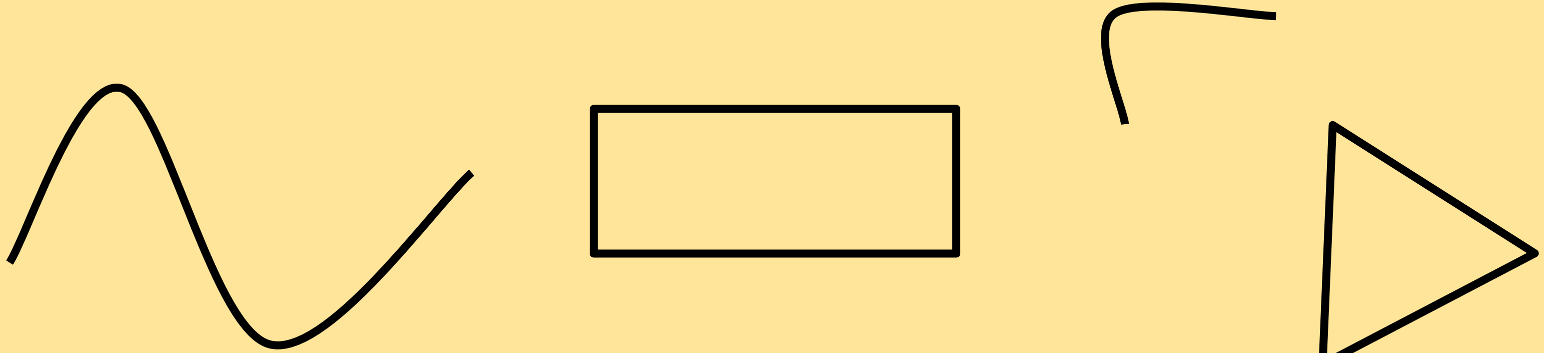
SSOL

Simple Shape Oriented Language

Jeevan Farias (Language Designer)
Daniel Mesko (System Architect)
Madeleine Tipp (Manager/Test Engineer)



Motivation

- Algorithmically create shapes and render in SVG format
- Concise syntax to describe shapes and ‘drawing boards’
- draw() function for writing of files
- General purpose language, with C-like syntax

Features

- All MicroC operators + dynamic declaration, arrays (access, literals, assignment), strings
- Built-in complex types and built in functions
- Linked with a C library that reads the SSOL types and generates SVG files representing them

Complex Types

`Point(float x, float y);`

- Takes two `float` arguments to define relative position on the `Canvas`
- Defined as an LLVM `struct_type`
    - `let ptstruct_t = L.struct_type context [| float_t ; float_t |]`

`Curve(Point a, Point b, Point c2, Point c2);`

- Takes four `Point` objects, two to anchor and two to define the “curve” attribute (cubic Bezier curve)
- Defined as an LLVM `struct_type`
    - `let cstruct_t  = L.struct_type context [| ptstruct_t ; ptstruct_t ; ptstruct_t ; ptstruct_t|]`

Complex Types

```
Canvas(float x, float y);
```

- Takes two float arguments to define relative image dimensions
- Holds a pointer to the first “canvas node” in a linked-list

  -
    ```
    let canvas_t   = L.struct_type context
      [| float_t ; float_t ; L.pointer_type canvasnode_t |]
    ```
- Each canvas node points to one curve, and the next node

  -
    ```
    let canvasnode_t = L.named_struct_type context "canvasnode" in
    ```

