PhysEx

A physical simulation language

Joshua Nuez Justin Pugliese Steven Ulahannan David
Pu

Motivation

Most of the programming languages can solve mathematical questions and simulate the process of finding the solutions pretty naturally.

Simulating a geometric series (for loop)

```
int powerOf2 (int exponent) {
    int x = 1;
    for (int i = 0; i < exponent; ++ i) {
        printf("%d ",x *= 2);
    }
    printf("\n");
    return x;
}
```

Simulating fibonacci series (recursion)

```
int fib(int n) {
    return n < 3 ? 1 : fib(n - 1) + fib(n - 2);
}
```

In both examples, the process goes like:

Define base cases → Define the "update" → Repeat & check for termination → Complete

Motivation and Overview

But what about simple physics?

We want to create a language that can naturally simulate simple physical forces. The process should be similar to what we have been doing for math.

Set initial conditions → Define interactions (forces) → Set termination condition (time)

Wrapped in fundamental components (Blobs)

→ Simulate

System function: Stimulus

Collaboration

GitHub

slack

Google Drive

Google Cloud Platform

Weekly meetings after class

Josh - Project Manager

Steve - Test Lead

David - System Architecture

Justin - Language Guru

Syntax

Comment:

```
// I am a comment
```

Operators:

```
+ - ++ -- * / % ^ == != < > <= >= && || !
= += -= *= /= %=
```

Variables:

```
longDouble a = 0;
int arr[10];
```

Control Flow:

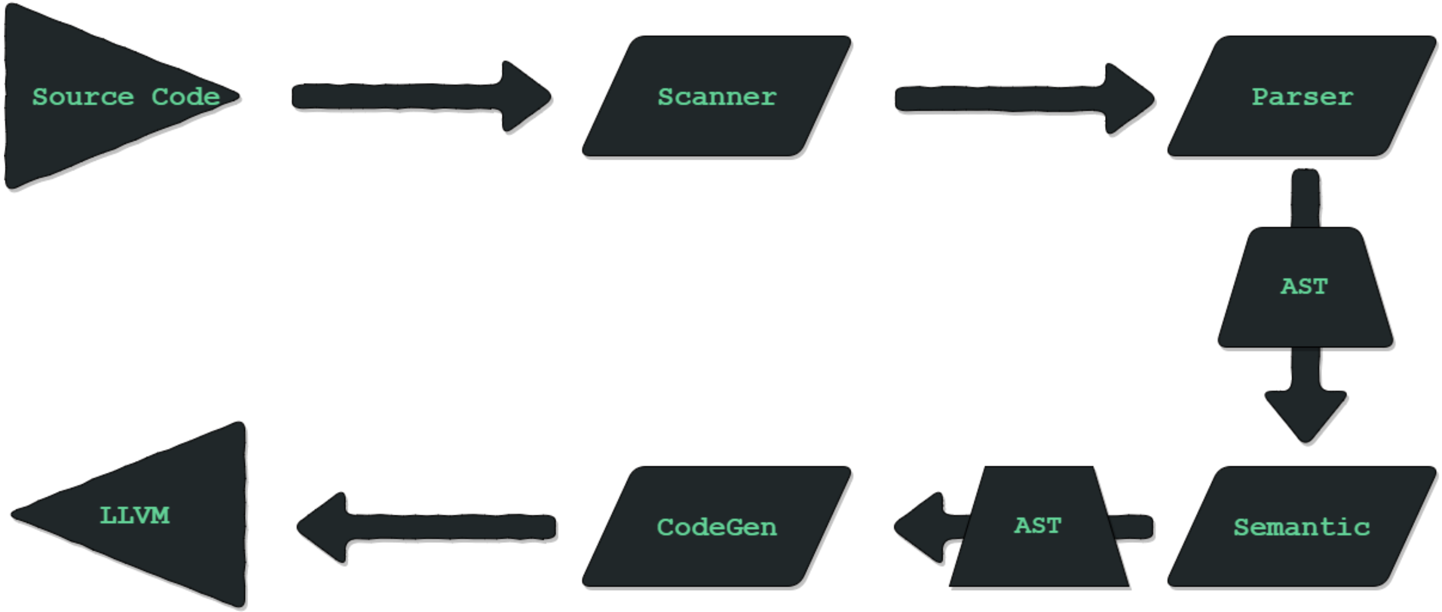
```
if (condition) { … }
if (condition) { … } else { … }
while (condition) { … }
for ( … ; … ; … ) {}
```

