PhysEx

Final Report

December 20, 2016

Project Manager: Joshua Nuez (jn2548)

System Architect: David Pu (sp3396)

Tester: Steven Ulahannan (su2206)

Language Guru: Justin Pugliese (jp3571)

Table of Contents

1. Introduction

Most of the programming languages can solve mathematical questions and simulate the process of finding the solutions pretty naturally. When simulating a geometric series or a fibonacci sequence these two simple C programs do the job.

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

int fib(int n) {
	printf("%d /n", n);
	return n < 3 ? 1 : fib(n - 1) + fib(n - 2);
}
```

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. Physics engines are useful simulation tools used to observe behaviors of objects in physical systems. They facilitate discovery of patterns which can assist in making future predictions. Unfortunately, traditional programming languages are not optimized for these types of applications. PhysEx is a high-level programming language focused on easing the creation of Physics Engines. An implicit loop simulates the passage of time and applies a stimulus, essentially a function, to each object in the environment on each iteration.

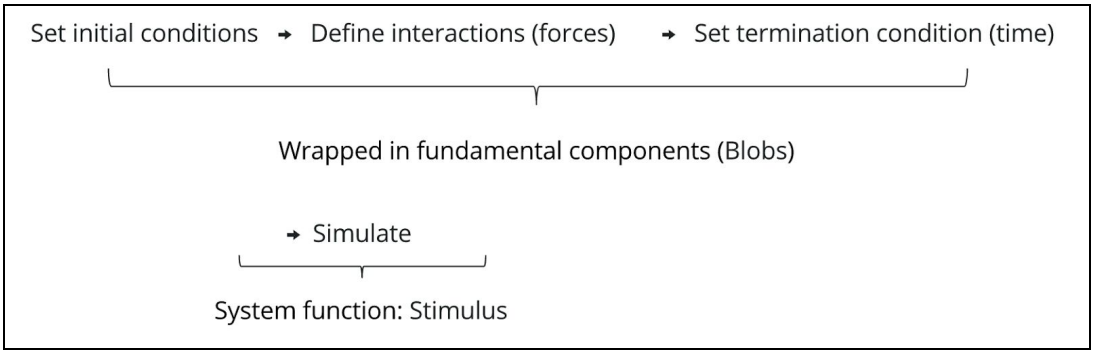


Fig.1. Conceptual overview of PhysEx

2. Language Tutorial

The syntax in PhysEx follows closely from the syntax in the C programming language. Operators, commenting, variable declaration and control flow are very similar to C style. In the following sections, we will provide examples of simple control flow and algorithm implementations.

2.1. Hello World

A typical **.x** file a **simulation** function. In the following example we use **simulation** to implement “hello world”:

```
void func simulation () {
	print("hello world");
}
```

A function in PhysEx is declared with the keyword **func** and in a PhysEx function, the return statement may be omitted if the function type is **void.** Let’s wrap this function in a file **“hello.x”** in the same directory as the source codes of PhysEx. Type the following commands to compile and run **“hello.x”.**

```
$ make
$ (./physex.native < hello.x) > test.ll
$ lli test.ll
```

2.2. GCD algorithm

The following implementation of Euclidean greatest common divisor algorithm demonstrates basic control flow of PhysEx.

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

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

2.3. Free fall

Now let's move on to a more interesting simulation of a free fall motion. We assume an object starts at **100m** above ground and experiences a downward acceleration of **-10 m/s^2** . The following program simulate this process and output the position of the object every second until reaching ground.

```
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;
  }
}
```

3. Language Reference Manual

The following is the reference manual for the PhysEx programming language. PhysEx is a high level language which compiles into LLVM IR and simplifies the programming process by abstracting away hardware specific commands.

PhysEx is an imperative and strongly typed language. The hope is to avoid runtime mismatches by type checking assignments during compilation.

3.1. Lexical Syntax

3.1.1. Identifiers

Identifiers are character sequences which combine together to describe PhysEx programs, the can be comprised of letters, digits, and underscores. PhysEx is a case-sensitive language, therefore `foo` and `Foo` are unique.

3.1.2. Keywords

Keywords are identifiers with special meaning, they are reserved and cannot be declared for general use. The following identifiers are defined by PhysEx:

```
null true false
int string float longDouble bool blob void
if else for while return func start
print printf printfl printi printb sleep simulation
```

3.2. Primitive Data Types

PhysEx is statically typed and provides the following primitive types:

- `string` - Used for storing `char` sets of arbitrary length. Values are wrapped in a pair of quotes.
- `int` - A 32-bit data type used for storing integer values.
- `longDouble` - A 64-bit data type used for storing large integer values.
- `float` - A single-precision floating point value.
- `bool` - A boolean value which can be indicated by the literals [`0, 1, true, false`].

Primitive data types can be directly assigned by using the following template:

```
primitiveType variableName = value;
```

If the type assigned to the variable does not match the variable's assigned type, an error will be thrown during compilation.

3.3. Arrays

Any array is a data structure which allows storage of one or more elements consecutively in memory. At this time, only integers can be stored in each index.

3.3.1. Declaration and Instantiation

Arrays are declared by creating a variable with a integer pointer type, its name, and setting it equal to a pair of brackets. The size of the array is defined between the brackets. For Example:

```
int* myArray = [10];
```

3.3.2. Array Access

Array elements can be accessed by specifying the array name, an open bracket, the index position, and a close bracket. This syntax allows for either read or write capabilities. The following statement updates the initial element in the array initialized in the previous section to the value of 9.

```
myArray[0] = 9;
```

3.3.3. Sparse Arrays

Sparse arrays are not currently supported. When an index is written, ensure all preceding indices have been initialized.

3.4. Comments

Single line, block comments are supported. Comments begin with a pair of forward slashes and end with the first newline character encountered. Any characters between those tokens will not be parsed. For example:

```
// This sentence will not be read by the parser.
```

3.5. Operators

PhysEx supports most standard operators you would find in a high level language. The tables below are listed from highest to lowest precedence and all operators are left-associative unless specified.

3.5.1. Arithmetic

Symbol	*Operator*	*Meaning*
()	Parenthesis	Overrides associativity of the other operators. Everything between them will have precedence over everything outside of them.
*	Multiplication	Multiplies the two operands and produces the product.
/	Division	Divides the left operand by the right one, and produces the quotient.
+	Addition	Adds the operands before and after it and produces a sum.
-	Subtraction	Subtracts the left operand by the right and produces the difference.

3.5.2. Relational

Symbol	*Operator*	*Meaning*
!=	Not Equal	Returns true if lvalue and rvalue are not equal to each other. Otherwise, returns false.
==	Equal	Returns true if lvalue and rvalue are equal to each other. Otherwise, returns false.
<	Less than	Returns true if the left operand is less in value than the right operand. Otherwise, false.
<=	Less than or equal to	Returns true if the left operand is less in value than the right operand or equal in value. Otherwise, false.
>	Greater than	Returns true if the left operand is greater in value than the right operand. Otherwise, false.
>=	Great than	Returns true if the left operand is greater in value than the

	or equal to	right operand or equal in value. Otherwise, false.

3.5.3. Logical

Symbol	*Operator*	*Meaning*
||	OR	If either of the two operands are non-zero, it returns true. Otherwise, false.
&&	AND	If both of the two operands are non-zero, it returns true. Otherwise, false.
!	NOT	Reverse the boolean value of the right operand.

3.5.4. Assignment

The following operators are right-associative.

Symbol	*Operator*	*Meaning*
=	Assignment	Takes the value of the right operand, and stores it into the left operands (assumes it is an identifier).

3.5.5. Precedence Summary

Category	*Operator*	*Associativity*
Postfix	()	Left to Right
Unary	!	Right to Left
Multiplicative	* /	Left to Right
Additive	+ -	Left to Right
Relational	< <= > >=	Left to Right
Equality	== !=	Left to Right
Logical AND	&&	Left to Right
Logical OR	||	Left to Right

3.6. Control Flow

PhysEx supports `if` statements, `while` loops, and `for` loops. From a syntax perspective, all parentheses and braces are mandatory.

3.6.1. `if` Statements

An `if` statement is used to execute a particular block of code based on the result of a boolean condition. In the following example, if *condition* is a true statement then *if-block* will be executed and *else-block* will not be executed. In the case where *condition* is a false statement, the opposite code block will be executed.

```
if (condition) {
        if-block
} else {
        else-block
}
```

Additionally, `else if` statements can be used to add additional conditions to a decision making flow. If the initial *if* condition is false, the *else-if* condition is then evaluated and works similarly to an `if` statement. For example:

```
if (x == 2) {
        if-block
else if (x == 17) {
        else-if-block
} else {
        else-block
}
```

3.6.2. `while` Loops

The condition of `while` loop is evaluated prior to each block execution, if the condition is true the *while-block* is executed. After execution the condition is again tested and, if true, the *while-block* is executed again. This series of events is repeated until the condition is false and the application continues onto the next statement. Example syntax:

```
while (condition) {
        while-block
}
```

3.6.3. `for` Loops

The `for` loop is similar to the `while` loop in that the *for-block* is executed as long as the condition evaluates to true. The primary difference is that a variable can be initialized in prior to the first evaluation of the condition statement and the step expression is executed before each evaluation of the condition statement. Once the condition evaluates to false, the code precedes with the subsequent code block. Example syntax:

```
for (initialize; condition; step) {
        for-block
}
```

3.7. Statements and Blocks

Every statement must be terminated with a semicolon. Blocks of code (e.g. code that executes in part of a control flow) must be contained between a set of braces. It is possible to nest blocks within other blocks. The compiler will go through each line of code sequentially.

3.8. Functions

Functions separate code blocks into re-usable, distinct subprocedures.

3.8.1. Declaration

Functions are declared by first specifying the return type, the `func` keyword, the name, an open parenthesis, a comma separated list of parameters, and a closing parenthesis. The general form is shown below:

```
return-type func name (type param1, type param2, …, type paramN)
```

3.8.2. Definition

The definition section is the block of code which is executed when the function is called. On each execution the parameters are updated to match the values specified in the caller and are passed into the function.

```
void func fooBar (int x, int y) {
        // code-block
}
```

In the above example, the braces are required in order to correctly define the contained code.

3.8.3. Program Entry

A function named `simulation` is the first function to be called in any application, therefore it is required that every program define this function.

3.8.4. `sleep` Function

Physex provides a native `sleep` function which pauses execution of the next statement for a short interval.

