Dice Project Report

"Java, but worse"

Project Manager:	David Watkins	djw2146
Language Guru:	Emily Chen	ec2805
System Architect:	Phillip Schiffrin	pjs2186
Tester & Verifier:	Khaled Atef	kaa2168

CONTENTS

1. INTRODUCTION

The Dice programming language is an object-oriented, general purpose programming language. It is designed to let programmers who are more familiar with object oriented programming languages to feel comfortable with common design patterns to build useful applications. The syntax of Dice resembles the Java programming language. Dice compiles down to LLVM IR which is a cross-platform runtime environment. This allows Dice code to work on any system as long as there is an LLVM port for it, which includes Windows, Mac OS X, and Linux or various processor architectures such as x86, MIPS, and ARM[1].

Dice lays programs out the same way a Java program would. Variables and methods of a class can be declared with private scope. There is a simple to use inheritance that allows for multiple children inheriting the fields and methods of its parent. Dice also allows for convenient use of functions that exist in C, such as malloc, open, and write. This allows the user to construct objects and call c functions using those objects.

Background

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects". These objects are data structures that contain data, in the form of fields, often known as attributes. The code itself are contained within methods in the code which are compiled to varying subroutines. The most useful aspect of OOP is that these methods and fields can modify one another allowing for a rich and varied use case.

Class based OOP specifically creates instances of classes, referred to as objects, which have their values modified at runtime. There are many languages that implement their language this way including Java and C#.

Inheritance is when an object or class is based on another class using the same implementation. This allows for a class to serve as a blueprint for subclasses. Polymorphism allows an object to take on many forms. This may include an object being assigned to a type that is a class it inherits from, or being used in place of a class it inherits from.

We want to leverage these capabilities using LLVM code to produce a syntactically Java-like language but offer a cross platform solution that is simple and easy to use. Implementing inheritance and objects in a c-like context like LLVM allows for fine control over the code.

Related Work

Object-oriented programming languages have existed since the late 20th century. Java, C#, C++, Objective-C, Python, and many more languages have facilities for defining custom user classes and manipulating them at runtime.

---

[1]http://llvm.org/

Implementing an object-oriented paradigm using C is a well-known solution, but compiling object-oriented code down to LLVM is not publicly available. We want to contribute to the LLVM community by adding additional information regarding the creation of a compiler using OCaml that compiles to LLVM code.

Goals

Cross-Platform

Utilizing the LLVM IR we are able to compile the source once and have it work on multiple architectures without fail.

Flexibility

Allowing the user to define their own classes and offering them the ability to inherit functionality from other user defined types offer a wide range of possibilities for their programs and also saves the user time when implementing large programs.

Transparency

Using the LLVM IR allows the user to see exactly what the program is doing after the compiler is done. For a more optimal result it can then be compiled to bitcode representation using the LLVM compiler.

Familiarity

Incorporate familiar primitive data types most commonly found in languages such as C, C++, and Java such as int, char, float, and bool.

2. LANGUAGE TUTORIAL

Environment Setup

The compiler has been built an tested using an Ubuntu 15.10 virtual machine. The ISO for downloading Ubuntu 15.10 can be found here[1]. This is followed by downloading virtualbox and following the corresponding tutorial for setting up a custom Ubuntu VM here[2]. Once inside the VM there are a series of packages that need to be installed before you can compile the compiler. Run the following commands to install the corresponding packages:

```
>sudo apt-get install m4 clang-3.7 clang-3.7-doc libclang-common-3.7-dev libclang-3.7-dev
↪    libclang1-3.7 libclang1-3.7-dbg libllvm-3.7-ocaml-dev libllvm3.7 libllvm3.7-dbg
↪    lldb-3.7 llvm-3.7 llvm-3.7-dev llvm-3.7-doc llvm-3.7-examples llvm-3.7-runtime
↪    clang-modernize-3.7 clang-format-3.7 python-clang-3.7 lldb-3.7-dev liblldb-3.7-dbg
↪    opam llvm-runtime
```

Then initialize OCaml's package manager (OPAM) in your home directory:

```
>opam init
>opam switch 4.02.1
>eval $(opam config env)
>opam install core batteries llvm yojson
```

After OPAM is initialized, go to the the directory where you want Dice installed and clone the git repository:

```
>git clone https://github.com/DavidWatkins/Dice.git
```

Using the Compiler

Inside the directory 'Dice' type **make**. This creates the dice compiler that takes in '.dice' files and compiles them to corresponding '.ll' files corresponding to LLVM IR. The syntax for running the dice executable is: **dice [optional-option] ⟨source file⟩**. There are also additional flags with respect to the compiler that allow for additional options.

- **-h** - Print help text

- **-tendl** - Prints tokens with newlines intact

- **-t** - Prints token stream

- **-p** - Pretty prints Ast as a program

- **-ast** - Prints abstract syntax tree as json

- **-sast** - Prints semantically checked syntax tree as json

---

[1]http://www.ubuntu.com/
[2]http://www.wikihow.com/Install-Ubuntu-on-VirtualBox

- **-c** - Compiles source and prints result to stdout

- **-f** - Compiles source to file (⟨filename⟩.⟨ext⟩ → ⟨filename⟩.ll)

The following sample dice code demonstrates the following features:

- The mandatory main function that exists within **only** one class. The syntax for a main declaration is **public void main(char[][] args)**

- Calling the built-in print function, which takes an arbitrary list of primitive values, including char[].

- A string literal with escape characters

- Defining a base class with one or more fields.

```
1   class example1 {
2       public void main(char[][] args) {
3           print("This is example 1\n");
4       }
5   }
```

To compile the sample code above, type:

```
> ./dice example1.dice
```

The output will be a file named **example1.ll** which will run using the **lli** command:

```
>lli example1.ll
This is example 1
>
```

If you get an error: "error: expected value token" from lli, that means your version of lli is probably set incorrectly. Run the following command to verify the version:

```
>lli --version
```

If it's anything other than version **3.7** change it with the following commands:

```
>sudo rm \usr\bin\lli
>ln -s /usr/lib/llvm-3.7/bin/lli /usr/bin/lli
```

The basics

Primitives

All primtives are declared starting with their type followed by an identification. Dice supports the following primitives:

- integers (int)

- floating point (float)

- characters (char)

- booleans (bool)

```
1   class example2 {
2       public void main(char[][] args) {
3
4               (* This is a comment (* with a nested one inside *) *)
5               int a; (* Declaring an integer primitive variable *)
6               a = 1; (* Assigning the number one to variable a *)
7
8               float b = 1.5; (* Combined declaration and assignment is okay *)
9
10              (* Characters and booleans are primitives as well *)
11              char c = 'c';    (* ASCII or digits only within single quotes*)
12              bool d = true;   (* or 'false' *)
13          }
14  }
```

Arrays

Arrays are indexed collections of values of a datatype (primitive or object). Dice allows for single dimension arrays only. The elements within the arrays created default to null which, like C, are implemented with zeros.

```
1   class example3 {
2       public void main(char[][] args) {
3
4               int[] a = new int[10]; (* int array with 10 elements set to zero *)
5
6               a[0] = 1; (* Access the first element and assign the integer 1 to it *)
7
8               int[] b = |0,1,2,3,4,5|; (* int array with 6 int elements *)
9
10              print(b.length); (* prints 6 *)
11
12              char[] c = |'h','i', 0|; (* ints are allowed to be stored in char
                ↪   elements *)
13
14          }
15  }
```

Operators

Dice supports the following binary operators:

- Arithmetic (+ , - , * , / ,
- Relational (== , != , ¿ , ¿= , ¡= , ¡)
- Logical (and, or)

Unary operators:

- Logical negation (not)
- Negative number (-)

```
1   class example4 {
2         public void main(char[][] args) {
3
4                 int a = 1 + 2;        (* a is now 3 *)
5                 float b = 2.5 - 2;  (* 2 is promoted to float, b is now 0.5 *)
6                 int c = 5 + 2 * 4;  (* c is 13 due to operator precedence *)
7                 int d = 10 / 5 + 3; (* d is now 5 *)
8                 int e = 5 % 3;              (* e is now 2 *)
9
10                bool f = true; bool g = false;
11                f == f; f != g; 5 > 2; 3 >= 3; f or g;  (* all expressions evaluate to
                  ↪   true *)
12                f and g; not f; (* evaluate to false *)
13
14                c = -a;    (* c is now -3 *)
15        }
16  }
```

Control Flow

The statements inside source files are generally executed from top to bottom, in the order that they appear.
Control flow statements, however, break up the flow of execution by employing decision making, looping, and
branching, enabling your program to conditionally execute particular blocks of code. This section describes
the decision-making statements (if-then, if-then-else), the looping statements (for, while), and the branching
statements (break, continue, return) supported by Dice.

