galois

Exercise 1:

point.bc:

```
clang -g -O0 -c -emit-llvm point.c -o point.bc
```

point1.saw:

```
let fresh_point_readonly name = do {
   p_ptr <- llvm_alloc_readonly (llvm_struct "struct.point");
   p_x <- llvm_fresh_var (str_concat name ".x") (llvm_int 32);
   p_y <- llvm_fresh_var (str_concat name ".y") (llvm_int 32);
   llvm_points_to p_ptr (llvm_struct_value [ llvm_term p_x, llvm_term p_y]);
   let p = {{ { x = p_x, y = p_y } }};  // type Point in Cryptol
   return (p, p_ptr);
};

let point_eq_setup = do {
   (p1, p1_ptr) <- fresh_point_readonly "p1";
   (p2, p2_ptr) <- fresh_point_readonly "p2";

   llvm_execute_func [p1_ptr, p2_ptr];

   // p1 == p2 can't be done yet since SAW doesn't yet support translating
   // Cryptol's bit type(s) into crucible-llvm's type system. This function
   // needs a [1] response.
   llvm_return (llvm_term {{ [p1 == p2] }});
};

let main : TopLevel () = do {
   m <- llvm_load_module "point.bc";
   point_eq_ov <- llvm_verify m "point_eq" [] true point_eq_setup z3;
   print "Done!";
};
```

running point1.saw:

```
saw point1.saw
[18:32:11.422] Loading file ".../point1.saw"
[18:32:11.577] Verifying point_eq ...
[18:32:11.594] Simulating point_eq ...
[18:32:11.602] Checking proof obligations point_eq ...
[18:32:11.619] Proof succeeded! point_eq
[18:32:11.620] Done!
```

Exercise 2:

point.bc:

```
clang -g -O0 -emit-llvm -c point.c -o point.bc
```

point2.saw:

```
// returns a pointer to a Point object, p is the Point object
let alloc_assign_point p = do {
   p_ptr <- llvm_alloc (llvm_struct "struct.point");
   llvm_points_to p_ptr (llvm_struct_value
                           [ llvm_term {{ p.x }}, llvm_term {{ p.y }}]);
   return p_ptr;
};

let point_new_setup = do {
   p_x <- llvm_fresh_var "p_x" (llvm_int 32);
   p_y <- llvm_fresh_var "p_y" (llvm_int 32);

   llvm_execute_func [ llvm_term p_x, llvm_term p_y ];

   ret_ptr <- alloc_assign_point {{ { x = p_x, y = p_y } }};
   llvm_return ret_ptr;
};

let main : TopLevel () = do {
   m <- llvm_load_module "point.bc";
   point_new_ov <- llvm_verify m "point_new" [] true point_new_setup z3;
   print "Done!";
};
```

running point2.saw:

```
saw point2.saw
[18:42:21.398] Loading file ".../point2.saw"
[18:42:21.550] Verifying point_new ...
[18:42:21.566] Simulating point_new ...
[18:42:21.570] Checking proof obligations point_new ...
[18:42:21.570] Proof succeeded! point_new
[18:42:21.570] Done!
```

Exercise 3:

clang:

```
clang -g -O0 -emit-llvm -c zero.c -o zero.bc
```

point3.saw:

```
let fresh_point_readonly name = do {
   p_ptr <- llvm_alloc_readonly (llvm_struct "struct.point");
   p_x <- llvm_fresh_var (str_concat name ".x") (llvm_int 32);
   p_y <- llvm_fresh_var (str_concat name ".y") (llvm_int 32);
   llvm_points_to p_ptr (llvm_struct_value [ llvm_term p_x, llvm_term p_y]);
   let p = {{ { x = p_x, y = p_y } }};
   return (p, p_ptr);
};
```

```
let alloc_assign_point p = do {
   p_ptr <- llvm_alloc (llvm_struct "struct.point");
   llvm_points_to p_ptr (llvm_struct_value
                           [ llvm_term {{ p.x }}, llvm_term {{ p.y }}]);
   return p_ptr;
};

let point_copy_spec = do {
   (p, p_ptr) <- fresh_point_readonly "p";

   llvm_execute_func [p_ptr];

   ret_ptr <- alloc_assign_point p;
   llvm_return ret_ptr;
};

let main : TopLevel () = do {
   m <- llvm_load_module "point.bc";
   point_copy_ov <- llvm_verify m "point_copy" [] true point_copy_spec z3;
   print "Done!";
};
```

running point3.saw:

```
[18:50:56.858] Loading file ".../point3.saw"
[18:50:57.010] Verifying point_copy ...
[18:50:57.028] Simulating point_copy ...
[18:50:57.032] Checking proof obligations point_copy ...
[18:50:57.032] Proof succeeded! point_copy
[18:50:57.032] Done!
```

Exercise 4:

clang:

```
clang -g -O0 -emit-llvm -c point.c -o point.bc
```

point4.saw:

```
import "Point.cry";

let fresh_point_readonly name = do {
   p_ptr <- llvm_alloc_readonly (llvm_struct "struct.point");
   p_x <- llvm_fresh_var (str_concat name ".x") (llvm_int 32);
   p_y <- llvm_fresh_var (str_concat name ".y") (llvm_int 32);
   llvm_points_to p_ptr (llvm_struct_value [ llvm_term p_x, llvm_term p_y]);
   let p = {{ { x = p_x, y = p_y } }};
   return (p, p_ptr);
};

let alloc_assign_point p = do {
   p_ptr <- llvm_alloc (llvm_struct "struct.point");
   llvm_points_to p_ptr (llvm_struct_value
                           [ llvm_term {{ p.x }}, llvm_term {{ p.y }}]);
   return p_ptr;
};
```

```
let point_add_spec = do {
   let zero_term = llvm_term {{ 0 : [32] }};
   llvm_alloc_global "ZERO";
   llvm_points_to (llvm_global "ZERO")
                  (llvm_struct_value [zero_term, zero_term]);

   (p1, p1_ptr) <- fresh_point_readonly "p1";
   (p2, p2_ptr) <- fresh_point_readonly "p2";

   llvm_execute_func [p1_ptr, p2_ptr];

   res_ptr <- alloc_assign_point {{ point_add p1 p2 }};
   llvm_return res_ptr;
};

let main : TopLevel () = do {
   m <- llvm_load_module "point.bc";
   llvm_verify m "point_add" [] true point_add_spec z3;
   print "Done!";
};
```

running point4.saw:

```
saw point4.saw
[18:57:07.170] Loading file ".../point4.saw"
[18:57:07.520] Verifying point_add ...
[18:57:07.540] Simulating point_add ...
[18:57:07.564] Checking proof obligations point_add ...
[18:57:07.651] Proof succeeded! point_add
[18:57:07.651] Done!
```