```
void sleep (int seconds);
```

3.8.5. `start` Method

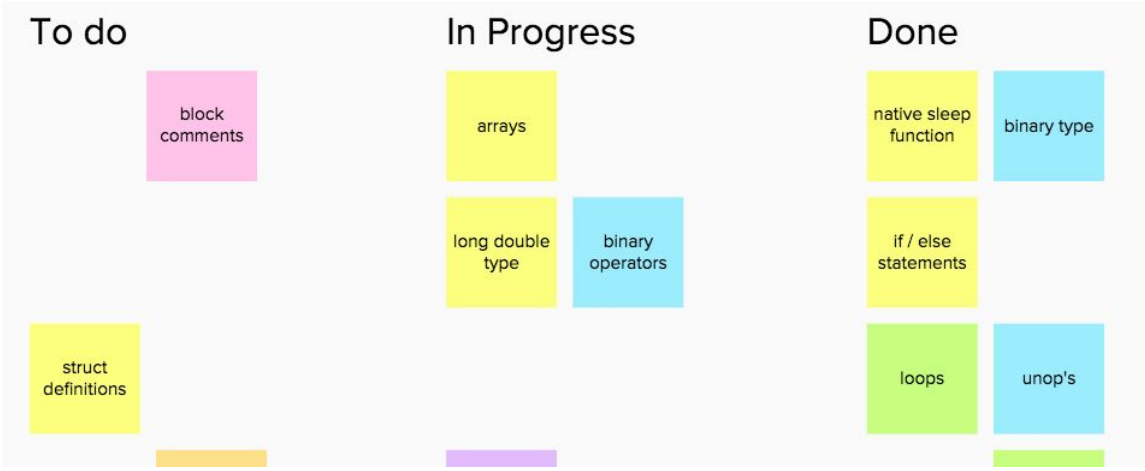
To begin a simulation the `start` method should be defined within the `simulation` function. The method accepts an integer which represents the number of times to repeat the block of code defined in the method. The syntax looks like:

```
start (int seconds) {
	// code to execute
}
```

4. Project Plan

4.1. Planning Process

As much as possible we tried to follow the principles of agile development. Each developer ensured all tests were passing prior to pushing code onto the main source control branch and progress was communicated on a regular basis. From a project management perspective, we followed a process similar to KanBan so we new the status of each feature and effort was not duplicated. A snapshot of tasks is shown below:



The team also relied on Slack for communication. Unlike in a working environment, each team member had a completely unique schedule and outside of a predefined weekly meeting it could be difficult to get everyone together. A persistent chat application became the tool we relied on most, after source control.

4.2. Specification Process

We did not have a formal specification process. Once we defined the usage for our language, each member was free to bring new ideas to the team and we made a decision together. Fortunately, our team was able to come to a compromise on specifications without too much turmoil.

4.3. Development Process

Our implementation followed the course deadlines pretty closely. By the end of November we had a working end-to-end compiler which could output a simple string. From there we worked on adding basic control flow and additional types. The final weeks were spent adding the pieces of the language which were aligned with our overall mission to make Physics simulations easier.

4.4. Testing Process

We primarily focused on small end-to-end tests for each component. As much as possible, we aimed to add tests as we added features, so that we did not have to do much backtracking and incur extra technical debt. Keeping the tests small allowed us to pinpoint where an error was occurring, which proved necessary since the OCaml error reporting can be hard to decipher.

4.5. Programming Style Guide

The team followed a loose style guide:

- No lines greater than 80 characters
- Align tab indention by group
- Two space tab width
- Snake-case for variables and functions in OCaml
- Camel-case for variables and functions in PhysEx
- Always use brackets
- Add newlines to separate code blocks

4.6. Roles and Responsibilities

Member	**Role**	**Responsibilities**
Joshua Nuez	Product Manager	Scheduling, Splitting up tasks and follow up on progress.
Justin Pugliese	Language Guru	Determine specific language features and implementation
Steven Ulahannan	Tester	Planned and Wrote the Test Suite
David Pu	System Architect	Decide system structure and physical aspect of the language

4.7. Development Environment

We used Git/Github as version control for this project. Several instances of Google cloud were used to host the LLVM as well as VirtualBox VM for x86_64bit Ubuntu Linux. PhysEx is compiled using OCaml, Ocamlyacc, Ocamllex and eventually translated to LLVM IR.

4.8. Project Log

Below is the list of git commits from the team. We all contributed.

```
*   commit bfe3a895a0495b669add91b8c9d677c682bb0bc6
|\  Merge: 0afb43c e74e16e
| | Author: su2206@columbia.edu <su2206@columbia.edu>
| | Date:   Tue Dec 20 01:10:13 2016 +0000
| |
| |     Merge branch 'master' of https://github.com/jnuez94/PhysEx
| |
| * commit e74e16e2f3be9cc3035b139f005a86cf9e16a20d
| | Author: plt4115 <jmp5167@gmail.com>
| | Date:   Mon Dec 19 19:13:07 2016 -0500
| |
| |     added a test or two
| |
| *   commit 57ae9ff8f41013f238a3fc92e492c9b02af5337a
| |\  Merge: 6475d6c e05ecb2
| | | Author: plt4115 <jmp5167@gmail.com>
| | | Date:   Mon Dec 19 19:10:27 2016 -0500
| | |
| | |     Merge branch 'master' of github.com:jnuez94/PhysEx
| | |
| * | commit 6475d6cfb62858ae46f810d1dd44228b8c902f98
| | | Author: plt4115 <jmp5167@gmail.com>
| | | Date:   Mon Dec 19 19:10:21 2016 -0500
| | |
| | |     stim updates
| | |
| * | commit 1d1de91a7a794c64c429ada8ca7d5575fbad3039
| | | Author: plt4115 <jmp5167@gmail.com>
| | | Date:   Mon Dec 19 19:09:00 2016 -0500
| | |
| | |     startEnv working
| | |
* | | commit 0afb43c83ca116abdf0e266129a1b3d73dbe506a
| |/  Author: su2206@columbia.edu <su2206@columbia.edu>
```

```
|/|   Date:   Tue Dec 20 01:10:09 2016 +0000
| |
| |       added bool, equal, gcd, factorial tests, printing ability for bool and float
| |
* |   commit e05ecb29fbf4ce7bc38e6bd1254f093b7b254a95
|\ \  Merge: 6121b17 2c98736
| | | Author: su2206@columbia.edu <su2206@columbia.edu>
| | | Date:   Mon Dec 19 23:30:42 2016 +0000
| | |
| | |     Merge branch 'master' of https://github.com/jnuez94/PhysEx
| | |
| * | commit 2c98736a70a00bb47617f624b3b4eac0b2bf2ac8
| |/  Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| |   Date:   Mon Dec 19 23:16:39 2016 +0000
| |
| |       Changed name of function entry point.
| |
* | commit 6121b17ec5de5e3df3311959bf8a6301b42137a6
|/  Author: su2206@columbia.edu <su2206@columbia.edu>
|   Date:   Mon Dec 19 23:30:08 2016 +0000
|
|       partially implemented float
|
* commit 49e471715617ef5df29a16ca3525967c3c3a9fb8
| Author: plt4115 <jmp5167@gmail.com>
| Date:   Mon Dec 19 15:30:19 2016 -0500
|
|     Semant for stimulus
|
* commit 33b357a36f05059a0e6583d136919fca86c84451
| Author: plt4115 <jmp5167@gmail.com>
| Date:   Mon Dec 19 15:28:35 2016 -0500
|
|     beginning stimulus stuff
|
* commit e7a8b591cbca892e258d55caa4544f89f1e331f9
| Author: plt4115 <jmp5167@gmail.com>
| Date:   Mon Dec 19 14:32:36 2016 -0500
|
|     added some clock stuff .. not sure if we'll need it though
|
* commit 72642eac56a38c75decebb0bb7331734610448ed
| Author: plt4115 <jmp5167@gmail.com>
| Date:   Mon Dec 19 09:35:30 2016 -0500
|
|     hopefully fixed sleep
|
* commit 161af8d672f61509adaaeb530463738b5948f76e
| Author: su2206@columbia.edu <su2206@columbia.edu>
| Date:   Mon Dec 19 03:30:16 2016 +0000
```

```
|
|     added while tests
|
* commit e512ae57d6b3efaf8fc3dc3f403ce501a006e3e4
| Author: plt4115 <jmp5167@gmail.com>
| Date:   Sun Dec 18 18:59:03 2016 -0500
|
|     added a sleep method.. hoping to use that for more stimulus work
|
*   commit 845b387f3c80b794459fcf7ab9f52fc200c4da95
|\  Merge: c236ecd adb6812
| | Author: plt4115 <jmp5167@gmail.com>
| | Date:   Sun Dec 18 12:55:50 2016 -0500
| |
| |     merges
| |
| * commit adb6812f84bfe5fd2ded4130cd74756bfb11c809
| | Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| | Date:   Sun Dec 18 05:02:22 2016 +0000
| |
| |     Else if working.
| |
| *   commit 0493c46200bb4c499da246aefc83478e4a44fb29
| |\  Merge: 3e5b9a7 3ba78c4
| | | Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| | | Date:   Sun Dec 18 04:50:34 2016 +0000
| | |
| | |     Merge branch 'master' of https://github.com/jnuez94/PhysEx
| | |
| | * commit 3ba78c4c39a3cf384e73d31adece299605f82f85
| | | Author: su2206@columbia.edu <su2206@columbia.edu>
| | | Date:   Sun Dec 18 03:53:19 2016 +0000
| | |
| | |     fixed function tests
| | |
| * |   commit 3e5b9a7a88acf79b1fc0f34a4f0b7b822f5b6c29
| |\ \  Merge: 4685e48 f6da776
| | |/  Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| | |   Date:   Sun Dec 18 04:50:27 2016 +0000
| | |
| | |       Refactored parser.
| | |
| | * commit f6da7765c6bc19aea5f165080ac9141917306088
| | | Author: su2206@columbia.edu <su2206@columbia.edu>
| | | Date:   Sun Dec 18 03:00:45 2016 +0000
| | |
| | |     added function test, check the ones with errors
| | |
| * | commit 4685e48d9f5f7d5fcfe5ac262125677d5ae2b1ce
| |/  Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
```

```
| |   Date:   Sun Dec 18 03:26:01 2016 +0000
| |
| |       Slight fix to scanner.
| |
| * commit 71cd5100d5aed59b762c2bdc16457415120efbc2
| | Author: su2206@columbia.edu <su2206@columbia.edu>
| | Date:   Sun Dec 18 00:52:37 2016 +0000
| |
| |     negative arithmatic failed test-op3.x
| |
| * commit 9475aca2c61d1677e2e7fb01487dcc9eb29236f8
| | Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| | Date:   Sun Dec 18 00:42:05 2016 +0000
| |
| |     Modification to test to fit functions.
| |
| *   commit c78c0d84a77868d9a3fcf3f16a85eb49dc247d06
| |\  Merge: 0793b10 59289b2
| | | Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| | | Date:   Sun Dec 18 00:33:17 2016 +0000
| | |
| | |     Merged while test.
| | |
| | * commit 59289b23f3e4e7b180b41b839edec01ee249da22
| | | Author: su2206@columbia.edu <su2206@columbia.edu>
| | | Date:   Sun Dec 18 00:17:34 2016 +0000
| | |
| | |     undid directory stuff
| | |
| * | commit 0793b10ef7f1509cc88a86b01760531e951b478d
| | | Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| | | Date:   Sun Dec 18 00:31:32 2016 +0000
| | |
| | |     Functions is declared with type.
| | |
* | | commit c236ecdd659258cd1afe4e23f63e60ab7ceed09d
| |/  Author: plt4115 <jmp5167@gmail.com>
|/|   Date:   Sun Dec 18 12:53:10 2016 -0500
| |
| |       sleep function stuff
| |
* | commit cfa64001af07d673dcfe016fda6117a801af3e74
| | Author: su2206@columbia.edu <su2206@columbia.edu>
| | Date:   Sat Dec 17 23:35:46 2016 +0000
| |
| |     test-if6.x failed
| |
* |   commit 9822d25acec131091aec584000b5f2a44b21454c
|\ \  Merge: d91f1f1 9e814ab
| |/  Author: su2206@columbia.edu <su2206@columbia.edu>
```

```
| |   Date:   Sat Dec 17 23:17:01 2016 +0000
| |
| |       Merge branch 'master' of https://github.com/jnuez94/PhysEx
| |
| *   commit 9e814ab3ef10b98825cdc9562528f85146f3acc9
| |\  Merge: 905b58a 182991c
| | | Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| | | Date:   Sat Dec 17 23:12:25 2016 +0000
| | |
| | |     Merge branch 'master' of https://github.com/jnuez94/PhysEx
| | |
| * | commit 905b58aa6d152db496b3c722d9f0c8fb8a7a7c47
| | | Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| | | Date:   Sat Dec 17 23:12:06 2016 +0000
| | |
| | |     Bug fix on for loop.
| | |
* | | commit d91f1f13f1678d98a19f09ea690f5a4f7fc8b08b
| | | Author: su2206@columbia.edu <su2206@columbia.edu>
| | | Date:   Sat Dec 17 23:16:46 2016 +0000
| | |
| | |     added tests to if
| | |
* | |   commit 73ca3ed7e071ee75dc19878cbb713a5b8400e683
|\ \ \  Merge: 3f02de6 182991c
| | |/  Author: su2206@columbia.edu <su2206@columbia.edu>
| |/|   Date:   Sat Dec 17 22:51:38 2016 +0000
| | |
| | |       Merge branch 'master' of https://github.com/jnuez94/PhysEx
| | |
| * |   commit 182991cfc1d34131398731ce2dd36ca3b6e42c46
| |\ \  Merge: 23d9124 96732e7
| | |/  Author: plt4115 <jmp5167@gmail.com>
| | |   Date:   Sat Dec 17 17:50:32 2016 -0500
| | |
| | |       loop merge
| | |
| * | commit 23d91247ede004b883b3ba7570270ff5e648024d
| | | Author: plt4115 <jmp5167@gmail.com>
| | | Date:   Sat Dec 17 17:44:52 2016 -0500
| | |
| | |     int arrays
| | |
* | | commit 3f02de6dd6d1e82044c70bfeb4650082bd8f6141
| |/  Author: su2206@columbia.edu <su2206@columbia.edu>
|/|   Date:   Sat Dec 17 22:50:34 2016 +0000
| |
| |       reorganized tests into folders
| |
* | commit 96732e79873ff6d6dea391b7688fbb8afb9c1f69
```