Branching

```
1   class example5 {
2         public void main(char[][] args) {
3                 int a;
4                 if (true)
5                         a = 1;
6                 else
7                         a = 0;
8                 (* a is now 1 *)
```

```
 9
10              int b;
11              if (false){
12                      b = 2; a = 3;
13              }
14              else {
15                      b = 0; a = 0;
16              }
17              (* b and a are now 0 *)
18
19              int c;
20              if(false){ a = 1; b = 1; c = 1;}
21              else if(true) { a = 5; b = 5; c = 5;}
22              else { a = 0; b = 0; c = 0;}
23              (* a,b,c are now set to 5 *)
24      }
25  }
```

Loops

The two types of loops that Dice supports are 'for' and 'while' loops. The for statement allows the programmer the iterate over a range of values. The while statement executes a user-defined block of statements as long as a particular conditional expression evaluates to true.

```
 1  class example6 {
 2      public void main(char[][] args) {
 3              int a = 0;
 4              int i;                    (* The loop counter must be declared outside the
                ↪ for loop *)
 5              for (i = 0 ; i < 5 ; i = i + 1) {
 6                  a = a + 2;
 7              }
 8              (* a is now set to 10 *)
 9
10              int b = 0;
11              int j;
12              for (j = 0 ; j < 5 ; j = j + 1) {
13                      a = a + 2;
14                  if(a >= 14){
15                          break;                     (* will break out of the parent for
                            ↪ loop *)
16                  }
17                  else { continue; } (* will skip the remaining code and start the
                    ↪ next iteration *)
18                  b = b + 10;
19              }
20              (* b is still zero, a is 14 *)
21
22              while(b<5){
23                      b = b + 1;
```

```
24                     }
25
26                     (* b is now 4 *)
27             }
28    }
```

Defining methods

Dice supports methods that return a datatype after execution or simply execute without returing anything.
Methods can accept arguments which are computed in an applicative order. Each method must also contain
a scope (public/private) which determine access for outside classes. The following example will show two
kinds of methods:

```
1     class example7 {
2             public int p(int i){
3                     print(i);
4                     return i;
5             }
6
7             public void q(int a, int b, int c){
8                     int total = a ;
9                     print(b);
10                    total = total + c ;
11            }
12
13            public void main(char[][] args) {
14                    this.q( this.p(1), 2, this.p(3));
15            }
16    }
```

The output of this program is:

```
132
```

Classes and Inheritance

Since Dice is an Object Oriented language, you can create custom classes that can serve as datatypes. A
class contains three sections:

- Fields

- Constructors

- Methods

These sections may be written in any order desired. You may also mix them up if desired. For example, a
constructor may be added inbetween field declarations if desired. If no constructors are defined, Dice will
use a default constructor to instantiate objects. A parent class can also be assigned any class that is a
descendant of it as shown below:

```
1     class shape {
2             public int xCoord;      (* Fields *)
```

```
3          public int yCoord;

4

5          constructor(){                    (* Constructor *)
6                  this.xCoord = 0;
7                  this.yCoord = 0;
8          }

9

10         (* Constructor with a different signature due to the two arguments *)
11         constructor(int x, int y){
12                 this.xCoord = x;
13                 this.yCoord = y;
14         }

15

16         public void myAction(){  (* Method *)
17                 print("shape");
18         }
19         }

20

21         class circle extends shape {
22         public int radius;           (* Field unique to circle *)

23

24         constructor(){
25                 this.xCoord = 0;          (* xCoord and yCoord from parent class 'shape'
                   ↪  *)
26                 this.yCoord = 0;
27                 this.radius = 0;
28         }

29

30         constructor(int x, int y, int r){
31                 this.xCoord = x;
32                 this.yCoord = y;
33                 this.radius = r;
34         }

35

36         public void myAction(){   (* This method overrides the one defined in parent
           ↪  class *)
37                 print("circle\n");
38                 print(this.radius);

39

40         }
41 }

42

43 class example8 {
44         public void main(char[][] args) {
45             class circle a = new circle(1, 2, 3);
46             class circle[] b = new class circle[10];
47             b[0] = a;
48             print(b[0].radius,"\n");

49
```

```
50            class shape c = new circle(4, 5, 6);  (* Inheritance in action! *)
51            c.myAction();
52            print("\n");
53        }
54  }
```

The output for example8 is:

```
3
circle
6
```

3. LANGUAGE REFERENCE MANUAL

Introduction

Dice is a general purpose, object-oriented programming language. The principal is simplicity, pulling many themes of the language from Java. Dice is a high level language that utilizes LLVM IR to abstract away hardware implementation of code. Utilizing the LLVM as a backend allows for automatic garbage collection of variables as well.

Dice is a strongly typed programming language, meaning that at compile time the language will be type-checked, thus preventing runtime errors of type.

This language reference manual is organized as follows:

- Chapter 2 Describes types, values, and variables, subdivided into primitive types and reference types

- Chapter 3 Describes the lexical structure of Dice, based on Java. The language is written in the ASCII character set

- Chapter 4 Describes the expressions and operators that are available to be used in the language

- Chapter 5 Describes different statements and how to invoke them

- Chapter 6 Describes the structure of a program and how to determine scope

- Chapter 7 Describes classes, how they are defined, fields of classes or their variables, and their methods

- Chapter 8 Discusses the different library classes provided with the compiler and their definitions

The syntax of the language is meant to be reminescent of Java, thereby allowing ease of use for the programmer.

Types

There are two kinds of types in the Dice programming language: primitive types and non-primitive types. There are, correspondingly, two kinds of data values that can be stored in variables, passed as arguments, returned by methods, and operated on: primitive values and non-primitive values.

```
Type:
    PrimitiveType
    NonprimitiveType
```

There is also a special null type, the type of the expression *null*, which has no name. Because the null type has no name, it is impossible to declare a variable of the null type. The null reference is the only possible value of an expression of null type. The null reference can always undergo a widening reference conversion to any reference type. In practice, the programmer can ignore the null type and just pretend that *null* is merely a special literal that can be of any reference type.

Primitive Types and Values

A primitive type is predefined by the Dice programming language and named by its reserved keyword.

```
PrimitiveType:
    NumericType
    bool
NumericType:
    IntegralType
    float
IntegralType: one of
    int char
```

int

A value of type *int* is stored as a 32-bit signed two's-complement integer. The *int* type can hold values ranging from -2,147,483,648 to 2,147,483,647, inclusive.

float

The float type stores the given value in 64 bits. The *float* type can hold values ranging from 1e-37 to 1e37. Since all values are represented in binary, certain floating point values must be approximated.

char

The *char* data type is a 8-bit ASCII character. A *char* value maps to an integral ASCII code. The decimal values 0 through 31, and 127, represent non-printable control characters. All other characters can be printed by the computer, i.e. displayed on the screen or printed on printers, and are called printable characters. The character 'A' has the code value of 65, 'B' has the value 66, and so on. The ASCII values of letters 'A' through 'Z' are in a contiguous increasing numeric sequence. The values of the lower case letters 'a' through 'z' are also in a contiguous increasing sequence starting at the code value 97. Similarly, the digit symbol characters '0' through '9' are also in an increasing contiguous sequence starting at the code value 48.

bool

A variable of type *bool* can take one of two values, *true* or *false*. A bool could also be *null*.

Non-Primitive Types

Non-primitive types include arrays and classes.

Arrays

An array stores one or more values of the same type contiguously in memory. The type of an array can be any primitive or an array type. This allows the creation of an n-dimensional array, the members of which can be accessed by first indexing to the desired element of the outermost array, which is of type *array*, and then accessing into the desired element of the immediately nested array, and continuing n-1 times.

Classes

Classes are user-defined types. See chapter 7 to learn about the usage of objects.

Casting

Casting is not supported in this language. There are interesting behaviors between ints and float defined in the section on operators that imitate casting, but there is no syntax to support casting between types directly.

Lexical Conventions

This chapter describes the lexical elements that make up Dice source code. These elements are called tokens. There are six types of tokens: identifiers, keywords, literals, separators, and operators. White space, sometimes required to separate tokens, is also described in this chapter.

Identifiers

Identifiers are sequences of characters used for naming variables, functions and new data types. Valid identifier characters include ASCII letters, decimal digits, and the underscore character '_'. The first character must be alphabetic.
An identifier cannot have the same spelling (character sequence) as a keyword, boolean or null literal, a compile-time error occurs. Lowercase letters and uppercase letters are distinct, such that foo and Foo are two different identifiers.

```
ID = "['a'-'z' 'A'-'Z'](['a'-'z' 'A'-'Z']|['0'-'9']|'\textunderscore')*"
```

Keywords

Keywords are special identifiers reserved for use as part of the programming language itself. You cannot use them for any other purpose. Dice recognizes the following keywords:

if	else	for	while	
break	continue	return		
int	float	bool	char	void
null	true	false	class	constructor
public	private	extends	include	this

Literals

A literal is the source code representation of a value of a primitive type or the null type.