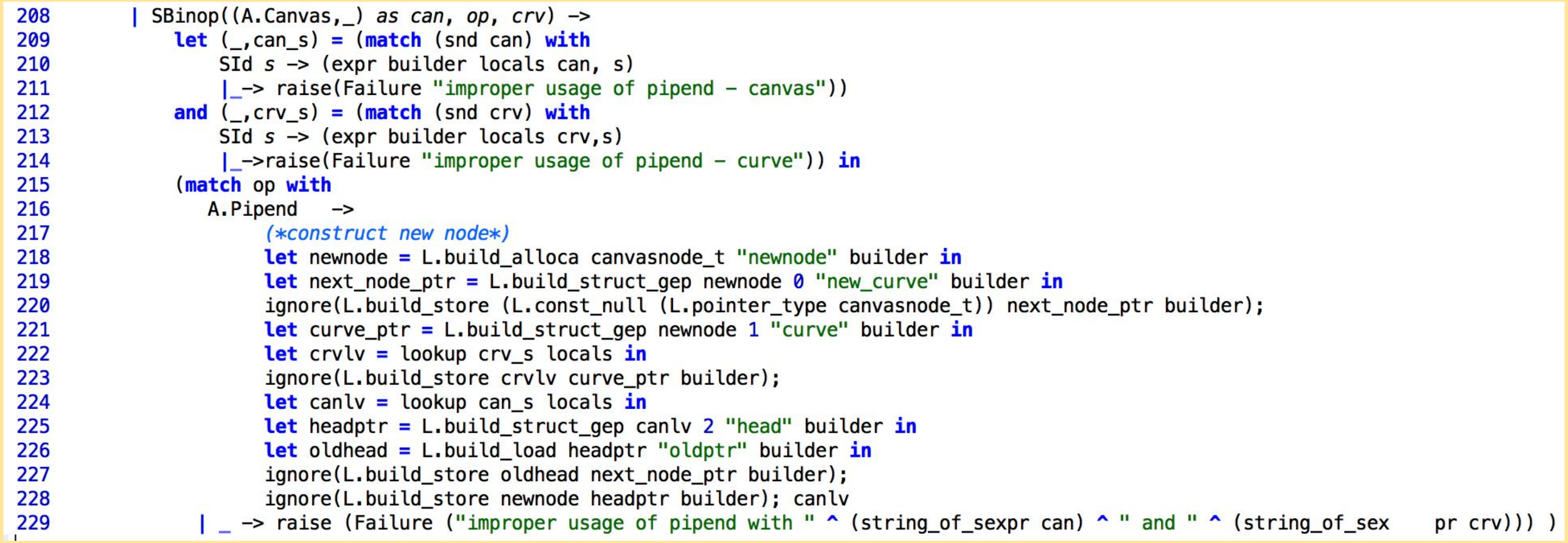
  -
    ```
    L.struct_set_body canvasnode_t [| L.pointer_type (canvasnode_t) ;
    (L.pointer_type cstruct_t) |] false);
    ```

  -
    ```
    let canvas_t = L.struct_type context [| float_t ; float_t ;
    L.pointer_type canvasnode_t |]
    ```

Special Operators

`Canvas |= Curve`

- Pipend, denoted `|=` , is the operator used to append elements to the canvas



```
| 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 pipend - canvas"))
    and (_,crv_s) = (match (snd crv) with
        SId s -> (expr builder locals crv,s)
        |_->raise(Failure "improper usage of pipend - curve")) in
    (match op with
       A.Pipend   ->
            (*construct new node*)
            let newnode = L.build_alloca canvasnode_t "newnode" builder in
            let next_node_ptr = L.build_struct_gep newnode 0 "new_curve" builder in
            ignore(L.build_store (L.const_null (L.pointer_type canvasnode_t)) next_node_ptr builder);
            let curve_ptr = L.build_struct_gep newnode 1 "curve" builder in
            let crvlv = lookup crv_s locals in
            ignore(L.build_store crvlv curve_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 pipend with " ^ (string_of_sexpr can) ^ " and " ^ (string_of_sex    pr crv))) )
```

Challenges / Next Steps

- Structs / Field access
  - Semantic checking
  - member_map_of_type
  - mem_to_ind
- Constructors
  - function calls - variables inside constructor calls
  - Written in C - structs mirror the SSOL types