```
| | Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| | Date:   Sat Dec 17 22:42:17 2016 +0000
| |
| |     quick fix for Control Flow.
| |
* | commit 2e0cb01f8f091d53426ed04fefc6f263be548996
|/  Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
|   Date:   Sat Dec 17 21:30:04 2016 +0000
|
|       Multiple statements in control flow.
|
* commit ac28deecb20b813790a739cf77024d84ebdb8c5c
| Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| Date:   Thu Dec 15 09:40:13 2016 +0000
|
|     Some test for control flow. If and For are working. Parsing error on while.
|
* commit eb766f1ad159855e1b0a12ccd446445f68f42e04
| Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| Date:   Wed Dec 14 00:54:06 2016 +0000
|
|     Adds the control flow. Not tested yet.
|
* commit 0098de317f9763a5f70fee6cba46a6ff2b414a37
| Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| Date:   Tue Dec 13 23:43:28 2016 +0000
|
|     Tests for binop and integer assignment.
|
* commit d443b3d49e7a74f3527f7ba6b18cdf41eb7b5208
| Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| Date:   Tue Dec 13 16:30:24 2016 +0000
|
|     Working binary operators. Tested, not automated. Implemented unary operator. Not
tested.
|
* commit 01575b60302a32a252d3a55ddd5d48826dfcbb90
| Author: plt4115 <jmp5167@gmail.com>
| Date:   Mon Nov 28 19:10:03 2016 -0500
|
|     made changes to a string can be assigned to a variable and then the variable can
be cout-ed
|
* commit 3c58494b42aeaee9ab1209202ac699c46679d0d9
| Author: plt4115 <jmp5167@gmail.com>
| Date:   Sat Nov 26 14:31:32 2016 -0500
|
|     updated ignore file
|
* commit acbce3049861d851de1b0232f61c898dbbb68356
```

```
| Author: plt4115 <jmp5167@gmail.com>
| Date:   Mon Nov 21 16:04:45 2016 -0500
|
|     print test suite
|
* commit 967b48395aa6e4739e682453b677a41055e7149d
| Author: plt4115 <jmp5167@gmail.com>
| Date:   Mon Nov 21 14:07:54 2016 -0500
|
|     fixed comments
|
* commit 2da897c203b49f0ab37d12390bdb038df2fd711c
| Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| Date:   Mon Nov 21 18:47:42 2016 +0000
|
|     Hello World! in working order.
|
* commit 84c13c2af33f764e1f660320f2f4784fa0c87ddb
| Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| Date:   Mon Nov 21 01:30:00 2016 +0000
|
|     Modified String to Str so no confusion with Ocaml String module.
|
* commit 911b9e6b8eee652d42c3e8092258d3182f19addf
| Author: su2206@columbia.edu <su2206@columbia.edu>
| Date:   Mon Nov 21 00:30:25 2016 +0000
|
|     added llvm compil instruc in readme. currently returns a random numer
|
* commit affa6b72f6574c46fdb62f62639042294a958c83
| Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| Date:   Sun Nov 20 23:21:56 2016 +0000
|
|     String input for print works now.
|
* commit 7a33b6115ca182d11598c0977d4ad2755d74eaad
| Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| Date:   Sun Nov 20 21:20:21 2016 +0000
|
|     Initial take on printing hello world.
|
* commit 0c9f615b61d7a25829feba53f8be1d16ec460307
| Author: Joshua Nuez <jn2548@cloud.cs.columbia.edu>
| Date:   Sun Nov 20 00:37:19 2016 +0000
|
|     Minor addition to the codegen.
|
*   commit 98f9c09afcb7a2bc884cccd9c5b9f652835ef259
|\  Merge: dd57201 f79c8b9
| | Author: JMax <jmp5167@gmail.com>
```

```
| | Date:   Fri Nov 18 19:34:37 2016 -0500
| |
| |     Merge pull request #5 from jnuez94/parser
| |
| |     Parser
| |
| * commit f79c8b942617950e53b19114b22ae32e289b2f3e
| | Author: plt4115 <jmp5167@gmail.com>
| | Date:   Fri Nov 18 19:33:52 2016 -0500
| |
| |     moved some regex so things work better
| |
| * commit a31a0e47260247a419e06d1f836ce20801f542b6
| | Author: plt4115 <jmp5167@gmail.com>
| | Date:   Fri Nov 18 15:09:53 2016 -0500
| |
| |     started coding the codegen
| |
| * commit cd9d6f3944fc0fb4ce54b4c15930db14ea3955d1
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Mon Nov 14 08:24:08 2016 -0500
| |
| |     some changes.. need to figure out testing
| |
| * commit b1d9c2581aa32ef1537372c66d8136773a3168e6
| | Author: plt4115 <jmp5167@gmail.com>
| | Date:   Mon Nov 14 07:48:26 2016 -0500
| |
| |     makefile tweaks
| |
| * commit bf1317481dfe2535696ee112f910e06e6436d8d0
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Mon Nov 14 07:42:48 2016 -0500
| |
| |     makefile adjustments for llvm
| |
| * commit 6484a81b4856969525bbd2348081f41c6965ecca
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Mon Nov 14 07:26:39 2016 -0500
| |
| |     adding the VM instructions from the prof
| |
| * commit 6154cb96d3cfa4c57cd7fefc1de4092a89e3e09d
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Mon Nov 14 06:23:10 2016 -0500
| |
| |     started symbol table
| |
| * commit 0c8f2ba3e13eb73a9a3d75fa7f6de0e13c7ec232
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
```

```
| | Date:   Sun Nov 13 16:41:30 2016 -0500
| |
| |     added built in print function
| |
| * commit 3562cc2dab2a1f97648c667d19a334b9e49867c4
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Sun Nov 13 14:58:51 2016 -0500
| |
| |     reduced semantic ml so that it all compiles
| |
| * commit 1dfddfb3c2656fcd3c7658646bdf90e4b66926b6
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Sun Nov 13 12:00:31 2016 -0500
| |
| |     Fixed some syntax errors
| |
| * commit 9b74ad77b368ef1144612bd0ac05d55689da286a
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Thu Nov 10 19:20:44 2016 -0500
| |
| |     added some more but theres a syntax error to work out
| |
* |   commit dd572018d739c771d41152ffcacacb2c29b8b86a
|\ \  Merge: 1ff59b3 036fdc2
| |/  Author: JMax <jmp5167@gmail.com>
| |   Date:   Wed Nov 9 18:42:08 2016 -0500
| |
| |       Merge pull request #4 from jnuez94/parser
| |
| |       Parser
| |
| * commit 036fdc26b52ff194a7227b5a25b650e611e8dafe
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Wed Nov 9 18:41:02 2016 -0500
| |
| |     got to slide 19 of the smeantic checker.. need to finish tomorrow
| |
| * commit a0eaa79f736ff65c47ed6101b1a1ce4800542bcd
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Wed Nov 9 18:05:55 2016 -0500
| |
| |     adding semantic checking
| |
| * commit 577cc53e8337095f6de7e890fb47b3738d04023e
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Tue Nov 8 18:52:52 2016 -0500
| |
| |     array parsing
| |
| * commit 76e92826a59669367316eb3e9c80a9bb11f0469a
```

```
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Tue Nov 8 18:46:32 2016 -0500
| |
| |     blob-city
| |
| * commit 275476d7e03cbf4d6f77c0d0d035b1fbda90276a
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Tue Nov 8 17:40:12 2016 -0500
| |
| |     finished the normal ast definitions
| |
| * commit b8699192a78345c8fb70cb4e1d9919e1170c33cc
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Mon Nov 7 19:58:58 2016 -0500
| |
| |     more progress on the ast file
| |
| * commit c2d424e5ad0230bdc0c91ee1ba67d1162a7b1fcc
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Fri Nov 4 14:21:07 2016 -0400
| |
| |     some ast updates
| |
| * commit f826c754ad702368c28e25e533f9ecbdaf17703f
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Thu Nov 3 19:17:33 2016 -0400
| |
| |     Started on AST
| |
| * commit 932581e1b8cc9cec5a37917a2dbac8802199eb73
| | Author: Joshua Nuez <jn2548@columbia.edu>
| | Date:   Wed Nov 2 21:24:18 2016 -0400
| |
| |     Made some minor modifications to parser and scanner.
| |
| * commit d279d8cb436d5421ea5c3c74548724b3bf541d2c
| | Author: Joshua Nuez <jn2548@columbia.edu>
| | Date:   Wed Nov 2 21:10:05 2016 -0400
| |
| |     Adds function and editted exprs.
| |
| * commit 095d0e6424e7efe5a5f18b56c05e0e133439db1d
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Wed Nov 2 19:06:39 2016 -0400
| |
| |     added more things for blobs and loops
| |
| * commit e7792fcfecc72a2302e6a76a290e0d877b94f1a8
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Wed Nov 2 18:09:59 2016 -0400
```

```
| |
| |     added comments to scanner
| |
| * commit e2ebd5da2deec940ecdb1356c48fbb2fd53a2277
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Wed Nov 2 18:06:06 2016 -0400
| |
| |     ast is confusing
| |
| * commit ff01a445233b7fc4ee555a02e15f1886ce8ed89d
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Wed Nov 2 17:50:13 2016 -0400
| |
| |     added makefile
| |
| * commit 0f8cf6747b17fe081d8302d6dbcf1f1b33752af8
| | Author: Joshua Nuez <jn2548@columbia.edu>
| | Date:   Wed Nov 2 17:06:51 2016 -0400
| |
| |     Adds comparators.
| |
| * commit a2a32b86743acacb445e61dab9a427a66a603edb
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Wed Nov 2 15:53:00 2016 -0400
| |
| |     updates to ignore
| |
| * commit c4fae077960d36ccb26b4632e624b67b4e1b2f34
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Wed Nov 2 15:38:30 2016 -0400
| |
| |     pusing some changes and cleanup
| |
| * commit 091766b298987069fd4ca88ef57e4aacfa161fb0
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Tue Nov 1 19:46:33 2016 -0400
| |
| |     more
| |
| * commit bdc200cc8f707c7e38b128187fd016615de15720
| | Author: Justin Pugliese <jmpuglie@us.ibm.com>
| | Date:   Tue Nov 1 19:24:41 2016 -0400
| |
| |     sorta started the parser again. but now it compiles .. whoo
| |
| * commit 1d504e6c0e643fb3e687bb94dffab116db78565c
|/  Author: Joshua Nuez <jn2548@columbia.edu>
|   Date:   Mon Oct 31 21:17:37 2016 -0400
|
|       Adds comments and some additional logical expressions.
```