Integer Literals

An integer literal is expressed in decimal (base 10). It is represented with either the single ASCII digit 0, representing the integer zero, or an ASCII digit from 1 to 9 optionally followed by one or more ASCII digits from 0 to 9.

```
INT = "['0'-'9']+"
```

Float Literals

A float literal has the following parts: an integer part, a decimal point (represented by an ASCII period character), and a fraction part. The integer and fraction parts are defined by a single digit 0 or one digit from 1-9 followed by more ASCII digits from 0 to 9.

```
FLOAT = "['0'-'9']+ ['.'] ['0'-'9']+"
```

Boolean Literals

The boolean type has two values, represented by the boolean literals true and false, formed from ASCII letters.

$$\text{BOOL} = \texttt{"true|false"}$$

Character Literals

A character literal is always of type *char*, and is formed by an ascii character appearing between two single quotes. The following characters are represented with an escape sequence, which consists of a backslash and another character:

- '\\' - backslash

- '\"' - double-quote

- '\'' - single-quote

- '\n' - newline

- '\r' - carriage return

- '\t' - tab character

It is a compile-time error for the character following the character literal to be other than a single-quote character '.

```
CHAR = "\' ( ([' '-'!' '#'-'[' ']'-'~'] | '\\' [ '\\' '\"' 'n' 'r' 't' ]) )\' "
```

String Literals

A string literal is always of type char[] and is initialized with zero or more characters or escape sequences enclosed in double quotes.

```
char[] x = "abcdef\n";
```

```
STRING = "\"( ([' '-'!' '#'-'[' ']'-'~'] | '\\' [ '\\' '\"' 'n' 'r' 't' ]) )*\""
```

Separators

A separator separates tokens. White space is a separator but it is not a token. The other separators are all single-character tokens themselves: ( ) [ ]   ; , .

```
'('        { LPAREN }
')'        { RPAREN }
'{'        { LBRACE }
'}'        { RBRACE }
';'        { SEMI }
','        { COMMA }
'['        { LBRACKET }
']'        { RBRACKET }
'.'        { DOT }
```

Operators

The following operators are reserved lexical elements in the language. See the expression and operators section for more detail on their defined behavior.

```
 +    -    *    /    =
==   !=   <   <=   >
>=
```

White Space

White space refers to one or more of the following characters:

- the ASCII SP character, also known as "space"

- the ASCII HT character, also known as "horizontal tab"

- the ASCII FF character, also known as "form feed"

- LineTerminator

White space is ignored, except when it is used to separate tokens. Aside from its use in separating tokens, it is optional. Hence, the following two snippets of source code are equivalent.

```
public int foo()
{
        print( "hello, world\n" );
        return 0;
}

public int foo(){print("hello, world\n"); return 0;}
```

```
WHITESPACE = "[' ' '\t' '\r' '\n']"
```

Comments

The characters (∗ introduce a comment, which terminates with the characters ∗). Multiline comments can be distinguished from code by preceding each line of the comment with a * similar to the following:

```
(* This is a long comment
 * that spans multiple lines because
 * there is a lot to say. *)
```

```
COMMENT = "(\* [^ \*)]* \*)"
```

Expressions and Operators

The precedence of expression operators is the same as the order of the major subsections of this section (highest precedence first). Within each subsection, the operators have the same precedence. Left- or right-associativity is specified in each subsection for the operators discussed therein.

Primary Expressions

Primary expressions involving . , subscripting, and function calls group left to right.

Identifier

An identifier is a primary expression, provided it has been suitably declared as discussed below. Its type is specified by its declaration.

Literal

Any of the literal types discussed in Chapter 3 is a primary expression, which evaluates to the type of the literal.

(expression)

A parenthesized expression is a primary expression whose type and value are identical to those of the unadorned expression. The presence of parentheses does not affect whether the expression is an lvalue.

primary-expression [expression]

A primary expression followed by an expression in square brackets is a primary expression. The intuitive meaning is that of a subscript. The primary expression has type *array* of . . . and the type of the result is . . . . The type of the subscript expression must be a type that is convertible to an integral type, or a compile-time error occurs.

primary-expression (expression-list-opt)

A function call is a primary expression followed by parentheses containing a possibly empty, comma-separated list of expressions which constitute the actual arguments to the function. The result of the function call is the function's return type. Recursive calls to any function are permissible.

primary-lvalue . member-of-structure

An lvalue expression followed by a dot followed by the name of a class member is a primary expression. The object referred to by the lvalue is assumed to be an instance of the class defining the class member. The given lvalue can be an instance of any user-defined class.

Unary operators

Expressions with unary operators group right-to-left.

** expression**

The result is the negative of the expression, and has the same type. The type of the expression must be *char*, *int*, or *float*.

not expression

The result of the logical negation operator *not* is *true* if the value of the expression is *false*, *false* if the value of the expression is *true*. The type of the result is *bool*. This operator is applicable only to operands that evaluate to *bool*.

Multiplicative operators

The multiplicative operators * and / group left-to-right.

expression * expression

The binary * operator indicates multiplication. Operands of *int*, *float*, and *char* types are allowed. If both operands are of type ..., the result is type .... If the operands are of two different types of the ones listed above, the result is the type of the left-most operand.

expression / expression

The binary / operator indicates division. The same type considerations as for multiplication apply.

Additive operators

The additive operators + and group left-to-right.

expression + expression

The value of the result is the sum of the expressions. The same type considerations as for multiplication apply. Overflow of a *char* type during an addition operation results in wraparound.

expression - expression

The value of the result is the difference of the expressions. The same type considerations as for multiplication apply.

Relational operators

The relational operators group left-to-right.

expression < expression

expression > expression

expression <= expression

expression >= expression

The operators < (less than), > (greater than), <= (less than or equal to) and >= (greater than or equal to) all yield *true* is the specified relation is true and *false* otherwise. The same type considerations as for multiplication apply.

Equality operators

expression == expression

expression != expression

The == (equal to) and the != (not equal to) operators are exactly analogous to the relational operators except for their lower precedence.

Logical operators

expression and expression

Both operands must evaluate to a value of type *bool*. The *and* operator returns *true* if both its operands evaluate to *true*, *false* otherwise. The second expression is not evaluated if the first evaluates to *false*.

expression or expression

Both operands must evaluate to a value of type *bool*. The *or* operator returns *true* if either of its operands evaluate to *true*, and *false* otherwise. The second operand is not evaluated if the value of the first operand evaluates to *true*.

Assignment operators

lvalue = expression

The value of the expression replaces that of the object referred to by the lvalue. Both operands must have the same type.

Statements

A statement forms a complete unit of execution.

Include Statement

If a .dice file contains a statement of the following form:

```
include(mylib)
```

then all classes defined in *mylib* are available to be used in definitions of classes in the .dice file in which the include statement appears.

Expression Statements

An expression statement consists of an expression followed by a semicolon. The execution of such a statement causes the associated expression to be evaluated. The following types of expressions can be made into a statement by terminating the expression with a semicolon (;):

```
(* Assignment expressions *)
aValue = 8933.234;
(* Method invocations *)
game.updateScore(Player1, 5);
(* Object creation expressions *)
Bicycle myBike = Bicycle();
```

Declaration Statements

A declaration statement declares a variable by specifying its data type and name.

```
float aValue;
```

Control Flow Statements

The statements inside source files are generally executed from top to bottom, in the order that they appear. Control flow statements, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to conditionally execute particular blocks of code. This section describes the decision-making statements (if-then, if-then-else), the looping statements (for, while), and the branching statements (break, continue, return) supported by the Dice programming language.

if-then, if-then-else

The 'if-then' statement tells the program to execute a certain section of code only if a particular test evaluates to true. The conditional expression that is evaluated is enclosed in balanced parentheses. The section of code that is conditionally executed is specified as a sequence of statements enclosed in balanced braces. If the conditional expression evaluates to false, control jumps to the end of the if-then statement.

```
if (condition) {
        <stmt>
}

if (not condition) {
        <stmt>
} (* if statement is skipped *)
```

The 'if-then-else' statement provides an alternate path of execution when "if" clause evaluates to false. This alternate path of execution is denoted by a sequence of statements enclosed in balanced braces, in the same format as the path of execution to take if the conditional evaluates to true, prefixed by the keyword "else".

```
if (condition) {
        <stmt>
} else {
        <stmt2>
} (* <stmt2> executed when not condition *)
```

Looping: for, while

The 'for' statement allows the programmer the iterate over a range of values. The 'for' statement has the following format:

```
for (initialization; termination; update) { <stmt> }
```

- The 'initialization' expression initializes the loop counter. It is executed once at the beginning of the 'for' statement

- When the 'termination' expression evaluates to false, the loop terminates.

- The 'update' expression is invoked after each iteration and can either increment or decrement the value of the loop counter.

