('
```

```
            vertexShader').textContent;
114         const fragmentShaderSource = document.getElementById('
            fragmentShader').textContent;
115
116         // シェーダをコンパイルして、プログラムオブジェクトにシェーダを
            割り当てる
117         const vertexShader = gl.createShader(gl.VERTEX_SHADER);
118         gl.shaderSource(vertexShader, vertexShaderSource);
119         gl.compileShader(vertexShader);
120         const fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
121         gl.shaderSource(fragmentShader, fragmentShaderSource);
122         gl.compileShader(fragmentShader);
123
124         // プログラムオブジェクトにシェーダを作成する
125         const program = gl.createProgram();
126         gl.attachShader(program, vertexShader);
127         gl.attachShader(program, fragmentShader);
128         gl.linkProgram(program);
129
130         // プログラムオブジェクトを有効にする
131         gl.useProgram(program);
132         return program
133       }
134
135 });
```