```
|
* commit 1ff59b3898cd106217bbe3ad3ee43b77608b53f9
| Author: David Pu <push0216@gmail.com>
| Date:   Wed Oct 26 23:07:02 2016 -0400
|
|     partially finished with parser
|
* commit 170ab634d83f73599248444a0a7d47edb556c916
| Author: David Pu <push0216@gmail.com>
| Date:   Wed Oct 26 20:07:10 2016 -0400
|
|     more grammar
|
* commit 2f6eab7813a66b3b81707c53f637ba42981525e4
| Author: David Pu <push0216@gmail.com>
| Date:   Wed Oct 26 18:55:37 2016 -0400
|
|     added some logic expressions
|
*   commit 72661f89a81ae4cc325abb2baedfc3daec4accd5
|\  Merge: 0e8b411 c88970c
| | Author: David Pu <push0216@gmail.com>
| | Date:   Tue Oct 25 23:22:01 2016 -0400
| |
| |     Merge branch 'master' of https://github.com/jnuez94/PhysEx into parser
| |
| *   commit c88970c8e585747facd50ce9da9ac752112b3840
| |\  Merge: 0c68ed3 89083fe
| | | Author: jnuez94 <nuez.joshua@gmail.com>
| | | Date:   Tue Oct 25 23:17:30 2016 -0400
| | |
| | |     Merge pull request #3 from jnuez94/scanner
| | |
| | |      Finished scanner for sections 3, 6.2, 6.3, 6.4, 8.1, 8.2, 8.3, 9, an…
| | |
| | * commit 89083fec5c7f6596a6c3ab2eab962ba03c2956f9
| | | Author: Joshua Nuez <jn2548@columbia.edu>
| | | Date:   Tue Oct 25 22:22:13 2016 -0400
| | |
| | |      Finished scanner for sections 3, 6.2, 6.3, 6.4, 8.1, 8.2, 8.3, 9, and 10.
| | |
* | |   commit 0e8b4116b02074a6004c05f9e0aa17392c0e8b92
|\ \ \  Merge: 8ff6379 0c68ed3
| |/ /  Author: David Pu <push0216@gmail.com>
| | |   Date:   Tue Oct 25 23:18:48 2016 -0400
| | |
| | |       Merge branch 'master' into parser
| | |
| * | commit 0c68ed3d56dab53840b3e76a3e49004471fada97
| |/  Author: steveula <su2206@columbia.edu>
```

```
| |   Date:   Tue Oct 25 22:45:08 2016 -0400
| |
| |       test.px
| |
| *   commit b37ee435ac758753cf76335f514cc3a8383bf955
| |\  Merge: 61b3d8d 67544a2
| | | Author: jnuez94 <nuez.joshua@gmail.com>
| | | Date:   Tue Oct 25 21:26:14 2016 -0400
| | |
| | |     Merge pull request #2 from jnuez94/parser
| | |
| | |     Adds parser and ast file.
| | |
| * |   commit 61b3d8d746561728b03f38fbad425e53eccdf0bc
| |\ \  Merge: cfe63b7 7619033
| | | | Author: jnuez94 <nuez.joshua@gmail.com>
| | | | Date:   Tue Oct 25 21:25:29 2016 -0400
| | | |
| | | |     Merge pull request #1 from jnuez94/scanner
| | | |
| | | |     Adds scanner.
| | | |
| | * | commit 7619033ba3f42831a10ce41e784e741361cc2699
| | |/ /  Author: Joshua Nuez <jn2548@columbia.edu>
| | | |   Date:   Mon Oct 24 16:24:36 2016 -0400
| | | |
| | | |       Adds scanner.
| | | |
* | | commit 8ff6379a1f62164a824bda4b3f7de6db2ae0f424
| |/  Author: David Pu <push0216@gmail.com>
|/|   Date:   Tue Oct 25 21:41:02 2016 -0400
| |
| |       a
| |
* | commit 67544a236c87fb34ecad5b5ea834d4cb0740f8da
|/  Author: Joshua Nuez <jn2548@columbia.edu>
|   Date:   Mon Oct 24 16:55:59 2016 -0400
|
|       Adds parser and ast file.
|
* commit cfe63b7d019563bad760c828f629dcc67b1721a6
  Author: jnuez94 <nuez.joshua@gmail.com>
  Date:   Mon Oct 24 15:44:36 2016 -0400

      Initial commit
```

5. Architectural Design

The PhysEx compiler is composed of 4 distinct modules and an interface. The diagram below shows how source code navigates through the compiler and results in LLVM instructions.

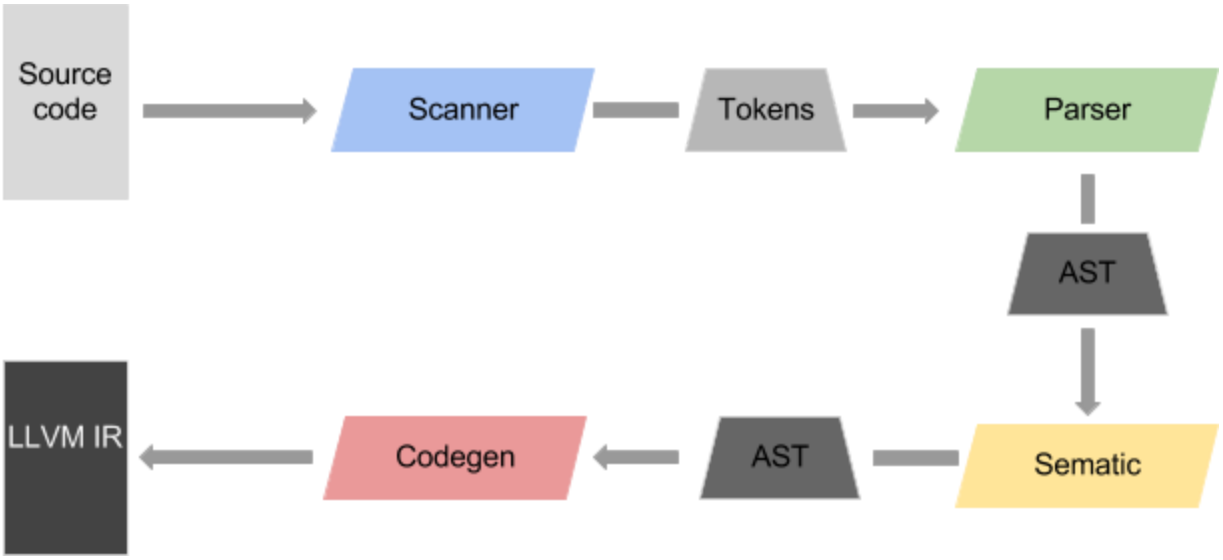


Fig.2 Flowchart of the Design

The Scanner

The scanner reads each character from the source code and creates tokens based on the regular expressions defined within it. Any characters which are deemed as unnecessary are discarded, such as comments and whitespace.

All team members contributed to the scanner.

The Parser

The next step is to turn the tokens into an abstract syntax tree based on the rules provided in the ast interface. The Parser produces the following high-level layout.

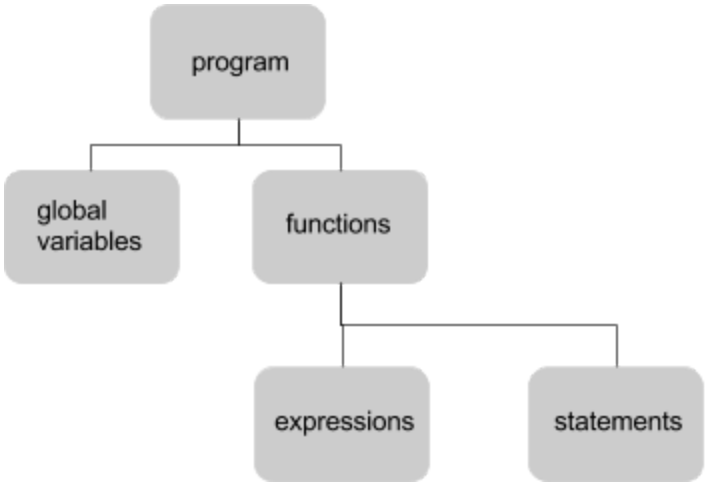


Fig.3 Layout produced by the Parser

David, Josh, and Justin implemented the Parser. Josh and Justin also implemented the AST.

The Semantic Analyzer

Once parsed and correctly conforming to the syntax tree, the code is semantically checked. PhysEx's analyzer ensures function definitions are not duplicated, variable definitions are not duplicated, and reserved words are not used for something unexpected. Additionally, since PhysEx is a statically typed language, the semantic analyzer ensures type safety.

If any of the check fail, the analyzer reports the failure as an error for the programmer to debug.

Justin and Josh implemented the Semantic Analyzer.

The Code Generator

Once the AST is deemed semantically correct it is passed into the code generator which has rules on how to interpret syntax tree node into LLVM instructions. Once complete, the source code which was originally passed in can be output into a file as LLVM IR.

Justin and Josh implemented the Code Generator.

6. Test Plan

A program that has not been tested is a broken program. By transitivity, a language compiler that has not been tested is a broken compiler. A vast, diverse set of tests are necessary to ensure that a compiler correctly performs basic operations, and particularly features that are unique to the language.

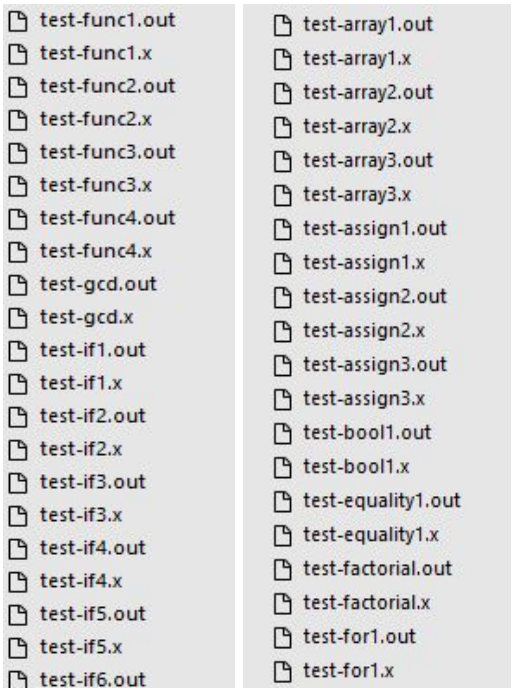


Fig.4 Test Suite to assess basic features of the language

Our test suite does exactly that. It has the checks for assignment, binary operations, if/else, while, for, function calls, etc. The tester wrote different versions of them to various complexities (nested if/else, nested while/for loops, function calls using the previous two, recursive calls, and so on). Such capabilities are imperative for any language to have to be useful, which is why the tester wrote such test cases. The suite also runs basic programs any language should be able to implement like gcd, factorial computation, sleep (borrowed from c), and a simulation program unique to our languages described below:

```
// sleep

int atom;
void func simulation () {
       atom = 3;
       sleep(3);
       printi(atom);
}
```

```
// factorial algo

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

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

```
// rudimentary sim program

int ball;
void func test(){
       ball = ball + 1;
       printi(ball);
}
void func simulation() {
       ball = 0;
       start (5) { test(); }
//prints height of ball over
5sec
}
```