The following example uses a 'for' statement to print the numbers from 1 to 10:

```
int loopCounter;
for (loopCounter=1; loopCounter<11; loopCounter++) {
        print(loopCounter);
}
```

The 'while' statement executes a user-defined block of statements as long as a particular conditional expression evaluates to true. The syntax of a 'while' statement is:

```
while (expression) {
        <stmt>
}
```

The following example uses a 'while' statement to print the numbers from 1 to 10:

```
int loopCounter;
loopCounter = 1;
while (loopCounter < 11) {
        print(loopCounter);
        loopCounter = loopCounter + 1;
}
```

Branching: break, continue, return

If a 'break' statement is included within either a 'for' or 'while' statement, then it terminates execution of the innermost looping statement it is nested within. All break statements have the same syntax:

```
break;
```

In the following example, the 'break' statement terminates execution of the inner 'while' statement and does not prevent the 'for' statement from executing its block of statements for all iterations of i from 1 to 10. This results in the the values of j from 100 to 110 being printed, in each of the 10 iterations of the 'for' loop.

```
int i;
int j;
for (i=1; i<11; i++) {
        j = 100;
        while (j<120) {
                if (j>110) {
                        break;
                }
                print(j);
                j = j + 1;
        }
}
```

In the following example, the 'break' statement terminates execution of the inner 'for' statement and does not prevent the 'while' statement from executing its block of statements for all iterations of i from 1 to 1000. This results in the the values of j from 100 to 110 being printed, in each of the 1000 iterations of the 'while' loop.

```
int i;
int j;

i = 1;
while (i<1001) {
        for (j=100; j<120; j++) {
                if (j>110) {
                        break;
                }
        }
        i = i + 1;
}
```

The continue statement skips the current iteration of a 'for' or 'while' statement, causing the flow of execution to skip to the end of the innermost loop's body and evaluate the conditional expression that controls the loop. The following example uses a 'continue' statement within a 'for' loop to print only the odd integers

between 1 and 10. The code prints "hello" 1000 times and on each of the 1000 'while' loop iterations, prints the odd integers.

```
int i;
int counter;
counter = 1;
while (counter < 1001) {
        print("hello");
        for (i=1; i<11; i++) {
                if (i - 2*(i/2) == 0) {
                        continue;
                } else {
                        print(i);
                }
        }
        counter = counter + 1;
}
```

The 'return' statement exits from the current method, and control flow returns to where the method was invoked. To return a value, simply put the value (or an expression that calculates the value) after the return keyword:

```
return count + 4;
```

The data type of the returned value must match the type of the method's declared return value. When a method is declared void, either no return statement is needed or the following 'return' statement is used:

```
return;
```

Blocks

A block is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed. The following example, BlockDemo, illustrates the use of blocks:

```
class BlockDemo {
        public void main(char[,] args) {
                bool condition;
                condition = true;
                if (condition) { (* begin block 1 *)
                        print("Condition is true.");
                } (* end block one *)
                else { (* begin block 2 *)
                        print("Condition is false.");
                } (* end block 2 *)
        }
}
```

Dice Functions

There are several reserved functions in Dice that cannot be overridden and follow a particular syntax and return type.

File I/O

Manipulating files is an important aspect of any programming languages. Open files are denoted by a particular $int fd$; that can be used to read or write from a file. A file must be closed by the end of a program or else undefined behavior may occur.

int fopen(char[] filename, bool isWriteEnabled)

Accepts a filename and a flag to determine whether the file will be written to. If the file exists, it will be opened in append mode, otherwise a new file will be created. If it is in read mode, it will return a file descriptor as normal, or if the file doesn't exist will return '-1'. Likewise for write enabled, if there is an error it will return -1.

```
int fd;
fd = fopen("hello.txt", false);
```

bool fwrite(int fd, char[] values, int num, int offset)

Accepts an array of values to be written to a file, the number of characters it should write, and the offset into the value array it should write from. If there is an error, returns false, otherwise returns true.

```
bool success;
success = fwrite(fd, "This should work", 4, 1); (* Writes "his " to a file *)
```

bool fread(int fd, char[] storage, int num)

Accepts an array to store values from the file that are to be read, and will read in num bytes. Returns true on success and false on error.

```
char[] a;
bool success;

a = char[100];
success = fread(fd, a, 20);
```

bool fclose(int fd)

Closes a file. Returns true on success, false on error.

```
bool success;
success = fclose(fd);
```

Reading and Writing from Console

Reading and writing to the console is defined by two simple to use functions that cannot be overriden.

void print(char[] string)

Accepts a char array and prints the string to the console.

```
print("hello world");
```

void print(int num)

Accepts an int and prints the int to the console.

```
print(1);
```

void input(char[] buf)

Accepts a buf that will hold read bytes from the console. Then it will write those bytes to the array passed. Terminates when a user enters a newline or an EOF.

```
char[] a;
a = char[100];
input(a);
```

Program Structure and Scope

Program structure and scope define what variables are accessible and where. When inside a class, there are many different cases of scope, however those are better defined in chapter 7.

Program Structure

A Dice program may exist either within one source file or spread among multiple files which can be linked at compile-time. An example of such a linked file is the standard library, or *stdlib.dice*. When an include statement is executed at compile time, it will load in the files mentioned at the includes and insert the code at that location as if it were part of the head source file. Therefore at compilation, one only needs to compile with *dicecmaster.dice*. If an included module defines a class that has the same name as one of the classes defined in the including module, then the compiler throws an error. The compiler does not resolve recursive includes; if *foo.dice* includes *bar.dice* and *bar.dice* includes *foo.dice*, the compiler throws an error.

A program consists of zero or more include statements, followed by one or more class definitions. Each class defined in a module must have a distinct name. Only one class out of all classes may have a main method, defined with *public void main(char[][] args)* which designates the entry point for a program to begin executing code. All Dice files are expected to end with the file extension *.dice* and follow the following syntactic layout.

```
include(stdlib)
include(mylib)

class FOO {

        (* my code *)

}

class BAR {

        (* my code *)

        public void main(char[][] args)

}
```

Scope

Scope refers to which variables, methods, and classes are available at any given time in the program. All classes are available to all other classes regardless of their relative position in a program or library. Variable

scope falls into two categories: fields (instance variables) which are defined at the top of a class, and local variables, which are defined within a method. Fields can be public or private. If a field is public then it is accessible whenever an instance of that class is instantiated. For instance, if I have a class X, then class Y can be defined as follows:

```
class Y {

        public int num;

        constructor() {

                X myObj;
                myObj = X();
                this.num = myObj.number;
        }
}

class X {

        public int number;

}
```

In this example, class Y has one field which is an int. In its constructor, an instance of class X is declared, and a public field within that object is used to set the value for the given int. If a field is declared private, however, it can only be accessed by the methods in the same class. For example, if there is a class Y with a private field, the following is valid:

```
class Y {

        private int num;

        constructor() {

                this.num = 5;
        }

        private int getNum() {

                return this.num;
        }
}
```

However, if I have a class X, that class cannot access the private field within Y. The following is invalid:

```
class X {

        public int number;

        constructor() {
```

```
                    Y myObj;
                    myObj = Y();
                    (* This code is invalid since num is a private field within Y *)
                    this.number = myObj.num;
            }
    }
```

Methods are also declared as public or private, and their accessibility is the same as fields. They must have a scope defined on them.

Local variables are variables that are declared inside of a method. Local variables are only accessible within the same method in which they are declared, and they may have the same name as fields within the same class since fields in a class are only accessible by calling the *this* keyword.

Classes

Classes are the constructs whereby a programmer defines their own types. All state changes in a Dice program must happen in the context of changes in state maintained by an object that is an instance of a user-defined class.

Class definition

A class definition starts with the keyword 'class' followed by the class name (see identifiers in chapter 2) and the class body. The class body, enclosed by a pair of curly braces, declares one or more of each of the following: fields, methods, and constructors.
The members of a class type are all of the following:

- Members inherited from its ancestors (its direct superclass and its ancestors)

- Members declared in the body of the class, with the exception of constructors

Access modifiers

Class member declarations must include access modifiers but the class declaration itself does not; there is no notion of a private class in Dice. Field and method declarations must include one of the access modifiers: *public* or *private*. Fields and methods with the access modifier *public* can be accessed by methods defined in any class. Fields and methods with the access modifier *private* can be accessed by methods defined either in the same class or in successor classes (classes derived directly from that class and their successors).

Fields

The only fields that can be declared are instance variables, which are freshly incarnated for each instance of the class. Field declarations have the following format:

```
<access modifier> <type> <VariableDeclaratorId>;
(* Example *) private int myInstanceVariable;
```

All instance variables must be declared before methods and constructors.

Methods

A method declares executable code that can be invoked, passing a fixed number of values as arguments. The only methods that can be declared are the 'main' method and instance methods. Instance methods are invoked with respect to some particular object that is an instance of a class type.

Method declarations constitute a method header followed by a method body. The method header has the following format:

```
<access modifier> <return type> <method name> <comma-separated list of parameters>
(* Example *) public double amountPaid(double wage, int duration)
```

The method body contains, enclosed between the ASCII characters '{' and '}', zero or more variable declarations followed by zero or more statements. If the type of the return value is not void, then the method body must include a return statement.

One and only one of the classes to be compiled must contain a definition for a method named "main" that executes when the program runs. The *main* method is not callable as an instance method. The *main* method must have a void return type and accept a single parameter of type char[][]. Hence, its signature must be:

```
public void main (char[][] args)
```

If either zero or more than one class contains a definition for a method with the signature above, this results in a compile-time error.