Function Declaration:

```
[type] func functionName (parameter1, parameter2, … )
{ … }
[type] func functionName (parameter1, parameter2, … )
{
        …
        return expr;
}
```

Stimulus: (intention)

```
stimulus stimulusName (optDelay, [optBlobs]);
```



Architecture

Testing Strategy

Automated Test Suite

```
if [ $# -ge 1 ]
then
    files=$@
else
    files="tests/test-*.x"
fi

for file in $files
do
    case $file in
    *test-*)
        Check $file 2>> $globallog
        ;;
    *fail-*)
        CheckFail $file 2>> $globallog
        ;;
    *)
        echo "unknown file type $file"
        globalerror=1
        ;;
    esac
done

exit $globalerror
```

- test-func1.out
- test-func1.x
- test-func2.out
- test-func2.x
- test-func3.out
- test-func3.x
- test-func4.out
- test-func4.x
- test-gcd.out
- test-gcd.x
- test-if1.out
- test-if1.x
- test-if2.out
- test-if2.x
- test-if3.out
- test-if3.x
- test-if4.out
- test-if4.x
- test-if5.out
- test-if5.x
- test-if6.out

- test-array1.out
- test-array1.x
- test-array2.out
- test-array2.x
- test-array3.out
- test-array3.x
- test-assign1.out
- test-assign1.x
- test-assign2.out
- test-assign2.x
- test-assign3.out
- test-assign3.x
- test-bool1.out
- test-bool1.x
- test-equality1.out
- test-equality1.x
- test-factorial.out
- test-factorial.x
- test-for1.out
- test-for1.x

Example Source Code

```
int func gcd (int x, int y)
{
  while (x != y) {
    if (x > y) {
      x = x - y;
    }
    else {
      y = y - x;
    }
  }
  return x;
}

void func simulation ()
{
  printi( gcd(8,12) );
}
```

```
int func fact (int n)
{
  int i;

  if (n <= 1) {
    return 1;
  }
  else {
    i = n * fact(n - 1);
  }

  return i;
}

void func simulation ()
{
  printi( fact(4) );
}
```

```
int i;
int j;
int k;

void func simulation ()
{
  i = 0;
  k = 0;
  while(i < 3) {

    for (j = 0; j < 3; j = j + 1) {

      while (k < 3) {
        printi(k);
        k = k + 1;
      }

      printi(k);
      k = 0;
    }

    printi(k);
    i = i+1;
  }
}
```

Demo

```
int time;
int accel;
int init_y;

int func distance() {
  int curr_y;
  curr_y  = (accel*time*time)/2 + init_y;
  if (curr_y > 0)
    return curr_y;
  print("Splat... ");
  return 0;
}

void func simulation() {
  time = 0;
  accel = -10;
  init_y = 100;

  start(6) {
    sleep(1);
    printi(distance());
    time = time + 1;
  }
}
```