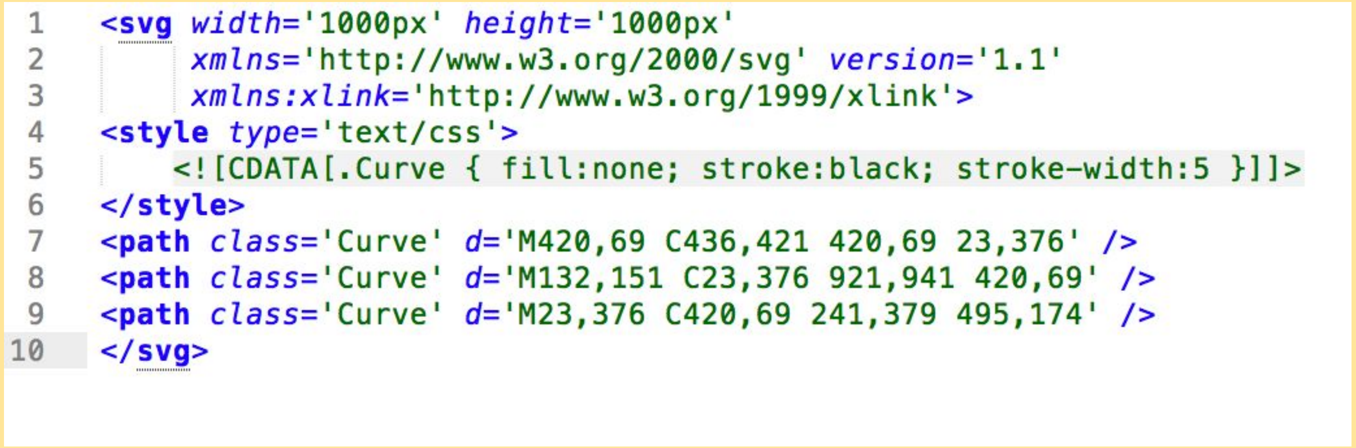
Special Functions

```
Draw(Canvas c, String file_name);
```

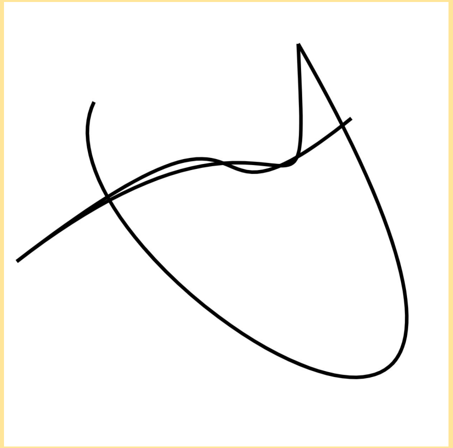
- The `draw()` function passes the linked list of `Curves` stored in the `Canvas` to C functions which produce the SVG file
- Most of the original source code was taken from http://www.code-in-c.com/writing-svg-library-c/