Methods can be overloaded: If two methods of a class (whether both declared in the same class, or both inherited by a class, or one declared and one inherited) have the same name but signatures that are not equivalent, then the method name is said to be overloaded. There can be multiple methods with the same name defined for a class, as long as each has a different number and/or type of parameters. The *main* method can never be overloaded because it has one and only one accepted signature. If two methods in the same class have the same signature, the compiler throws an error.

Constructors

Constructors are similar to methods but cannot be invoked as an instance method; they are used to initialize new class instances. A constructor has no return type and its formal parameters are identical in syntax and semantics to those of a method. A constructor definition has the following format:

```
constructor (<comma-separated formal arguments>) {
        <list of variable declarations>
        <list of statements>
}
(* Example *) constructor (int a, char[] b) {...}
```

Unlike fields and methods, access to constructors is not governed by access modifiers. Constructors are accessible from any class.

Constructor declarations are never inherited and therefore are not subject to overriding.

If no constructors are defined, the compiler defines a default constructor. Like methods, they may be overloaded. It is a compile-time error to declare two constructors with equivalent signatures in a class.

When the programmer declares an instance of the class, either a user-defined constructor or the default constructor is automatically called.

```
class Foo {
        constructor (int x) {...}
        ...
}
class Bar {
        public void main (char[][] args) {
                int x;
                Foo myFooObj;
                x = 5;
                myFooObj = Foo(x);
        }
}
```

Referencing instances

The keyword 'this' is used in the body of method and constructor declarations to reference the instance of the object that the method or constructor will bind to at runtime.

Inheritance

The members of a class include both declared and inherited members. A class inherits all members of its direct superclass and superclasses of that class. To define a class Y that inherits members of an existing class named "X" and all superclasses of X, use the keyword *extends* when defining Y.

```
class Y extends X {...}
```

Overriding

Newly declared methods can override methods declared in any ancestor class. An instance method m1, declared in class C, overrides another instance method m2, declared in class A iff both of the following are true:

- C is a subclass of A

- The signature of m1 is identical to the signature of m2

Grammar

Below you will find an entire grammar listing for our language. You will see several tokens that were generated directly from our Scanner. The following are the list of tokens and their associated regexes:

```
1  let alpha = ['a'-'z' 'A'-'Z']
2  let escape = '\\' ['\\' ''' '"' 'n' 'r' 't']
3  let escape_char = ''' (escape) '''
4  let ascii = ([' '-'!' '#'-'[' ']'-'~'])
5  let digit = ['0'-'9']
6  let id = alpha (alpha | digit | '_')*
7  let string = '"' ( (ascii | escape)* as s) '"'
```

```
 8   let char = ''' ( ascii | digit ) '''
 9   let float = (digit+) ['.'] digit+
10   let int = digit+
11
12   | '('       { LPAREN }
13   | ')'       { RPAREN }
14   | '{'       { LBRACE }
15   | '}'       { RBRACE }
16   | ';'       { SEMI }
17   | ','       { COMMA }
18   | '+'       { PLUS }
19   | '-'       { MINUS }
20   | '*'       { TIMES }
21   | '/'       { DIVIDE }
22   | '%'       { MODULO }
23   | '='       { ASSIGN }
24   | "=="      { EQ }
25   | "!="      { NEQ }
26   | '<'       { LT }
27   | "<="      { LEQ }
28   | ">"       { GT }
29   | ">="      { GEQ }
30   | "and"     { AND }
31   | "or"      { OR }
32   | "not"     { NOT }
33   | '.'       { DOT }
34   | '['       { LBRACKET }
35   | ']'       { RBRACKET }
36   | '|'           { BAR }
37   | "if"      { IF }
38   | "else"    { ELSE }
39   | "for"     { FOR }
40   | "while"   { WHILE }
41   | "return"  { RETURN }
42   | "int"     { INT }
43   | "float"   { FLOAT }
44   | "bool"    { BOOL }
45   | "char"    { CHAR }
46   | "void"    { VOID }
47   | "null"    { NULL }
48   | "true"    { TRUE }
49   | "false"   { FALSE }
50   | "class"       { CLASS }
51   | "constructor" { CONSTRUCTOR }
52   | "public"      { PUBLIC }
53   | "private"     { PRIVATE }
54   | "extends"     { EXTENDS }
55   | "include"     { INCLUDE }
56   | "this"        { THIS }
```

```
57   | "break"                    { BREAK }
58   | "continue"          { CONTINUE }
59   | "new"                    { NEW }
60   | "delete"                      { DELETE }
61
62   | int as lxm                        { INT_LITERAL(int_of_string lxm) }
63   | float as lxm                      { FLOAT_LITERAL(float_of_string lxm) }
64   | char as lxm                       { CHAR_LITERAL( String.get lxm 1 ) }
65   | escape_char as lxm{ CHAR_LITERAL( String.get (unescape lxm) 1) }
66   | string                            { STRING_LITERAL(unescape s) }
67   | id as lxm                    { ID(lxm) }
68   | eof                          { EOF }
69
70   | (* *) {COMMENT*}
```

It should be noted that comments were handled to allow for nested comments. Therefore this cannot be captured strictly using a grammar, and instead is better shown in the scanner.mll documentation at the end of this document. The following grammar is the same as the grammar shown in parser.mly at the end of this document except it does not have the rules it will turn into regarding OCaml code. This is very similar to the syntax for ocamlyacc.

```
1    program:
2            includes cdecls EOF
3
4    includes:
5                    /* nothing */
6            |       include_list
7
8    include_list:
9                    include_decl
10           |       include_list include_decl
11
12   include_decl:
13           INCLUDE LPAREN STRING_LITERAL RPAREN SEMI
14
15   cdecls:
16           cdecl_list
17
18   cdecl_list:
19                   cdecl
20           |       cdecl_list cdecl
21
22   cdecl:
23                   CLASS ID LBRACE cbody RBRACE
24           |        CLASS ID EXTENDS ID LBRACE cbody RBRACE
25
26   cbody:
27                   /* nothing */
28           |       cbody field
29           |       cbody constructor
```

```
30                |             cbody fdecl

31

32  constructor:

33          CONSTRUCTOR LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE

34

35  scope:

36                  PRIVATE

37          |       PUBLIC

38

39  field:

40          scope datatype ID SEMI

41

42  fname:

43          ID

44

45  fdecl:

46          scope datatype fname LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE

47

48  formals_opt:

49                  /* nothing */

50          |       formal_list

51

52  formal_list:

53                  formal

54          |       formal_list COMMA formal

55

56  formal:

57          datatype ID

58

59  actuals_opt:

60                  /* nothing */

61          |       actuals_list

62

63  actuals_list:

64                  expr

65          |       actuals_list COMMA expr

66

67  primitive:

68                  INT

69          |       FLOAT

70          |       CHAR

71          |       BOOL

72          |       VOID

73

74  name:

75          CLASS ID

76

77  type_tag:

78                  primitive
```

```
79                 |           name
80
81   array_type:
82           type_tag LBRACKET brackets RBRACKET
83
84   datatype:
85                   type_tag
86           |         array_type
87
88   brackets:
89                   /* nothing */
90           |         brackets RBRACKET LBRACKET
91
92   stmt_list:
93                   /* nothing */
94           |         stmt_list stmt
95
96   stmt:
97           expr SEMI
98   |         RETURN expr SEMI
99   |        RETURN SEMI
100  |         LBRACE stmt_list RBRACE
101  |         IF LPAREN expr RPAREN stmt
102  |         IF LPAREN expr RPAREN stmt ELSE stmt
103  |         FOR LPAREN expr_opt SEMI expr_opt SEMI expr_opt RPAREN stmt
104  |         WHILE LPAREN expr RPAREN stmt
105  |        BREAK SEMI
106  |        CONTINUE SEMI
107  |     datatype ID SEMI
108  |         datatype ID ASSIGN expr SEMI
109
110  expr_opt:
111                  /* nothing */
112          |         expr
113
114  expr:
115                  literals
116          |        expr PLUS   expr
117          |        expr MINUS  expr
118          |        expr TIMES  expr
119          |        expr DIVIDE expr
120          |        expr EQ     expr
121          |        expr NEQ    expr
122          |        expr LT     expr
123          |        expr LEQ    expr
124          |        expr GT     expr
125          |        expr GEQ    expr
126          |        expr AND    expr
127          |        expr MODULO expr
```

```
128                  |              NOT   expr
129                  |              expr OR      expr
130                  |              expr DOT     expr
131                  |              expr ASSIGN expr
132                  |              DELETE expr
133             |      MINUS expr
134                  |              ID LPAREN actuals_opt RPAREN
135                  |              NEW ID LPAREN actuals_opt RPAREN
136                  |             NEW type_tag bracket_args RBRACKET
137                  |              expr bracket_args RBRACKET
138                  |              LPAREN expr RPAREN
139
140  bracket_args:
141                  LBRACKET expr
142          |          bracket_args RBRACKET LBRACKET expr
143
144  literals:
145                  INT_LITERAL
146          |          FLOAT_LITERAL
147          |          TRUE
148          |          FALSE
149          |          STRING_LITERAL
150          |          CHAR_LITERAL
151          |          THIS
152          |          ID
153          |          NULL
154          |          BAR array_prim BAR
155
156  array_prim:
157                  expr
158          |           array_prim COMMA expr
```