Sample Test Programs from the Test Suite

Test Automation

The team used a shell script, which runs all of the test in the test suite in one sweep and tells the user if the output matches the user's expected output:

```
#!/bin/sh

# Regression testing script for PhysEx
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the microc compiler.  Usually "./microc.native"
# Try "_build/microc.native" if ocamlbuild was unable to create a symbolic link.
PHYSEX="./physex.native"
#MICROC="_build/microc.native"

# Set time limit for all operations
ulimit -t 30

globallog=physex.log
rm -f $globallog
error=0
globalerror=0

keep=0

Usage() {
    echo "Usage: testall.sh [options] [.mc files]"
    echo "-k    Keep intermediate files"
    echo "-h    Print this help"
    exit 1
}

SignalError() {
    if [ $error -eq 0 ] ; then
       echo "FAILED"
       error=1
    fi
    echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
```

```
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
       SignalError "$1 differs"
       echo "FAILED $1 differs from $2" 1>&2
    }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    {
                      if [ -n `eval $*`] ;then
                             return 0
                      else
                             return 1
                      fi
              } || {
       SignalError "$1 failed on $*"
       return 1
    }
}

# RunFail <args>
# Report the command, run it, and expect an error
# RunFail() {
#     echo $* 1>&2
#     eval $* && {
#      SignalError "failed: $* did not report an error"
#      return 1
#     }
#     return 0
# }

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                             s/.x//'`
    reffile=`echo $1 | sed 's/.x$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.out" &&
```

```
    Run "$PHYSEX" "<" $1 ">" "${basename}.ll" &&
    Run "$LLI" "${basename}.ll" ">" "${basename}.out" &&
    Compare ${basename}.out ${reffile}.out ${basename}.diff

    # Report the status and clean up the generated files

  if [ $error -eq 0 ] ; then
       if [ $keep -eq 0 ] ; then
    rm -f $generatedfiles
       fi
       echo "OK"
       echo "###### SUCCESS" 1>&2
  else
       echo "###### FAILED" 1>&2
       globalerror=$error
  fi
}

# CheckFail() {
#     error=0
#     basename=`echo $1 | sed 's/.*\\///
#                               s/.x//'`
#     reffile=`echo $1 | sed 's/.x$//'`
#     basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

#     echo -n "$basename..."

#     echo 1>&2
#     echo "###### Testing $basename" 1>&2

#     generatedfiles=""

#     generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
#     RunFail "$PHYSEX" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
#     Compare ${basename}.err ${reffile}.err ${basename}.diff

#     # Report the status and clean up the generated files

#     if [ $error -eq 0 ] ; then
#      if [ $keep -eq 0 ] ; then
#          rm -f $generatedfiles
#      fi
#      echo "OK"
#      echo "###### SUCCESS" 1>&2
#     else
#      echo "###### FAILED" 1>&2
#      globalerror=$error
#     fi
# }
```

```
while getopts kdpsh c; do
    case $c in
       k) # Keep intermediate files
           keep=1
           ;;
       h) # Help
           Usage
           ;;
    esac
done

shift `expr $OPTIND - 1`

LLIFail() {
  echo "Could not find the LLVM interpreter \"$LLI\"."
  echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
  exit 1
}

which "$LLI" >> $globallog || LLIFail


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
```

While some of the team members wrote the initial, trivial tests for some features, the tester Steven wrote the majority (~87%) of the tests in the test suite.

7. Lessons Learned

Steven Ulahannan

I definitely appreciate how much goes into constructing a commercial language. What we did was just the tip of the iceberg, but even that prove to be quite a hassle. I am surprised that our idea was deemed “feasible,” as we had fairly high expectations in our project proposal. However, it was nice to see how far our project pieced together as much as it did; debugging the semantics while making tests taught me patience. My advice for future teams is to consistently work on the project; even a small amount like 10 lines of code a day can compound into a tremendous impact on your project overall. Also, it definitely pays to work with a group of students who are willing to meet up outside of the TA’s meetings and regularly keeps you posted of the latest updates. Your team’s ability to work together is of paramount importance.

Justin Pugliese

In an effort to impart something helpful and outside of the obvious (start early, join a good group, etc), I suggest taking time, in the early stages, to understand how LLVM works and think of a project that’s feasible within those bounds. We began planning our project with the assumption that we could compile into our choice of language and ultimately committed to developing something we might not have had we known the LLVM requirement.

Joshua Nuez

As the project manager of the group it was part of my responsibility to keep track of our progress and ensure that we were on track to finish. Unfortunately, I was unable to fulfill my responsibilities. With this in mind, I learned the importance of creating a loose roadmap, subject to major changes, but with a clear short and long term goal. It is important to create some sort of routine with your team where you meet and code for at least an hour a week to ensure that there is progress in your code base. The things I learned in a standpoint is that documentation is key whether it is your code or someone else’s. Reading the documentation for the llvm bindings in ocaml was painful to say the least. There are also not that many resources available and those that are available sometimes don’t provide examples for syntax and usage. I’d also like to take this opportunity to thank Justin (Max) for all the hard work and hours he put into this project.

David Pu

I suppose the biggest lesson I have learned is to be ambitious and persistent. Starting out thinking about making a JavaScript-like syntax, we soon realized it would be pretty hard to implement it with LLVM. However, as the system architect, I did not seek to ask for permission for an alternate solution or actually dig into LLVM and thoroughly understand it. I think always having a clear mind on the most essential feature is also important. We have relatively short amount of time to complete a language design. Trying to make the language general purpose is helpful, but the priority should be given to features that define this language and these features should drive the implementation of each step of the language. The learning curve is also pretty substantial once you fell a little behind. So it is really crucial to always catch up with teammates and move along.

8. Appendix

Makefile

```
EXE = physex.native

build:
	ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis,llvm.linker,llvm.bitreader -cflags
-w,+a-4 $(EXE)

ocaml:
	ocamllex scanner.mll
	ocamlyacc -v parser.mly
	ocamlc -c ast.ml
	ocamlc -c parser.mli
	ocamlc -c scanner.ml
	ocamlc -c parser.ml
	ocamlc -c semantic.ml
	ocamlc -c physex.ml
	ocamlc -o physex parser.cmo scanner.cmo ast.cmo semantic.cmo physex.cmo

clean:
	ocamlbuild -clean
	rm -rf scanner.ml parser.ml parser.mli
	rm -rf *.cmx *.cmi *.cmo *.cmx *.o *.bc
	rm -rf physex *.pdi *.out *.ll *.diff
```

physex.ml

```
let _ =
       let lexbuf = Lexing.from_channel stdin in
              let ast = Parser.program Scanner.token lexbuf in
                     Semantic.checker ast;

       print_string (Llvm.string_of_llmodule (Codegen.translate ast))
```

scanner.mll

```
{ open Parser
       let unescape s = Scanf.sscanf ("\"" ^ s ^ "\"") "%S%!" (fun x -> x)
}

let escape = '\\' ['\\' ''' '"' 'n' 'r' 't']
let escape_char = ''' (escape) '''
let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let string = '"' ((ascii | escape)* as s) '"'


rule token = parse
               [' ' '\t' '\r' '\n']  { token lexbuf }
       |       "//"                  { comment lexbuf}

       |       eof                   { EOF }
       |       ':'                   { COLON }
       | ';'                         { SEMICOLON }
       | ','                         { COMMA }

       | "null"       { NULL }
       | "true"       { TRUE }
       | "false"      { FALSE }

       | '=' { ASN }
       | '+' { PLUS }
       | '-' { MINUS }
       | '*' { TIMES }
       | '/' { DIVIDE }

       | '(' { L_PAREN }
       | ')' { R_PAREN }
       | '{' { L_BRACE }
       | '}' { R_BRACE }
       | '[' { L_BRACKET }
       | ']' { R_BRACKET }

       | "||" { OR }
       | '!'  { NOT }
       | "&&" { AND }

       | "==" { EQ }
       | "!=" { NEQ }
       | "<"  { LT }
       | "<=" { LEQ }
       | ">"  { GT }
       | ">=" { GEQ }
```

```
        | "int"        { INT }
        | "string"     { STR }
        | "float"      { FLT }
        | "longDouble" { LD }
        | "bool"       { BOOL }
        | "blob"       { BLOB }
        | "void"       { VOID }

        | "if"         { IF }
        | "else"       { ELSE }

        | "for"        { FOR }
        | "while"      { WHILE }

        | "return"     { RETURN }
        | "func"       { FUNC }
        | "start"      { ST_ENV }

        | ['0'-'9']+ as lit   { NUM_LITERAL(int_of_string lit) }
        | ['0'-'9']+ '.' ['0'-'9']* as lit  { FLOAT_LITERAL(float_of_string lit) }
        | ['$' '_' 'a'-'z' 'A'-'Z'] ['$' '_' 'a'-'z' 'A'-'Z' '0'-'9']* as lit { ID(lit) }
        | string                            { STRING(unescape s) }

and comment = parse
               ['\r' '\n']          { token lexbuf }
        | _                         { comment lexbuf }
```

parser.ml

```
%{
open Ast
%}

%token SEMICOLON L_PAREN R_PAREN L_BRACE R_BRACE L_BRACKET R_BRACKET COMMA
%token PLUS MINUS TIMES DIVIDE ASN PLSASN SUBASN MULASN DIVASN NOT
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR NULL
%token RETURN IF ELSE FOR WHILE ST_ENV INT BOOL VOID STR FLT BLOB LD
%token FUNC COLON
%token <int> NUM_LITERAL
%token <float> FLOAT_LITERAL
%token <string> STRING
%token <string> ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%right NOT NEG

%start program
%type <Ast.program> program

%%

program:
       decls EOF { $1 }

decls:
        /* nothing */ { [], [] }
       | decls fdecl { fst $1, ($2 :: snd $1) }
       | decls vdecl { ($2 :: fst $1), snd $1 }

fdecl:
       typ FUNC ID L_PAREN formals_opt R_PAREN L_BRACE vdecl_list stmt_list R_BRACE {{
              typ = $1;
              fname = $3;
              formals = $5;
              locals = List.rev $8;
              body = List.rev $9
       }}
```

```
formals_opt:
          /* nothing */ { [] }
        | formal_list   { List.rev $1 }

formal_list:
               typ ID                   { [($1,$2)] }
        | formal_list COMMA typ ID { ($3,$4) :: $1 }

typ:
          INT { Int }
        | FLT  { Float }
        | BOOL { Bool }
        | VOID { Void }
        | STR   { Str }
        | LD   { LongDouble }
        | INT TIMES {Int_p}

vdecl_list:
    /* nothing */    { [] }
  | vdecl_list vdecl { $2 :: $1 }

vdecl:
   typ ID SEMICOLON { ($1, $2) }

stmt_list:
    /* nothing */  { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
    expr SEMICOLON { Expr $1 }
  | RETURN SEMICOLON { Return Noexpr }
  | RETURN expr SEMICOLON { Return $2 }
  | L_BRACE stmt_list R_BRACE { Block(List.rev $2) }
  | IF L_PAREN expr R_PAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF L_PAREN expr R_PAREN stmt ELSE stmt    { If($3, $5, $7) }
  | FOR L_PAREN expr SEMICOLON expr SEMICOLON expr R_PAREN stmt
     { For($3, $5, $7, $9) }
  | WHILE L_PAREN expr R_PAREN stmt { While($3, $5) }
  | ST_ENV L_PAREN expr R_PAREN stmt { Environment($3, $5) }


kv_pairs:
        | kv_pair COMMA kv_pairs {$1 :: $3}

kv_pair:
        | expr COLON expr {$1, $3}

expr:
        /* Literals */
```

```
    TRUE             { BoolLit(true) }
  | FALSE            { BoolLit(false) }
  | ID               { Id($1) }
  | FLOAT_LITERAL            { FloatLit($1) }
  | NUM_LITERAL       { NumLit($1) }
  | STRING       { StringLit($1) }

  /* Unary Operators */
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr         { Unop(Not, $2) }

  /* Logical Operators */
  | expr AND    expr { Binop($1, And,   $3) }
  | expr OR     expr { Binop($1, Or,    $3) }

  /* Comparators */
  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }

  /* Arithmetic Operators */
  | ID ASN expr   { Assign($1, $3) }
  | expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }

  /* Arrays */
  | ID ASN L_BRACKET expr R_BRACKET         { ArrayInit($1, $4) }
  | ID L_BRACKET expr R_BRACKET ASN expr { ArrayAsn($1, $3, $6) }
  | ID L_BRACKET expr R_BRACKET                                    { ArrayRead($1, $3) }

  | ID L_PAREN actuals_opt R_PAREN { Call($1, $3) }
  | L_PAREN expr R_PAREN { $2 }

  /* TODO: Redefine Blob */



actuals_opt:
    /* nothing */ { [] }
  | actuals_list  { List.rev $1 }

actuals_list:
    expr                    { [$1] }
  | actuals_list COMMA expr { $3 :: $1
```

