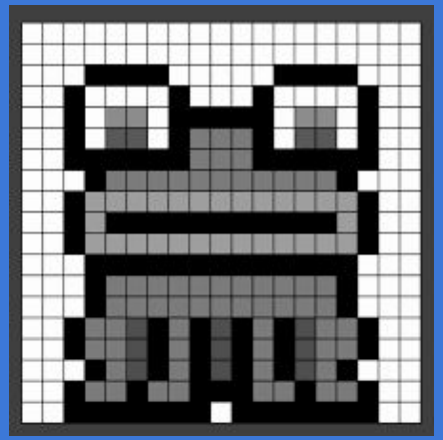
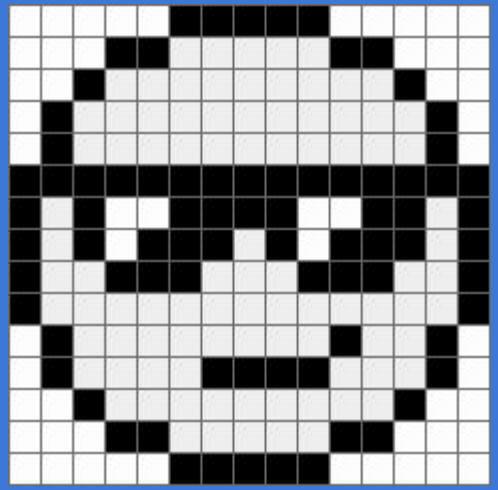
# SEE++



Featuring the all power building block:

.



By Adar Tulloch, Vishrut Tiwari, Winston Zhang, and Jack LaVelle

# Motivation

```
void setup() {  // Defines the display window parameters.
  size(700, 400);
  smooth();
}

void draw() {    // Defines an action taken if a mouse-button is p
  if (mousePressed) {
    fill(0);
  } else {
    fill(255);
  }
  ellipse(mouseX, mouseY, 80, 80);
}
```

We wanted to build a:

1. Simple C/Java like language
2. Give the user the best support to build art
3. Implement a primitive type that everyone can use and build on

# Language Features

- C-like syntax and behavior (variable assignments, operations, scope-rules)
- Advanced types like Pixels, Points, Circles and an adjustable Canvas for the user to render their drawings

# Sample programs

```
int main() {
  Canvas(5.0, 6.0);
  return 0;
}
```

```
int main(){
  Pixel p =
Pixel(Point(10.0, 10.0));
  float test = p.ep1.x;
  printf(test);
  return 0;
}
```

```
int main(){
    CanvasCircle can = CanvasCircle(1000.0,1000.0);
    Point pt = Point(500.0, 500.0);
    Circle px = Circle(pt, 100.0);
    Point ppt = Point(10.0, 10.0);
    Pixel pix = Pixel(ppt);
    Canvas can1 = Canvas(1000.0, 1000.0);
    can1 -> append() pix;
    can -> append().circle px;
    drawcircle(can, "result.svg");
    return 0;
}
```

# How did we achieve this?

```
let ptstruct_t = L.struct_type context [| float_t ; float_t |] in
let pstruct_t  = L.struct_type context [| ptstruct_t |] in
let cstruct_t  = L.struct_type context [| ptstruct_t; float_t |] in

let canvasnode_t = L.named_struct_type context "canvasnode" in
ignore(L.struct_set_body canvasnode_t [| L.pointer_type (canvasnode_t) ;
    (L.pointer_type pstruct_t) |] false);

let canvascirclenode_t = L.named_struct_type context "canvascirclenode" in
ignore(L.struct_set_body canvascirclenode_t [| L.pointer_type (canvascirclenode_t) ;
    (L.pointer_type cstruct_t) |] false);

let canvas_t = L.struct_type context [| float_t ; float_t ;
    (L.pointer_type canvasnode_t) |]
in

let canvascircle_t = L.struct_type context [| float_t ; float_t ;
    (L.pointer_type canvascirclenode_t) |]
in
```

```
struct point Point(double x, double y)
{
        struct point pt;
        pt.x = x;
        pt.y = y;
        return pt;
}

struct pixel Pixel(struct point ep1)
{
        struct pixel cv;
        cv.ep1 = ep1;
        return cv;
}

struct canvas Canvas(double x, double y)
{
        struct canvas c;

        c.x = x;
        c.y = y;
        c.first = 0;
        return c;
}
```