4. Project Plan

Planning Process

Throughout the project we embodied the principles of agile development. At any point in time during our development we had working code on the master branch and every member of the team was brought up to speed with what has been completed and worked on. All goals for the project were put on Github and as they were resolved they were cleared. We created several milestones which captured our goals for completing the parser, scanner, analyzer, codegen, and final report milestones. We also worked closely with Professor Edwards at Columbia University to receive guidance on how best to implement this language. The following milestones were created and cleared over the course of the semester:
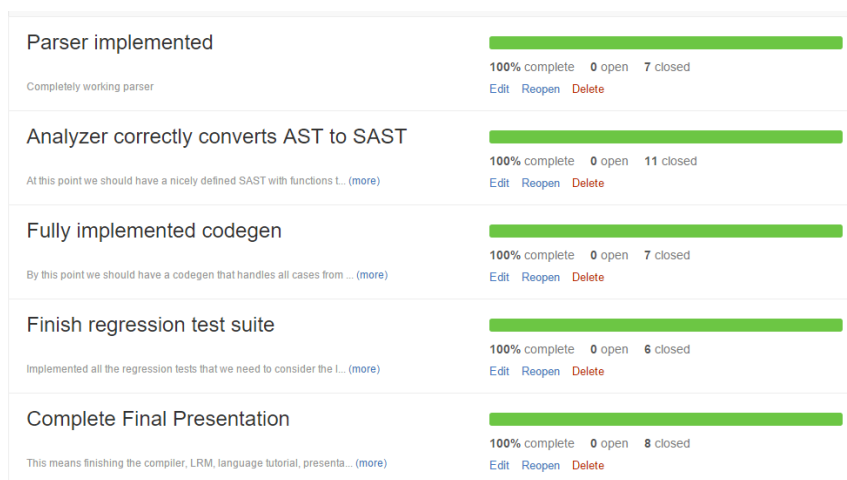
Figure 4.1: Milestoning on Github.

Specification Process

At the beginning of the semester we had originally intended our language to be a distributed software solution that would conveniently allow the developer to distribute tasks to various slave machines that had compiled the tasks to LLVM IR. After discussing this with professor Edwards we then decided to opt for an object oriented programming language that specifically compiled to LLVM IR. This way we as a team could learn more about making compilers and showing the power of LLVM.

Once we decided on the theme of Dice, we met to discuss the features we wanted most in our object oriented language. In our case we wanted arrays, inheritance, objects, and file IO to be some of the key highlights of our language. We then built up the scanner and parser to get a more solid idea as to what the language would look like, and by November 15th we had solidified our plans to implement the aforementioned features.

Development Process

Implementation was very dependent on the course deadlines. We started with the scanner and parser specifically so the language reference manual was better defined. This was completed by October 26th. We then iterated on the analyzer and codegen until it was capable of producing hello world. This was completed on November 15th. The month afterwards was spent implementing inheritance and arrays until they were finally completed on December 18th.

Testing Process

Throughout the development process we had numerous tests. The plan was to always have tests that were non-functional so a feature could then be implemented to get them working. If we encountered an error that we were unsure of how to fix, we added more error messages in our compiler until we could exactly pinpoint where the error was occurring. We also made a rule for our team to handle each and every exception that could occur as a custom error message to be printed out by the compiler.

Team Responsibilities

Team responsibilities were divided up and evenly distributed amongst the four group members. While we could not adhere to a strict division of labor based on group member titles, every member contributed to the codebase.

Team Member	Responsibility
David Watkins	Scanner, Parser, Analyzer, Codegen, Utils, LRM, Final Report, Latex, Code cleanup
Emily Chen	Inheritance in Analyzer, Expression types in Analyzer, LRM
Khaled Atef	Test Suite, Binary and unary expression evaluation in codegen
Phillip Schiffrin	Standard Library, Class map generation

Github Usernames

The following Github usernames correspond to the following group members:

- Emily Chen - six5532one, ec2805

- Khaled Atef - KhaledAtef

- David Watkins - DavidWatkins

- Phillip Schiffrin - nethacker11

Project Log

To demonstrate our timeline we captured the number of git commits over time for our project.
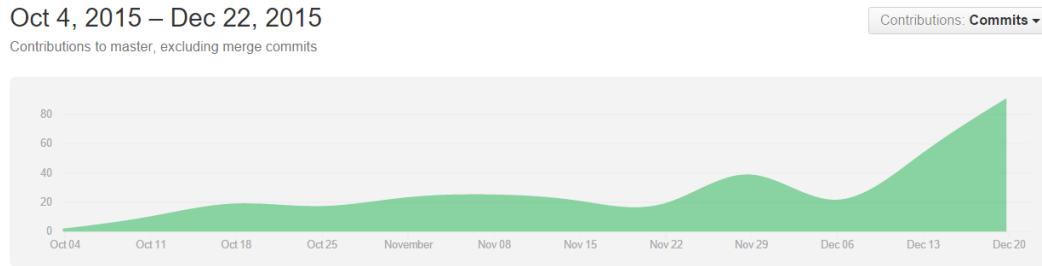
Figure 4.2: Commit timeline on Github.

The timeline shows that we have been diligent at constantly working on the project since the beginning of the semester. All group members have contributed to this project. The following issues are a list of git issues that were cleared as part of our project, as well as the person who closed the issue. We did not have a rule for who closed an issue so sometimes the person who completed the issue may not have been the one to close it.

- #71 Should not be able to access variables outside of scope

- #137 Awesome!

- #134 Subclass assignment [by @six5532one, @DavidWatkins]

- #133 string length tests

- #132 fix delete test, no multiple arrays

- #131 this should raise no exceptions

- #130 Expected stderr: "exception Exceptions.LocalAssignTypeMismatch("B", "C")"

- #129 passing in an inherited class for classes

- #128 E-test-privateFieldsAccess.dice

- #127 Create test for cyclical inheritance

- #126 Add error message for assigning parameters

- #125 test-gcd.dice Bug. You cannot assign values to parameters

- #124 test-constructor1.dice is written incorrectly

- #123 Maximum float is limited to 6 digits after the decimal

- #122 char[][] args does not work in main

- #121 Test max/min floats

- #120 Test default constructor

- #119 Test overloading std-lib functions

- #118 Exit not working in runtime

- #117 Test args

- #116 assign ints to floats

- #115 Integer toString generates string twice

- #114 concat adds an extra character to the string

- #113 Test exit

- #111 Errors.log from script output isn't working properly

- #110 add teststdlib .out

- #109 Add Test returning objects

- #108 add tests for empty blocks

- #107 For inheritance of functions we should have an id to determine which function to call

- #106 Includes should check with String_lit not ID

- #105 Odd invalid numbering of blocks bug

- #104 Fix parameters on library functions

- #103 Get Dice exec working so tests can run again.

- #102 "Get the t-shirts made"

- #101 Adapt codegen to changes in analyzer that add inherited fields to sprogram.classes

- #100 Need to test includes

- #99 add test for empty conditionals

- #98 add empty for loop test

- #97 Add nested comments

- #96 test order of fdecl,fields,constructor in classses

- #95 primitive type limit tests

- #94 test constructors

- #93 test private scope function

- #92 Help needed: env.env_class_maps seems correct but exception is raised when I try to access an inherited field

- #91 default constructors

- #90 Need to add an environment variable to point to the includes

- #89 Strings need to be initialized and accessed differently from normal arrays

- #88 This should raise "UndefinedClass: H"

- #87 Use of Delete

- #86 add static scoping test

- #85 Add applicative order test

- #84 Add delete command to free memory

- #82 Add exit call

- #81 return statements in branches aren't recognized

- #80 dice executable doesn't run without any args

- #79 Kappa [by @DavidWatkins]

- #78 Add tests for recursion

- #77 Obj access [by @DavidWatkins]

- #75 Test invalid functions

- #74 Test multiple classes

- #73 Parent cannot have fields of type of its children

- #72 Cannot call return inside of a constructor

- #135 check for overridden methods takes ret type into account [by @six5532one, @DavidWatkins]

- #69 Casting rules questions

- #68 Kappa [by @DavidWatkins]

- #67 Floats print with extra trailing zeros. Kinda ugly.

- #66 Emily [by @six5532one, @DavidWatkins]

- #65 local decl (primitives): stderr should be "DuplicateLocal: myc"

- #64 object creation: this should raise no exception

- #63 object creation: this should raise no exceptions

- #62 Compiler doesn't allow formal to be an object

- #61 object creation: This should throw no exceptions

- #60 Object creation: this should raise "ConstructorNotFound: Foo.constructor.int.bool.char.float"

- #59 object decl without assignment expr: This should throw no exceptions

- #58 This should throw exception "UndefinedClass: Baz"

- #57 incorrect check for duplicate constructors

- #56 Emily [by @six5532one, @DavidWatkins]

- #55 Create arith tests that have signed values

- #54 Parser issue with reading user-defined objects.