ast.ml

```
type op =
          Add
        | Sub
        | Mult
        | Div
        | Equal
        | And
        | Or
        | Neq
        | Less
        | Leq
        | Greater
        | Geq

type uop =
          Neg
        | Not

type typ =
          Int
        | Bool
        | Blob
        | Null (* Need to fix this later: NULL not a type i think. *)
        | Void
        | Float
        | Float_p
        | LongDouble
        | Str
        | Str_p
        | Int_p
        | Stim

type bind = typ * string

type expr =
          NumLit of int
        | BoolLit of bool
        | FloatLit of float
        | Id of string
        | StringLit of string
        | Noexpr
        | Binop of expr * op * expr
        | Unop of uop * expr
        | Assign of string * expr
```

```
        | Call of string * expr list
        | ArrayInit of string * expr
        | ArrayAsn of string * expr * expr
        | ArrayRead of string * expr
        | MapLit of (expr * expr) list

type stmt =
          Block of stmt list
        | Expr of expr
        | If of expr * stmt * stmt
        | For of expr * expr * expr * stmt
        | While of expr * stmt
        | Return of expr
        | Environment of expr * stmt

type func_decl = {
       typ                   : typ;
       fname                 : string;
       formals               : bind list;
       locals                : bind list;
       body                  : stmt list;
}

type program = bind list * func_decl list

let string_of_op = function
          Add -> "+"
        | Sub -> "-"
        | Mult -> "*"
        | Div -> "/"
        | Equal -> "=="
        | Neq -> "!="
        | Less -> "<"
        | Leq -> "<="
        | Greater -> ">"
        | Geq -> ">="
        | And -> "&&"
        | Or -> "||"

let string_of_uop = function
          Neg -> "-"
        | Not -> "!"

let rec expr_value = function
       NumLit(l) -> l

let rec string_of_expr = function
          NumLit(l) -> string_of_int l
        | BoolLit(true) -> "true"
        | BoolLit(false) -> "false"
```

```
        | FloatLit(f) -> string_of_float f
        | StringLit(s) -> s
        | Id(s) -> s
        | Binop(e1, o, e2) -> string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr
e2
        | Unop(o, e) -> string_of_uop o ^ string_of_expr e
        | Call(f, el) -> f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
        | Noexpr -> ""
        | Assign(v, e) -> v ^ " = " ^ string_of_expr e
        | ArrayAsn (var, i, e) -> var ^ "[" ^ string_of_expr i ^ "] = " ^ string_of_expr e
        | ArrayRead (var, i) -> var ^ " from index " ^ string_of_expr i

let rec string_of_stmt = function
        Block(stmts) -> "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
        | Expr(expr) -> string_of_expr expr ^ ";\n";
        | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
        | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^
                        string_of_stmt s
        | If(e, s1, s2) -> "if (" ^ string_of_expr e ^ ")\n" ^
                        string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
        | For(e1, e2, e3, s) ->
                        "for (" ^ string_of_expr e1 ^ " ; " ^ string_of_expr e2 ^ " ; " ^
                        string_of_expr e3 ^ ") " ^ string_of_stmt s
        | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s

let string_of_typ = function
          Int -> "int"
        | Bool -> "bool"
        | Void -> "void"
        | Blob -> "blob"
        | Float -> "float"
        | Str -> "string"
        | Str_p -> "string pointer"
        | Int_p -> "int pointer"
        | Float_p -> "float pointer"
        | LongDouble -> "long double"

let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

let string_of_fdecl fdecl =
        "function " ^ fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
        ")\n{\n" ^
        String.concat "" (List.map string_of_vdecl fdecl.locals) ^
        String.concat "" (List.map string_of_stmt fdecl.body) ^
        "}\n"

let string_of_program (vars, funcs) =
        String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
        String.concat "\n" (List.map string_of_fdecl funcs)
```

semantic.ml

```
open Ast

module StringMap = Map.Make(String)

(*     Semantic checking of a program. Returns void if successful,
       throws an exception if something is wrong.
       Check each global variable, then check each function *)

let checker (globals, functions) =

       (**    Raise an exception if the given list has a duplicate
        *------------------------------------------------------------------------*)
       let report_duplicate exceptf list =
              let rec helper = function
                            n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
                     |      _ :: t -> helper t
                     | [] -> ()
              in helper (List.sort compare list)
       in

       (**    Helper functions
        *------------------------------------------------------------------------*)
              (*     Raise an exception if a given binding is to a void type    *)
              let check_not_void exceptf = function
                            (Void, n) -> raise (Failure (exceptf n))
                     | _ -> ()
              in

              (* Raise an exception for mismatched type. *)
              let rec check_assign lvaluet rvaluet err =
                     if lvaluet == rvaluet then
                            lvaluet
                     else
                            let fn = function
                                          LongDouble -> check_assign lvaluet Int err
                                   | _ -> raise err
                            in fn (rvaluet)
              in

              (* Checking Global Variables *)
              List.iter (check_not_void (fun n -> "illegal void global " ^ n)) globals;
              report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd globals);

              (* Defend built-in functions *)
              if List.mem "print" (List.map (fun fd -> fd.fname) functions)
              then raise (Failure ("function print may not be defined")) else ();
```

```
        report_duplicate (fun n -> "duplicate function " ^ n)
                (List.map (fun fd -> fd.fname) functions);

    (**    Declare built-in functions
     *----------------------------------------------------------------------------*)
        let built_in_decls = StringMap.add "print" {
                typ = Void; fname = "print"; formals = [(Str, "x")];
                locals = []; body = [] }
                (StringMap.add "printfl" {
                typ = Void; fname = "printfl"; formals = [(Float, "x")];
                locals = []; body = [] }
                (StringMap.add "printi" {
                typ = Void; fname = "printi"; formals = [(Int, "x")];
                locals = []; body = [] }
                (StringMap.add "printb" {
                typ = Bool; fname = "printb"; formals = [(Bool, "x")];
                locals = []; body = [] }
                (StringMap.add "clock" {
                typ = LongDouble; fname = "clock"; formals = [];
                locals = []; body = [] }
                (StringMap.singleton "sleep" {
                typ = Void; fname = "sleep"; formals = [(Int, "x")];
                locals = []; body = []
        })))))
        (* Create print function for int *)
        in
        let function_decls =
                List.fold_left (fun m fd -> StringMap.add fd.fname fd m)
                        built_in_decls functions
        in
        let function_decl s = try StringMap.find s function_decls
                with Not_found -> raise (Failure ("unrecognized function " ^ s))
        in


    (**    Check function declarations
     *----------------------------------------------------------------------------*)
        let check_function func =
                List.iter (check_not_void (fun n ->
                        "illegal void formal " ^ n ^ " in " ^ func.fname)) func.formals;

                report_duplicate (fun n ->
                        "duplicate formal " ^ n ^ " in " ^ func.fname)(List.map snd
func.formals);

                List.iter (check_not_void (fun n ->
                        "illegal void local " ^ n ^ " in " ^ func.fname)) func.locals;

                report_duplicate (fun n ->
```

```
                        "duplicate local " ^ n ^ " in " ^ func.fname)(List.map snd
func.locals);

        (**    Variable symbol table
         *---------------------------------------------------------------------------*)
               let symbols = List.fold_left (fun m (t, n) -> StringMap.add n t m)
                      StringMap.empty (globals @ func.formals @ func.locals)
               in

               let type_of_identifier s =
                      try StringMap.find s symbols
                      with Not_found ->
                             raise (Failure ("undeclared identifier " ^ s))
               in

               let rec expr = function
                             NumLit _       -> Int
                      | BoolLit _ -> Bool
                      | StringLit _ -> Str
                      | FloatLit _ -> Float
                      | Id s                 -> type_of_identifier s
                      | Noexpr -> Void
                      | Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2 in
                                    (match op with
                                           Add | Sub | Mult | Div when t1 = Int && t2 = Int
-> Int
                                    | Equal | Neq when t1 = t2 -> Bool
                                    | Less | Leq | Greater | Geq when t1 = Int && t2 = Int ->
Bool
                                    | And | Or when t1 = Bool && t2 = Bool -> Bool
                                    | _ -> raise (Failure ("illegal binary operator " ^
                                                  string_of_typ t1 ^ " " ^ string_of_op op ^
" " ^
                                                  string_of_typ t2 ^ " in " ^ string_of_expr
e)))
                      | Unop(op, e) as ex -> let t = expr e in
                                    (match op with
                                           Neg when t = Int -> Int
                                    | Not when t = Bool -> Bool
                                    | _ -> raise (Failure ("illegal unary operator " ^
string_of_uop op ^
                                           string_of_typ t ^ " in " ^ string_of_expr ex)))
                      | Call(fname,actuals) as call -> let fd = function_decl fname in
                                    if List.length actuals != List.length fd.formals then
                                           raise (Failure ("expecting " ^ string_of_int
(List.length fd.formals)
                                           ^ " arguments in " ^ string_of_expr call)) (*
string_of_expr call *)
                                    else
                                           List.iter2 (fun (ft, _) e -> let et = expr e in
```

```
                                                ignore (check_assign ft et
                                                       (Failure ("illegal actual agrument:
found " ^ string_of_typ et ^
                                                       ", expected " ^ string_of_typ ft ^ "
in " ^ string_of_expr e))))
                                           fd.formals actuals;
                                          fd.typ (* FIX: This is for testing purposes *)

                    | Assign(var, e) as ex -> let lt = type_of_identifier var and rt = expr
e in

                    check_assign lt rt (Failure ("illegal assignment " ^ string_of_typ lt ^

                         " != " ^ string_of_typ rt ^ " in " ^

                         string_of_expr ex))

                    |      ArrayInit (v, s) -> type_of_identifier v

                    | ArrayAsn (v, i, e) as ex -> let fn = function Int_p -> Int in
                                  let lt = fn (type_of_identifier v) and rt = expr e in

              check_assign lt rt (Failure ("illegal assignment " ^ string_of_typ lt ^

                           " = " ^ string_of_typ rt ^ " in " ^

                           string_of_expr ex))

                    | ArrayRead (v, i) -> let fn = function Int_p -> Int in
                                  fn (type_of_identifier v)

              in

              let check_bool_expr e = if expr e != Bool
                    then raise (Failure ("expected Boolean expression"))
              else () in

              let check_int_expr e = if expr e != Int
                    then raise (Failure ("expected Integer expression"))
              in

              let rec stmt = function
                           Expr e -> ignore (expr e)
                    | Block sl -> let rec check_block = function
                                  [Return _ as s] -> stmt s
                           | Return _ :: _ -> raise (Failure "nothing may follow a return")
                           | Block sl :: ss -> check_block (sl @ ss)
                           | s :: ss -> stmt s ; check_block ss
                           | [] -> ()
                    in check_block sl
```

```
                    | Return e -> let t = expr e in if t = func.typ then () else
                                  raise (Failure ("unknown return type " ^ string_of_typ t ^
" expected "
                                  ^ string_of_typ func.typ ^ " in " ^ string_of_expr e))
                    | If(p, b1, b2) -> check_bool_expr p; stmt b1; stmt b2
                    | For(e1, e2, e3, st) -> ignore (expr e1); check_bool_expr e2;
                                  ignore (expr e3); stmt st
                    | While(p, s) -> check_bool_expr p; stmt s
                    | Environment (i, s) -> check_int_expr i; stmt s
            in

            stmt (Block func.body)
      in List.iter check_function functions
```