# How did we achieve this?

```
| SBinop((A.Canvas, _) as can, op, crv) ->
    let (_,can_s) = (match (snd can) with
        SId s -> (expr builder locals can, s)
        |_-> raise(Failure "improper usage of shoehorn - canvas"))
    and (_,px_s) = (match (snd crv) with
        SId s -> (expr builder locals crv,s)
        |_->raise(Failure "improper usage of shoehorn - pixel")) in
    (match op with
        A.Shoehorn   ->
          (* construct new node, add it to front of list *)
          let newnode = L.build_alloca canvasnode_t "newnode" builder in
          let next_node_ptr = L.build_struct_gep newnode 0 "new_pixel" builder in
          ignore(L.build_store (L.const_null (L.pointer_type canvasnode_t)) next_node_ptr builder);
          let pixel_ptr = L.build_struct_gep newnode 1 "pixel" builder in
          let pxlv = lookup px_s locals in
          ignore(L.build_store pxlv pixel_ptr builder);
          let canlv = lookup can_s locals in
          let headptr = L.build_struct_gep canlv 2 "head" builder in
          let oldhead = L.build_load headptr "oldptr" builder in
          ignore(L.build_store oldhead next_node_ptr builder);
          ignore(L.build_store newnode headptr builder); canlv
      | _ -> raise (Failure ("improper usage of shoehorn: -> append() " ^
            (string_of_sexpr can) ^ " and " ^ (string_of_sexpr crv))))
```

```
int draw(struct canvas canv, char *filename)
{
        svg* psvg;
        psvg = svg_create(canv.x, canv.y);

        if (psvg == NULL) {
                fprintf(stderr, "could not store SVG meta data, malloc returned null");
                exit(1);
        }
        else{
            read_canvas(canv.first, psvg);

            svg_finalize(psvg);
            svg_save(psvg, filename);
            svg_free(psvg);
        }

        return 0;
}
```

# How did we achieve this?

```
| SBinop((A.CanvasCircle,_) as can, op, crl) ->
    let (_,can_s) = (match (snd can) with
        SId s -> (expr builder locals can, s)
        |_-> raise(Failure "improper usage of shoehorn - canvas"))
    and (_,cl_s) = (match (snd crl) with
        SId s -> (expr builder locals crl,s)
        |_->raise(Failure "improper usage of shoehorn - circle: -> append().circle")) in
    (match op with
        A.ShoehornCircle   ->
          (* construct new node, add it to front of list *)
          let newnode = L.build_alloca canvascirclenode_t "newnode" builder in
          let next_node_ptr = L.build_struct_gep newnode 0 "new_circle" builder in
          ignore(L.build_store (L.const_null (L.pointer_type canvascirclenode_t)) next_node_ptr builder)
          let circle_ptr = L.build_struct_gep newnode 1 "circle" builder in
          let pxlv = lookup cl_s locals in
          ignore(L.build_store pxlv circle_ptr builder);
          let canlv = lookup can_s locals in
          let headptr = L.build_struct_gep canlv 2 "head" builder in
          let oldhead = L.build_load headptr "oldptr" builder in
          ignore(L.build_store oldhead next_node_ptr builder);
          ignore(L.build_store newnode headptr builder); canlv
      | _ -> raise (Failure ("improper usage of shoehornCircle with " ^
            (string_of_sexpr can) ^ " and " ^ (string_of_sexpr crl))))
```

```
int drawcircle(struct canvascircle cancirv, char *filename){
        svg* psvg;
        psvg = svg_create(cancirv.x, cancirv.y);
        if (psvg == NULL) {
                fprintf(stderr, "could not store SVG meta data, malloc returned null");
                exit(1);
        }
        else{
            read_canvascircle(cancirv.first, psvg);

            svg_finalize(psvg);
            svg_save(psvg, filename);
            svg_free(psvg);
        }

        return 0;
}
```