- #53 Emily [by @six5532one]

- #52 Decide whether to promote all ints to floats in binops

- #51 Consecutive print statements don't work. Compiler only outputs first print statement.

- #50 Epsilon [by @six5532one]

- #49 Reorganize object accesses for functions

- #46 Kreygasm [by @DavidWatkins]

- #45 Add shakespeare and stephen number to tester

- #44 Create symbol table for cdecls, fdecls, fields

- #39 static analysis checks for variable access

- #38 use 'new' keyword for object and array instantiation

- #37 support addition of chars and ints

- #36 Update LRM: support addition of chars and ints

- #35 Change parser array create type to type tag and not primitive

- #34 Evaluate whether to add new as a keyword to object initialization

- #33 Exceptions, try, catch?

- #32 Implement basic primitive expressions for codegen

- #31 Should we add continuous checking even when an illegal character/parser error occurs like java?

- #30 Add annotation for source code position to AST

- #29 We should evaluate whether we want to move variable declarations to stmts

- #28 Do we need to add an additional layer of abstraction from SAST to Codegen?

- #27 Complete pretty printing abstract syntax tree to Utils

- #26 How does LLVM handle allocating on the heap

- #24 Strings with escape characters are not being displayed properly

- #23 Create OCamlDoc Documentation

- #22 Should we switch the llvm package to ollvm?

- #21 Add file operator functions to Codegen

- #20 Write the File class

- #19 Write the String class

- #18 Write the Math class

- #17 Add support for utilizing line number and character number in Analyzer

- #16 Add class name and function name collission detection

- #15 Add testing for arrays

- #14 Evaluate the type of an expression in Analyzer.get_expr_type

- #13 Add testing for extends

- #12 Add mentioning of unary minus to LRM

- #11 Remove '-' symbol from regex in floats and ints of LRM

- #10 Convert AST.cdecl to SAST.cdecl

- #9 Convert AST.expr to SAST.expr in Analyzer.convert_expr

- #8 Analyzer.process_includes does not check absolute path

- #7 Delta [by @DavidWatkins]

- #6 Delta [by @DavidWatkins]

- #5 Special chars (tabs/newlines/etc) aren't getting tokenized properly

- #4 float limit

- #3 David fix [by @DavidWatkins]

- #2 Merge pull request #1 from DavidWatkins/DavidFix [by @DavidWatkins]

- #1 David fix [by @DavidWatkins]

Git Commit History

Here are all of the commits as performed by the team. Everyone contributed to the project.

- EDIT ME

Software Development Environment

From the beginning of the project we agreed to the following development environment with the following software versions:

- **Ubuntu 15.10** - Very simple to use linux distribution that had the LLVM software and OCaml software easily accessible. Ubuntu was used within Virtualbox to ensure consistency across hardware as well.

- **LLVM-3.7** - The latest version of LLVM and allowed for easy code generation in OCaml using the LLVM module

- **OCaml Packages** - There were some features, such as JSON manipulation, that required additional OCaml packages. Therefore we included the following four OCaml packages in our development process: core, batteries, llvm, and yojson.

- **Slack** - We agreed that the Slack chat messaging platform was the most convenient and efficient way to share code snippets and communicate. It also brought up morale in the group in the form of emojis.

- **Github** - In order to version control our software and maintain a working version at any time, we used Github as our go to source code repository. It made integration with the team simpler and everyone was able to view the repository conveniently in their browser.

- **Latex** - In order to compile the documentation we made sure to all use Latex to ensure high quality material being produced for the project.

- **Vim/Sublime** - We could not create a consensus on which text editor to use, but in the end it did not matter to much which members used which.

Programming Style Guide

We adhered to the following style guide as much as possible:

- No lines greater than 80 characters

- Ensure that pattern matches are on the same indent with respect to each other

- Use tabbed indentation as opposed to spaces. Ensure that the tab width is 4 spaces.

5. ARCHITECTURE

The Compiler

To give a quick overview of our compiler, we have a total of 8 modules:

- **analyzer.ml** - Semantically checks incoming AST representation to make sure that it includes existing files, adheres to the rules of inheritance, and expressions are properly type-checked

- **codegen.ml** - Converts a semantically checked AST into a working LLVM code by producing LLVM IR

- **dice.ml** - Main module that calls on all the other modules depending on compiler flags passed to it

- **filepath.ml** - Uses system calls to determine the absolute path to any file in the system. Useful for uniquely checking if an include statement refers to the same files

- **parser.mly** - Reads in tokens from the scanner to produce an AST representation of the program

- **processor.ml** - Handles communication between scanner and parser so that error messages regarding invalid input can be handled better

- **scanner.mll** - Reads a source file and tokenizes it to the corresponding token output

- **utils.ml** - Contains several functions for printing out the string representation of various intermediate representations in our language. Most critically used for debugging

and we have 4 interfaces

- **ast.ml** - Representation of program after parser

- **conf.ml** - Contains paths for accessing standard library and bindings

- **exceptions.ml** - All exceptions in the compiler

- **sast.ml** - The semantically checked representation of the language

and we have 2 library files

- **bindings.c** - A c file containing critical functions written in c that are usable in the language. This is compiled to LLVM bitcode and then linked with all source files compiled in our language

- **stdlib.dice** - A file containing user defined classes written in dice that are usable by the user

The Scanner

The Scanner scans through the input file and tokenizes the input, discarding characters which are no longer need such as whitespace.

The Parser

The parser scans the tokens passed to it by the scanner and constructs an abstract syntax tree based on the definitions provided and the input tokens. The top level of the abstract syntax tree is a structure containing all classes and a structure containing all include statements. The Parser produces the following layout:
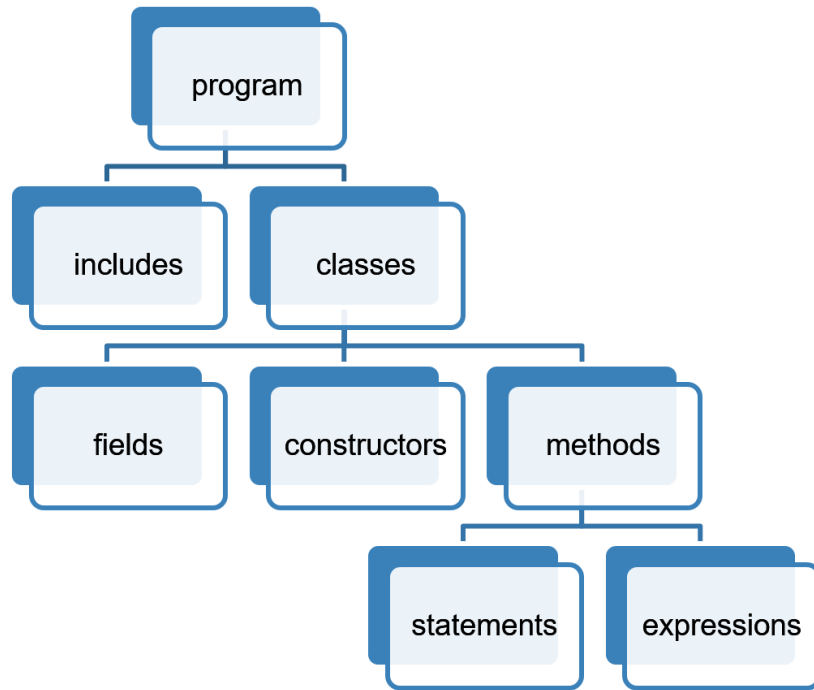
Figure 5.1: AST program representation.

The Semantic Analyzer

The first job of the Analyzer is to run the Scanner and Parser on any files contained in the includes statements of the given abstract syntax tree. The process of building an abstract syntax tree is the same for these files as for the originally compiled file. If any of these new abstract syntax trees contain include statements, the same process is run until there are no more includes. Similarly, each time a new included file's abstract syntax tree is passed to the Analyzer, all classes contained in the class structure of the new abstract syntax tree are appended to the original class list contained in the original class structure which was in the original abstract syntax tree. Once this process is complete, the analyzer is left with a class structure which contains every class defined in every file which was included with the originally compiled file.

Next, the Analyzer performs an inheritance analysis by looking through the class list contained in the class structure and performs an analysis to determine whether any classes are children or parents of other classes. If there are any such relationships, the fields of each parent class are added to the front of its child's fields list, and the methods of each parent class are added to the child's method's list. However, if the child has declared a method or field which shares the same name as the parent's field or method, the child's field or method is not overwritten by the parent. As the inheritance analysis is performed, the list of fields for each class is also assigned a integer key beginning with 0 which will serve as the key to a lookup table which, at runtime, contains pointers to every function for each class.