codegen.ml

```
module L = Llvm
module A = Ast
module StringMap = Map.Make(String)

open Llvm_bitreader
open Llvm_linker

let translate (globals, functions) =
    let context = L.global_context () in
    let the_module = L.create_module context "PhysEx"
      and f32_t = L.float_type context
      and i64_t = L.i64_type context
      and i32_t = L.i32_type context
      and i8_t = L.i8_type context
      and i1_t = L.i1_type context
      and void_t = L.void_type context in
    let str_t = L.pointer_type i8_t in

    let ltype_of_typ = function
        A.Int -> i32_t
      | A.Bool -> i1_t
      | A.Void -> void_t
      | A.Str -> str_t
      | A.Float -> f32_t
      | A.LongDouble -> i64_t
      | A.Str_p -> L.pointer_type str_t
      | A.Int_p -> L.pointer_type i32_t
      | A.Float_p -> L.pointer_type f32_t
    in
(*
    let linker filename =
      let llctx = L.global_context () in
      let llmem = L.MemoryBuffer.of_file filename in
      let llm = Llvm_bitreader.parse_bitcode llctx llmem in
      ignore(link_modules' the_module llm)
    in

    let r = linker "psleep.bc" in *)

    let global_vars =
      let global_var m (t, n) =
        let init = L.const_null (ltype_of_typ t) (* Check if 0 is needed *)
        in StringMap.add n (L.define_global n init the_module) m in
      List.fold_left global_var StringMap.empty globals
    in

    let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
```

```
    let calloc_t = L.function_type str_t [|i32_t;i32_t|] in
    let sleep_t = L.function_type i32_t [| i32_t |] in
    let clock_t = L.function_type i64_t [||] in

    let printf_func = L.declare_function "printf" printf_t the_module in
    let calloc_func = L.declare_function "calloc" calloc_t the_module in
    let sleep_func = L.declare_function "sleep" sleep_t the_module in
    let clock_func = L.declare_function "clock" clock_t the_module in

    let function_decls =
      let function_decl m fdecl =
        let name =
               if fdecl.A.fname = "simulation" then "main" else fdecl.A.fname and
            formal_types = Array.of_list (List.map(fun (t,_) -> ltype_of_typ t)
fdecl.A.formals)
        in
        let ftype = L.function_type (ltype_of_typ fdecl.A.typ) formal_types in
          StringMap.add name (L.define_function name ftype the_module, fdecl) m in
          List.fold_left function_decl StringMap.empty functions in

        let build_function_body fdecl =
               let name = if fdecl.A.fname = "simulation" then "main" else fdecl.A.fname in
          let (the_function, _) = StringMap.find name function_decls in
          let builder = L.builder_at_end context (L.entry_block the_function) in

          let str_format = L.build_global_stringptr "%s\n" "fmt" builder in
          let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder in

          let local_vars =
            let add_formal m (t, n) p = L.set_value_name n p;

        let local = L.build_alloca (ltype_of_typ t) n builder in ignore (L.build_store p local
builder);
        StringMap.add n local m in
          let add_local m (t, n) =
            let local_var = L.build_alloca (ltype_of_typ t) n builder
            in StringMap.add n local_var m in

            let formals = List.fold_left2 add_formal StringMap.empty fdecl.A.formals
              (Array.to_list (L.params the_function)) in
            List.fold_left add_local formals fdecl.A.locals in

        let lookup n = try StringMap.find n local_vars
                with Not_found -> StringMap.find n global_vars
        in


        let init_arr v s = let tp = L.element_type (L.type_of v) in
          let sz = L.size_of tp in
          let sz = L.build_intcast sz (i32_t) "" builder in
```

```
          let dt = L.build_bitcast (L.build_call calloc_func [|s;sz|] "" builder) tp ""
builder in
             L.build_store dt v builder
        in

        let rec expr builder = function
            A.NumLit i -> L.const_int i32_t i
          | A.FloatLit f -> L.const_float i32_t f
          | A.BoolLit b -> L.const_int i1_t (if b then 1 else 0)
          | A.Id s-> L.build_load (lookup s) s builder

          | A.ArrayInit (v, s)-> let var = (lookup v) and size = (expr builder s) in
            init_arr var size

          | ArrayAsn (v, i, e) -> let var = (expr builder (A.Id v))
              and index = (expr builder i) and value = (expr builder e) in
              let k = L.build_in_bounds_gep var [|index|] "" builder in
              L.build_store value k builder

          | ArrayRead (v, i) -> let var = (expr builder (A.Id v))
              and index = (expr builder i) in
              let k = L.build_in_bounds_gep var [|index|] "" builder in
              L.build_load k "" builder

          | A.Noexpr -> L.const_int i32_t 0
          | A.Binop(e1, op ,e2) ->
              let e1' = expr builder e1
              and e2' = expr builder e2 in
              (match op with
                A.Add     -> L.build_add
              | A.Sub     -> L.build_sub
              | A.Mult    -> L.build_mul
              | A.Div     -> L.build_sdiv
              | A.And     -> L.build_and
              | A.Or      -> L.build_or
              | A.Equal   -> L.build_icmp L.Icmp.Eq
              | A.Neq     -> L.build_icmp L.Icmp.Ne
              | A.Less    -> L.build_icmp L.Icmp.Slt
              | A.Leq     -> L.build_icmp L.Icmp.Sle
              | A.Greater -> L.build_icmp L.Icmp.Sgt
              | A.Geq     -> L.build_icmp L.Icmp.Sge
              ) e1' e2' "tmp" builder
          | A.Unop(op, e) ->
              let e' = expr builder e in
              (match op with
                A.Neg     -> L.build_neg
              | A.Not     -> L.build_not) e' "tmp" builder
          | A.StringLit b ->
                      let arr = L.build_global_stringptr b "" builder in
                      let zero = L.const_int i32_t 0 in
```

```
            let s = L.build_in_bounds_gep arr [| zero |] "" builder in s
  | A.Call ("print", [e]) ->
      L.build_call printf_func [| str_format; (expr builder e) |]
      "printf" builder
  | A.Call ("printi", [e]) ->
      L.build_call printf_func [| int_format_str; (expr builder e) |]
      "printf" builder
  | A.Call ("printb", [e]) ->
      L.build_call printf_func [| int_format_str; (expr builder e) |]
      "printf" builder
  | A.Call ("sleep", [e]) ->
    L.build_call sleep_func [| (expr builder e) |]
    "sleep" builder
  | A.Call ("clock", e) ->
    L.build_call clock_func [||]
    "clock" builder
  | A.Call ("printfl", [e]) ->
      L.build_call printf_func [| int_format_str; (expr builder e) |]
      "printf" builder

  | A.Call (f, act) ->
      let (fdef, fdecl) = StringMap.find f function_decls in
        let actuals = List.rev (List.map (expr builder) (List.rev act)) in
        let result = (match fdecl.A.typ with A.Void -> ""
                      | _ -> f ^ "_result") in
        L.build_call fdef (Array.of_list actuals) result builder
  | A.Assign (s, e) -> let e' = expr builder e in
         ignore (L.build_store e' (lookup s) builder); e'
in

let add_terminal builder f =
    match L.block_terminator (L.insertion_block builder) with Some _ -> ()
  | None -> ignore (f builder)
in


let rec stmt builder = function
    A.Block sl -> List.fold_left stmt builder sl
  | A.Expr e -> ignore (expr builder e); builder
  | A.Return e -> ignore (match fdecl.A.typ with
      A.Void -> L.build_ret_void builder
      | _ -> L.build_ret (expr builder e) builder); builder
  | A.If (predicate, then_stmt, else_stmt) ->
      let bool_val = expr builder predicate in
      let merge_bb = L.append_block context "merge" the_function in
      let then_bb = L.append_block context "then" the_function in
      add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
        (L.build_br merge_bb);
      let else_bb = L.append_block context "else" the_function in
      add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
```

```
            (L.build_br merge_bb);

          ignore (L.build_cond_br bool_val then_bb else_bb builder);
          L.builder_at_end context merge_bb
      | A.While (predicate, body) ->
          let pred_bb = L.append_block context "while" the_function in
          ignore (L.build_br pred_bb builder);

          let body_bb = L.append_block context "while_body" the_function in
          add_terminal (stmt (L.builder_at_end context body_bb) body)
            (L.build_br pred_bb);

          let pred_builder = L.builder_at_end context pred_bb in
          let bool_val = expr pred_builder predicate in

          let merge_bb = L.append_block context "merge" the_function in
          ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
          L.builder_at_end context merge_bb
      | A.For (e1, e2, e3, body) -> stmt builder
          (A.Block [A.Expr e1; A.While (e2, A.Block [body ; A.Expr e3])])

      | Environment (it, body) ->
        let i = (A.expr_value it) in
        let rec rollout c i =
          let c = c + 1 in
            if c < i then begin
              List.fold_left stmt builder [body];
              rollout c i
              end
            else
              List.fold_left stmt builder [body]

      in rollout 0 i
    in

    let builder = stmt builder (A.Block fdecl.A.body) in

    add_terminal builder (match fdecl.A.typ with
        A.Void -> L.build_ret_void
      | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))

in

List.iter build_function_body functions;
the_module
```

test.sh

```
#!/bin/sh

# Regression testing script for MicroC
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the PhysEx compiler.
PHYSEX="./physex.native"

# Time limit for all operations
ulimit -t 30

keep=0

SignalError() {
    if [ $error -eq 0 ] ; then
       echo "FAILED"
       error=1
    fi
    echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
       SignalError "$1 differs"
       echo "FAILED $1 differs from $2" 1>&2
    }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    {
        if [ -n `eval $*` ] ; then
            return 0
        else
            return 1
```

```
        fi
    } || {
    SignalError "$1 failed on $*"
    return 1
    }
}

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                             s/.x//'`
    reffile=`echo $1 | sed 's/.x$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.out" &&
    Run "($PHYSEX" "<" $1 ") >" "${basename}.ll" &&
    Run "$LLI" "${basename}.ll" ">" "${basename}.out" &&
    Compare ${basename}.out ${reffile}.out ${basename}.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
       if [ $keep -eq 0 ] ; then
           rm -f $generatedfiles
       fi
       echo "OK"
       echo "###### SUCCESS" 1>&2
        else
       echo "###### FAILED" 1>&2
       globalerror=$error
    fi
}


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

# Run test cases
```

```
for file in $files
do
    case $file in
       *test-*)
           Check $file 2
           ;;
       *fail-*)
           CheckFail $file 2
           ;;
       *)
           echo "unknown file type $file"
           ;;
    esac
Done
```

testall.sh

```
#!/bin/sh

# Regression testing script for MicroC
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the microc compiler.  Usually "./microc.native"
# Try "_build/microc.native" if ocamlbuild was unable to create a symbolic link.
PHYSEX="./physex.native"
#MICROC="_build/microc.native"

# Set time limit for all operations
ulimit -t 30

globallog=physex.log
rm -f $globallog
error=0
globalerror=0

keep=0

Usage() {
    echo "Usage: testall.sh [options] [.mc files]"
    echo "-k    Keep intermediate files"
    echo "-h    Print this help"
    exit 1
}

SignalError() {
    if [ $error -eq 0 ] ; then
       echo "FAILED"
       error=1
    fi
    echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
```

```
        SignalError "$1 differs"
        echo "FAILED $1 differs from $2" 1>&2
    }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    {
                    if [ -n `eval $*`] ;then
                            return 0
                    else
                            return 1
                    fi
            } || {
        SignalError "$1 failed on $*"
        return 1
    }
}

# RunFail <args>
# Report the command, run it, and expect an error
# RunFail() {
#     echo $* 1>&2
#     eval $* && {
#      SignalError "failed: $* did not report an error"
#      return 1
#     }
#     return 0
# }

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                            s/.x//'`
    reffile=`echo $1 | sed 's/.x$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.out" &&
    Run "$PHYSEX" "<" $1 ">" "${basename}.ll" &&
    Run "$LLI" "${basename}.ll" ">" "${basename}.out" &&
    Compare ${basename}.out ${reffile}.out ${basename}.diff
```

```
    # Report the status and clean up the generated files

  if [ $error -eq 0 ] ; then
       if [ $keep -eq 0 ] ; then
    rm -f $generatedfiles
       fi
       echo "OK"
       echo "###### SUCCESS" 1>&2
  else
       echo "###### FAILED" 1>&2
       globalerror=$error
  fi
}

# CheckFail() {
#     error=0
#     basename=`echo $1 | sed 's/.*\\///
#                              s/.x//'`
#     reffile=`echo $1 | sed 's/.x$//'`
#     basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

#     echo -n "$basename..."

#     echo 1>&2
#     echo "###### Testing $basename" 1>&2

#     generatedfiles=""

#     generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
#     RunFail "$PHYSEX" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
#     Compare ${basename}.err ${reffile}.err ${basename}.diff

#     # Report the status and clean up the generated files

#     if [ $error -eq 0 ] ; then
#      if [ $keep -eq 0 ] ; then
#          rm -f $generatedfiles
#      fi
#      echo "OK"
#      echo "###### SUCCESS" 1>&2
#     else
#      echo "###### FAILED" 1>&2
#      globalerror=$error
#     fi
# }

while getopts kdpsh c; do
    case $c in
       k) # Keep intermediate files
```

```
            keep=1
            ;;
        h) # Help
            Usage
            ;;
    esac
done

shift `expr $OPTIND - 1`

LLIFail() {
  echo "Could not find the LLVM interpreter \"$LLI\"."
  echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
  exit 1
}

which "$LLI" >> $globallog || LLIFail


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
```