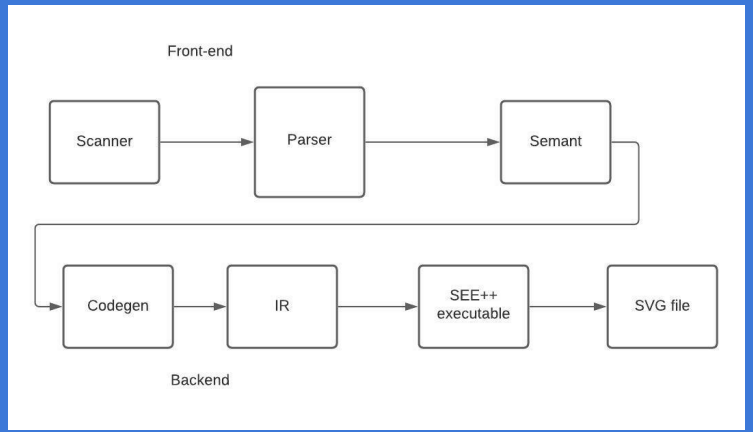
```
struct circle Circle(struct point ep1, double r)
{
        printf("Entered Circle.\n");
        struct circle crl;
        crl.r = r;
        crl.x = ep1.x;
        crl.y = ep1.y;
        crl.ep1 = ep1;
        return crl;

}

struct canvascircle CanvasCircle(double x, double y)
{
        struct canvascircle c;
        c.x = x;
        c.y = y;
        c.first = 0;
        printf("Finished CanvasCircle.\n");
        return c;
}
```

# How did we achieve this?

```
generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe ${basename}.out" &&
Run "$SEEPP" "$1" ">" "${basename}.ll" &&
Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">" "${basename}.s" &&
Run "$CC" "-o" "${basename}.exe" "${basename}.s" "printbig.o" "draw.o" "drawcircle.o" "svg.o" &&
Run "./${basename}.exe" > "${basename}.out" &&
Compare ${basename}.out ${reffile}.out ${basename}.diff
```



# The potential

```
def line(point1, point2, f, canvas, step):
    if point1[0] < point2[0]:
        slope = (point2[1] - point1[1])/(point2[0] - point1[0])
        max = point2[0]
        x = point1[0]
        y = point1[0]
    else:
        slope = (point1[1] - point2[1])/(point1[0] - point2[0])
        max = point1[0]
        x = point2[0]
        y = point2[0]

    if slope == 0:
        b = point2[1]
    else:
        b = point2[1]/(slope*point2[0])



    while x <= max:
        if not (x,y) in pixels:
            pixels.append((x,y))
            px = "Point pt" + str(abs(int(x))) + str(abs(int(y)))+" = Point(" + str(abs(int(x))) + ".0 , " + str(abs(int(y))) + ".0);"
            pixel = "Pixel px"+ str(abs(int(x))) + str(abs(int(y)))+ " = Pixel("+"pt" + str(abs(int(x))) + str(abs(int(y)))+");"
            shoehorn = canvas + " -> append() px"+ str(abs(int(x))) + str(abs(int(y))) + ";"
            f.write(px+"\n"+pixel+"\n"+shoehorn+"\n")
        x += step
        y = x*slope + b
```

# Scope for further improvement

We wanted to build a:

1. Make one Shoe Horn operation
2. Add built-in shapes like squares, lines, triangles etc.
3. Compose multiple canvases onto each other to create complex images
4. Add color

# Demo Time