Llvm Output

```
int time;
int accel;
int init_y;

int func distance() {
  int curr_y;
  curr_y  = (accel*time*time)/2
+ init_y;
  if (curr_y > 0)
    return curr_y;
  print("Splat... ");
  return 0;
}

void func simulation() {
  time = 0;
  accel = -10;
  init_y = 100;

  start(6) {
    sleep(1);
    printi(distance());
    time = time + 1;
  }
}
```

```
; ModuleID = 'PhysEx'

@init_y = global i32 0
@accel = global i32 0
@time = global i32 0
@fmt = private unnamed_addr constant [3 x i8] c"%s\00"
@fmt1 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt2 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmt3 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@0 = private unnamed_addr constant [10 x i8] c"Splat... \00"

declare i32 @printf(i8*, ...)

declare i8* @calloc(i32, i32)

declare i32 @sleep(i32)

declare i64 @clock()

define void @main() {
entry:
  store i32 0, i32* @time
  store i32 -10, i32* @accel
  store i32 100, i32* @init_y
  %sleep = call i32 @sleep(i32 1)
  %distance_result = call i32 @distance()
  %printf = call i32 (i8*, ...)* @printf(i8* getelementptr
inbounds ([4 x i8]* @fmt1, i32 0, i32 0), i32
%distance_result)
  %time = load i32* @time
  %tmp = add i32 %time, 1
  store i32 %tmp, i32* @time
  %sleep1 = call i32 @sleep(i32 1)
  %distance_result2 = call i32 @distance()
  %printf3 = call i32 (i8*, ...)* @printf(i8* getelementptr
inbounds ([4 x i8]* @fmt1, i32 0, i32 0), i32
%distance_result2)
  %time4 = load i32* @time
  %tmp5 = add i32 %time4, 1
  store i32 %tmp5, i32* @time
  %sleep6 = call i32 @sleep(i32 1)
  %distance_result7 = call i32 @distance()
  %printf8 = call i32 (i8*, ...)* @printf(i8* getelementptr
inbounds ([4 x i8]* @fmt1, i32 0, i32 0), i32
%distance_result7)
  %time9 = load i32* @time
  %tmp10 = add i32 %time9, 1
  store i32 %tmp10, i32* @time
  %sleep11 = call i32 @sleep(i32 1)
  %distance_result12 = call i32 @distance()
```

```
@fmt1, i32 0, i32 0), i32 %distance_result12)
  %time14 = load i32* @time
  %tmp15 = add i32 %time14, 1
  store i32 %tmp15, i32* @time
  %sleep16 = call i32 @sleep(i32 1)
  %distance_result17 = call i32 @distance()
  %printf18 = call i32 (i8*, ...)* @printf(i8* getelementptr
inbounds ([4 x i8]* @fmt1, i32 0, i32 0), i32
%distance_result17)
  %time19 = load i32* @time
  %tmp20 = add i32 %time19, 1
  store i32 %tmp20, i32* @time
  %sleep21 = call i32 @sleep(i32 1)
  %distance_result22 = call i32 @distance()
  %printf23 = call i32 (i8*, ...)* @printf(i8* getelementptr
inbounds ([4 x i8]* @fmt1, i32 0, i32 0), i32
%distance_result22)
  %time24 = load i32* @time
  %tmp25 = add i32 %time24, 1
  store i32 %tmp25, i32* @time
  ret void
}
define i32 @distance() {
entry:
  %curr_y = alloca i32
  %accel = load i32* @accel
  %time = load i32* @time
  %tmp = mul i32 %accel, %time
  %time1 = load i32* @time
  %tmp2 = mul i32 %tmp, %time1
  %tmp3 = sdiv i32 %tmp2, 2
  %init_y = load i32* @init_y
  %tmp4 = add i32 %tmp3, %init_y
  store i32 %tmp4, i32* %curr_y
  %curr_y5 = load i32* %curr_y
  %tmp6 = icmp sgt i32 %curr_y5, 0
  br i1 %tmp6, label %then, label %else
merge:                                          ; preds =
%else
  %printf = call i32 (i8*, ...)* @printf(i8* getelementptr
inbounds ([3 x i8]* @fmt2, i32 0, i32 0), i8* getelementptr
inbounds ([10 x i8]* @0, i32 0, i32 0))
  ret i32 0
then:                                           ; preds =
%entry
  %curr_y7 = load i32* %curr_y
  ret i32 %curr_y7
else:                                           ; preds =
%entry
  br label %merge
}
```

Thank you.