pclock.c

```
#include <time.h>
#include <stdio.h>

clock_t st, end;
double total;
int i =0;

int main () {
  st = clock();
  printf("starting of the program, start = %ld\n", st);

  for(i=0; i < 10000000; i++) {}

  end = clock();
  printf("loop has ended, end = %ld\n", end);

  total= (double)(end - st) / CLOCKS_PER_SEC;
  printf("CPU time: %f\n", total);
}

// gcc -o pclock pclock.c
// clang -S -emit-llvm pclock.c
```

psleep.c

```
#include <unistd.h>

int psleep (int duration) {
  sleep(duration);
  return 1;
}
```

test-array1.out

```
42
```

test-array1.x

```
int *arr;
void func simulation () {
        arr = [5];
        arr[0] = 42;

        printi(arr[0]);
}
```

test-array2.out

```
42

24


42
```

test-array2.x

```
int *arr;
void func simulation () {
       arr = [5];
       arr[0] = 42;
  arr[1] = 24;
  arr[2] = 42;

       printi(arr[0]);
  print(" ");
  printi(arr[1]);
  print(" ");
  printi(arr[2]);
}
```

test-array3.out

```
42
42
42
42
42
```

test-array3.x

```
int i;
int *arr;
void func simulation () {
       arr = [5];
       arr[0] = 42;
  arr[1] = 42;
  arr[2] = 42;
  arr[3] = 42;
  arr[4] = 42;

  for (i = 0; i < 5; i=i+1) {
```

```
    printi(arr[i]);
  }
}
```

test-assign1.out

```
12
5
6
52
3
2
```

test-assign1.x

```
int a;
int b;
int c;
int d;
int e;
int f;

void func simulation () {
       a = 12;
       b = 5;
       c = 6;
       d = 52;
       e = 3;
       f = 2;

       printi(a);
       printi(b);
       printi(c);
       printi(d);
       printi(e);
       printi(f);
}
```

test-assign2.out

```
printing a
printing b
printing c
printing d
```

test-assign2.x

```
string a;
string b;
string c;
string d;

void func simulation () {
	a = "printing a";
	b = "printing b";
	c = "printing c";
	d = "printing d";

	print(a);
	print(b);
	print(c);
	print(d);
}
```

test-assign3.out

```
2
2
2
-5
2
0
```

test-assign3.x

```
int a;
int b;
int c;
int d;
int e;
int f;

void func simulation () {
	a = b = c = d = e = f = 2;
	d = -5;
	f = 0;

	printi(a);
	printi(b);
	printi(c);
	printi(d);
```

```
	printi(e);
	printi(f);
}
```

test-bool1.out

```
1
0
1
1
0
1
1
1
1
```

test-bool1.x

```
//bringing home the bacon
void func main()
{
	printb(1 < 3);
	printb(1 > 3);
	printb(3 == 3);
	printb(3 != 3);
	printb(3 <= 3);
	printb(3 <= 9);
	printb(3 >= 3);
	printb(3 >= 1);

}
```

test-equality1.out

```
passed test
```

test-equality1.x

```
void func main()
{
	int j;
	int i;
	i = 3;
	j = 5;
	if (i != j){
```

```
                print("passed test");
        }
        else{
                print("failed test");
        }
}
```

test-factorial.out

```
24
```

test-factorial.x

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

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

test-for1.out

```
0
0
1
2
3
10
```

test-for1.x

```
int i;
int j;
void func simulation () {
        j = 0;
        for(i = 0; i < 5; i=i+1){
```

```
                printi(j);
                j = i;
        }
        printi(10);
}
```

test-for2.out

```
0
0
0
1
0
2
0
3
0
4
1
0
1
1
1
2
1
3
1
4
2
0
2
1
2
2
2
3
2
4
3
0
3
1
3
2
3
3
3
4
4
0
```

```
4
1
4
2
4
3
4
4
test done
```

test-for2.x

```
int i;
int j;
void func simulation () {
	j = 0;
	for(i = 0; i < 5; i=i+1){
		for (j = 0; j < 5; j=j+1){
			printi(i);
			printi(j);
		}
	}
	print("test done");
}
```

test-func1.out

```
passed func test
```

test-func1.x

```
void func test(){
	print("passed func test");
}
void func simulation () {
	test();
}
```

test-func2.out

```
Kaahhh..meehhhhh...
haa..meehhhhh...
HAAAAAAAAAAAA!
```

test-func2.x

```
void func finishing_move(int i, int j){
	if(i == 0){
		print("Kaahhh..meehhhhh...");
		return;
	}
	if(i == 1){
		print("haa..meehhhhh...");
	}
	else{
		print("HAAAAAAAAAAAA!");
	}

}
void func simulation () {
	int i;
	int j;
	j = 1;
	for(i = 0; i < 3; i = i + 1){
		finishing_move(i, j);
	}
}
```

test-func3.out

```
Passed test
```

test-func3.x

```
void func test(){
	test2();
}

void func test2(){
	print("Passed test");
}

void func simulation () {
	test();
}
```

test-func4.out

```
1
2
```

```
3
4
5
6
7
8
9
10
```

test-func4.x

```
int func RecTest(int i){
        int result;
        result = 0;
        if (i == 1){
        return 1;
        }
    else{
        result = RecTest(i - 1);
        printi(result);
        result = result + 1;
    }
    return result;
}

void func simulation () {
        int i;
        i = 11;
        RecTest(i);
}
```

test-gcd.out

```
4
```

test-gcd.x

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

```
	return x;
}


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

test-if1.out

```
42
17
```

test-if1.x

```
void func simulation () {
	if (true) { printi(42);}
	printi(17);
}
```

test-if2.out

```
42
7
```

test-if2.x

```
void func simulation () {
	if (false) {
		printi(13);
	} else {
		printi(42);
	}
	printi(7);
}
```

test-if3.out

```
42
17
5
```

test-if3.x

```
void func simulation () {
```

```
	if (true) {
		printi(42);
	}
	if (false) {
		printi(5);
	} else {
		printi(17);
	}
	printi(5);
}
```

test-if4.out

```
17
```

test-if4.x

```
void func simulation () {
	if (true) {
		if (false){
			printi(42);
		}
	}
	printi(17);
}
```

test-if5.out

```
47
7
```

test-if5.x

```
void func simulation () {
	if (false) {
		printi(13);
	} else {
		if (4 < 5){
			printi(47);
		}
	}
	printi(7);
}
```

test-if6.out

```
42
```

```
17
5
```

test-if6.x

```
void func simulation () {
	if (true) {
		printi(42);
	}
	if (false) {
		printi(5);
	} else {
		if (3 != 6){
			if (5 == 5) {
				printi(17);
			}
			else {
				print("passed string equality");
			}
		}
	}
	printi(5);
}
```

test-if7.out

```
17
```

test-if7.x

```
void func simulation () {
	if (1 == 5) {
		printi(42);
	}else if ((5 - 3) == 2){
		printi(17);
	}
	else {
		print("dank memes");
	}
}
```

test-op1.out

```
10
1
-3
72
```

```
900
20
3
87
124
6
0
24
27
15
```

test-op1.x

```
void func simulation () {
	printi(1+2+3+4);
	printi(5-4);
	printi(3-6);
	printi(9*8);
	printi(100*9);
	printi(100/5);
	printi(33/11);
	printi(87);
	printi(124);
	printi(10-7+3);
	printi(12+3-15);
	printi(8*9/3);
	printi(90/10*3);
	printi(5+4*3-4/2);
}
```

test-op2.out

```
17
72
1
-4
6
60
26
2
-30
-2
10
15
0
8
```

test-op2.x

```
int a;
int b;
int c;
int d;
int e;
int f;

void func simulation () {
	a = 12;
	b = 5;
	c = 6;
	d = 52;
	e = 3;
	f = 2;

	printi(a+b);
	printi(a+c+d+f);
	printi(e - f);
	printi(f - c);
	printi(f*e);
	printi(a*b);
	printi(d/f);
	printi(a/c);
	printi(a - d+b+b);
	printi(f+e+b - a);
	printi(b*c/e);
	printi(c/f*b);
	printi(f*e - c);
	printi(f*e+a/c);
}
```

test-op3.out

```
17
56
5
4
-6
60
-26
-2
-30
-6
-10
15
0
```

```
-8
12
```

test-op3.x

```
int a;
int b;
int c;
int d;
int e;
int f;

void func simulation() {
	a = 12;
	b = 5;
	c = -6;
	d = 52;
	e = 3;
	f = -2;

	printi(a+b);
	printi(a+c+d+f);
	printi(e-f);
	printi(f-c);
	printi(f*e);
	printi(a*b);
	printi(d/f);
	printi(a/c);
	printi(a-d+b+b);
	printi(f+e+b-a);
	printi(b*c/e);
	printi(c/f*b);
	printi(f*e-c);
	printi(f*e+a/c);
	printi(f*c);
}
```

test-print1.out

```
hello world
```

test-print1.x

```
void func simulation () {
	print("hello world");
}
```

test-print2.out

```
Hello world, again!
```

test-print2.x

```
// Demonstrates ability to assign values to variables and output them
string out;
void func simulation () {
	out ="Hello world, again!";
	print(out);
}
```

test-simulation.out

```
1
2
3
4
5
```

test-simulation.x

```
int ball;
void func test(){
	ball = ball + 1;
	printi(ball);
}
void func simulation() {
	ball = 0;
	start (5) {
		test();
	}
}
```

test-sleep1.out

```
3
```

test-sleep1.x

```
int atom;
void func simulation () {
```

```
        atom = 3;
        sleep(3);
        printi(atom);
}
```

test-while1.out

```
0
1
2
3
4
```

test-while1.x

```
int i;
void func simulation() {
	i = 0;
	while(i < 5) {
		printi(i);
		i=i+1;
	}
}
```

test-while2.out

```
slam
slam
slam
dank
slam
slam
slam
dank
slam
slam
slam
dank
memes
slam
slam
slam
dank
slam
slam
slam
dank
slam
slam
slam
dank
memes
```

```
slam
slam
slam
dank
slam
slam
slam
dank
slam
slam
slam
dank
memes
```

test-while2.x

```
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){
				print("slam");
				k = k + 1;
			}
			print("dank");
			k = 0;
		}
		print ("memes");
		i=i+1;
	}
}
```

test-while3.out

```
passed test
```

test-while3.x

```
int i;

void func test2(){
	i = i - 1;
}

void func test(){
	if (i == 1){
		print ("passed test");
	}
	else{
		test2();
	}
}

void func simulation() {
	i = 3;
	while(i > 0) {
		test();
		i = i - 1;
	}
}
```