How we are producing svg files

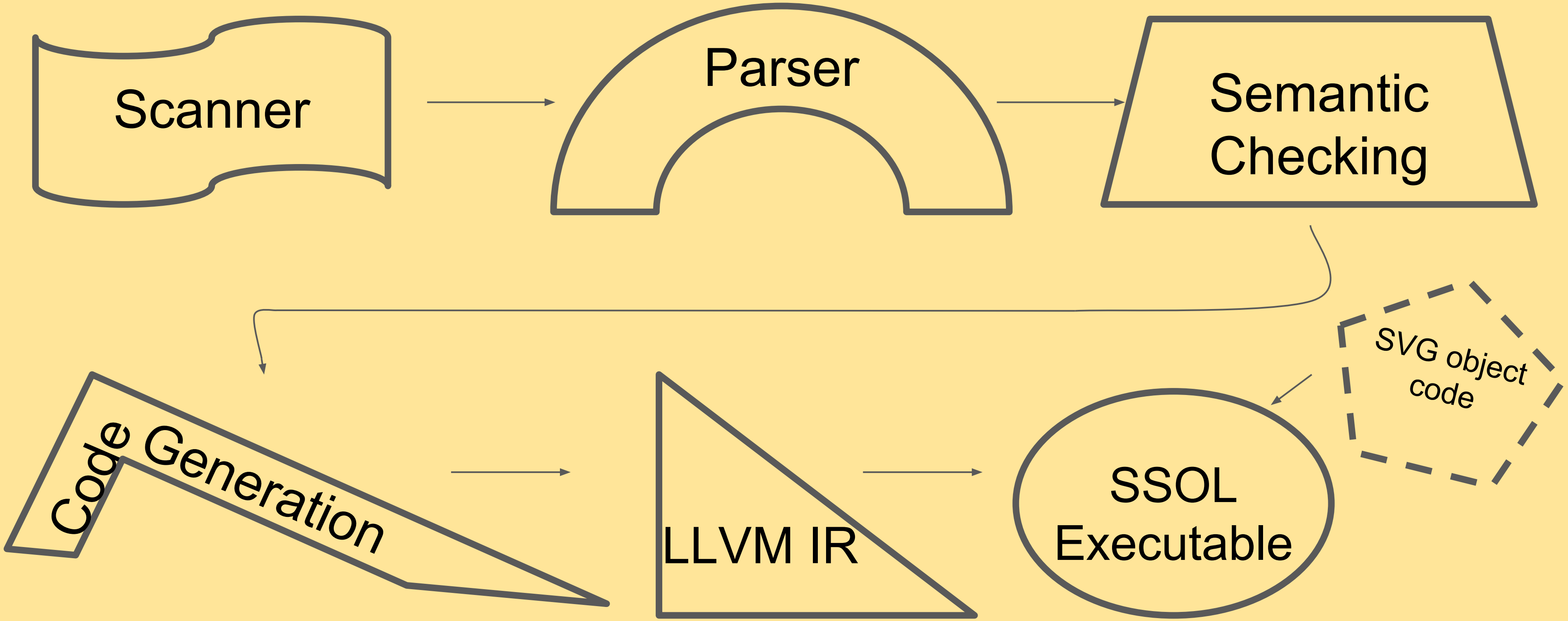
- SVG struct defined in svg.h
- SVG syntax similar to XML



```
<svg width='1000px' height='1000px'
     xmlns='http://www.w3.org/2000/svg' version='1.1'
     xmlns:xlink='http://www.w3.org/1999/xlink'>
<style type='text/css'>
    <![CDATA[.Curve { fill:none; stroke:black; stroke-width:5 }]]>
</style>
<path class='Curve' d='M420,69 C436,421 420,69 23,376' />
<path class='Curve' d='M132,151 C23,376 921,941 420,69' />
<path class='Curve' d='M23,376 C420,69 241,379 495,174' />
</svg>
```



Architecture



Demo

```
float fib(float x)
{
  if (x < 2.0) return 1.0;
  return fib(x-1.0) + fib(x-2.0);
}
```

```
Curve makeCurve(float rad, Point init,float xd, float yd ){
        float fx;
        float fy;
        float sx;
        float sy;

        sx = init.x;
        sy = init.y;

        fx  = init.x;
        fx = fx + (rad *xd);
        fy = init.y;
        fy = fy + (rad *yd);

        Point ep2;
        ep2.x = fx;
        ep2.y = fy;

        Point cp1;
        Point cp2;

        if (xd == yd){
            float t;

            cp1.x = sx;
            t =  sx + (rad*0.65* xd);
            cp2.x = t;

            t =  sy + (rad*0.65* yd);
            cp1.y = t;
            cp2.y = fy;

        }else{
            float t;
            cp1.y = sy;
            t =  sy + (rad*0.65* yd);
            cp2.y = sy;

            t =  sx + (rad*0.65* xd);
            cp1.x = t;
            cp2.x = fx;

        }

    Curve c = Curve(init,ep2, cp1, cp2);

    return c;
}
```

```
int main()
{
    Point p = Point(1.0,1.0);
    float i;
    Canvas c = Canvas(750.0,750.0);
    Point start = Point(400.0,265.0);
  float ydir = 1.0;
    float xdir = -1.0;
    int tmp = 1;
    for (i=0.0; i<10.0; i=i+1.0){

        float j = fib(i)*6.0;
        Curve crv = makeCurve(j,start,xdir,ydir);
        if (tmp>0){
            xdir = xdir *-1.0;
        } else {
            ydir = ydir *-1.0;
        }
        tmp = tmp *-1;
        start = crv.ep2;
        c |= crv;
    }

    draw(c,"fib_spiral.svg");

    return 0;
}
```