Once the inheritance analysis is performed, semantic analysis is performed on each statement and expression in each block of code in every method for every class. This semantic analysis consists of making sure that
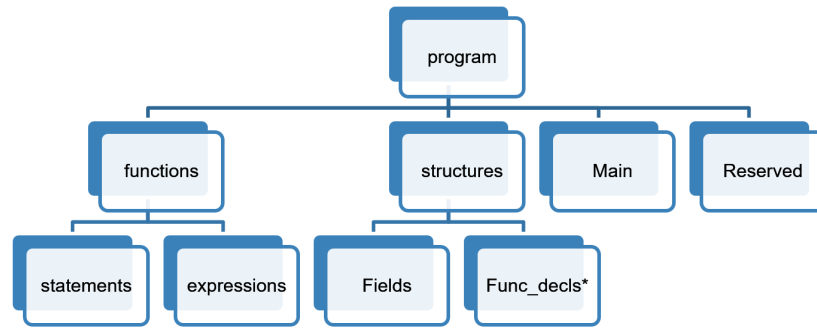
Figure 5.2: SAST representation.

types are consistent in every expression, making sure variables are declared and in the proper scope, and making sure that variables are only declared once. For instance, if an integer x is declared and x is assigned to the return of a method, the analyzer checks that the called method returns the type of x, namely an integer.

As this analysis is performed, the analyzer is simultaneously constructing a semantic abstract syntax tree. The purpose of this new data structure is to provide the code generator with data that is organized more similarly to the LLVM code that it will eventually produce. Thus, instead of classes containing methods and fields, the top level program structure now contains separate sections for methods and fields. This is useful for the code generator because the LLVM code that is produced uses structs to store the fields of a class and functions to store the code within a class's methods. Thus, there is no inherent connection between the functions and the structs in LLVM. However, the analyzer modifies each method so that an instance of the structure containing the fields of the given class is passed in as the first argument to every function for that class. In this way, functions can access each field of a given class by accessing the data inside of the structure.

The Code Generator

The code generator uses the semantic abstract syntax tree passed to it by the analyzer to construct the LLVM IR file which contains the final instructions for the program.

Structs and Inheritance

All structs are given an integer key at the beginning of their definition which will allow them to directly get their own virtual function table. Even if a subclass inherits from a parent class, it will be initialized with a specific key that is unique to the class at the beginning of each struct. For inherited fields they are organized in the order they were inherited, allowing multiple levels of inheritance. However it was too complex of a problem to solve multiple inheritance so we chose not to implement it.
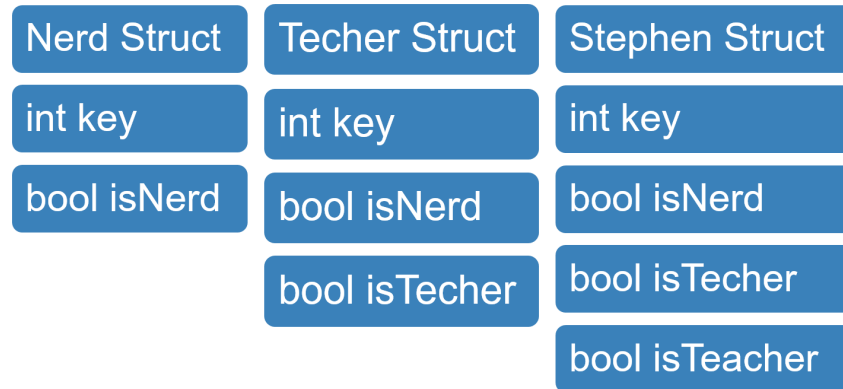
Figure 5.3: Structs example with inheritance.

The Virtual Function Table

At compile time, an intermediate representation of the virtual function table is produced in LLVM IR. It is a function defined as "lookup" that is able to lookup a classes virtual function array by its class index and a function index unique to that function. The function index is generated from the Func_decl list of a struct in the SAST. This way all subclasses have the same index for referring to the same function. Take for example

Class Indexes→	Nerd	Techer	Stephen
Function Indexes→	isNerd:Nerd	isNerd:Nerd	isNerd:Stephen
			isTeacher:Stephen

Figure 5.4: Virtual Function Table Example.

a class Nerd which has a subclass Techer, which itself has a subclass Stephen. Nerd has an isNerd method defined, Techer then inherits that method. Stephen would inherit that method but instead overrides them with its own implementation. But if a Nerd type variable is assigned to a Stephen type variable, the casted struct would still have the corresponding key to the Stephen class, and the function call would receive the correct index of 1 if isNerd were called.

Expressions and Bindings

Once the inheritance code is generated, the code generator iterates through the entire semantic abstract syntax tree and produces the necessary LLVM code for each function, statement, and expression. This code generation is done using the OCaml LLVM library, which uses OCaml functions to produce the desired LLVM code. We then link the resulting LLVM module with a precompiled bindings.bc which allows for the custom C functions we wrote to be incorporated into a user program in LLVM.

The Utilities

Using the utils.ml module we were able to pretty print, print to JSON for AST and SAST, and print out the tokens for any given program. This made debugging the semantic analyzer much easier as we were able to see what went into it and what it produces at any time. The following is an example of what the SAST looks like in JSON.

```json
{
"sprogram": {
      "classes": [
      { "scdecl": { "scname": "test", "sfields": [], "sfuncs": [] } }
      ],
      "functions": [
      {
            "sfdecl": {
                  "sfname": "test.constructor",
                  "sreturnType": "class test",
                  "sformals": [],
                  "sbody": [
                  {
                  "slocal": {
                  "datatype": "class test",
                  "name": "this",
                  "val": {
                        "call": {
                        "name": "cast",
                        "params": [
                        {
                        "call": {
                        "name": "malloc",
                        "params": [
                        {
                              "call": {
                                    "name": "sizeof",
                                    "params": [
                                    {
                                          "id": {
                                                "name": "ignore",
                                                "datatype": "class test"
                                          }
                                    }
                                    ],
                                    "index": 0,
                                    "datatype": "int"
                              }
                        }
                        ],
                        "index": 0,
                        "datatype": "char[]"
                        }
```

```
44                              }
45                              ],
46                              "index": 0,
47                              "datatype": "class test"
48                              }
49                      }
50                      }
51                      },
52                      {
53                      "sexpr": {
54                              "expr": {
55                                      "assign": {
56                                              "lhs": {
57                                                      "objaccess": {
58                                                              "lhs": {
59                                                                      "id": { "name":
                                                                      ↪  "this",
                                                                      ↪  "datatype":
                                                                      ↪  "class test"
                                                                      ↪  }
60                                                              },
61                                                              "op": ".",
62                                                              "rhs": {
63                                                                      "id": { "name":
                                                                      ↪  ".key",
                                                                      ↪  "datatype":
                                                                      ↪  "int" }
64                                                              },
65                                                              "datatype": "int"
66                                                      }
67                                              },
68                                              "op": "=",
69                                              "rhs": { "int_lit": { "val": 0,
                                              ↪  "datatype": "int" } },
70                                              "datatype": "int"
71                                      }
72                              },
73                              "datatype": "int"
74                      }
75                      },
76                      {
77                      "sreturn": {
78                              "return": {
79                                      "id": { "name": "this", "datatype": "class test"
                                      ↪  }
80                              },
81                              "datatype": "class test"
82                      }
83                      }
```

```
84                      ],
85                      "func_type": "user"
86                  }
87          }
88          ],
89      "main": {
90              "sfdecl": {
91                      "sfname": "main",
92                      "sreturnType": "void",
93                      "sformals": [
94                      { "name": "this", "datatype": "class test" },
95                      { "name": "args", "datatype": "char[][]" }
96                      ],
97                      "sbody": [
98                      {
99                      "slocal": {
100                     "datatype": "class test",
101                     "name": "this",
102                     "val": {
103                             "call": {
104                                     "name": "cast",
105                                     "params": [
106                                     {
107                                             "call": {
108                                                     "name": "malloc",
109                                                     "params": [
110                                                     {
111                                                             "call": {
112                                                                     "name": "sizeof",
113                                                                     "params": [
114                                                                     {
115                                                                             "id": {
116                                                                                     "name":
                                                                                 ↪   "ignore",
117                                                                                     "datatype":
                                                                                 ↪   "class
                                                                                 ↪   test"
118                                                                             }
119                                                                     }
120                                                                     ],
121                                                                     "index": 0,
122                                                                     "datatype": "int"
123                                                             }
124                                                     }
125                                                     ],
126                                                     "index": 0,
127                                                     "datatype": "char[]"
128                                             }
129                                     }
```

```
130                                   ],
131                                   "index": 0,
132                                   "datatype": "class test"
133                               }
134                           }
135                       }
136                   },
137                   {
138               "sexpr": {
139               "expr": {
140                       "assign": {
141                       "lhs": {
142                           "objaccess": {
143                               "lhs": {
144                                   "id": { "name": "this",
                                        ↪  "datatype": "class test" }
145                               },
146                               "op": ".",
147                               "rhs": { "id": { "name": ".key",
                                        ↪  "datatype": "int" } },
148                               "datatype": "int"
149                           }
150                       },
151                       "op": "=",
152                       "rhs": { "int_lit": { "val": 0, "datatype": "int" } },
153                       "datatype": "int"
154                       }
155                   },
156               "datatype": "int"
157                   }
158                   },
159                   {
160               "sexpr": {
161               "expr": {
162                       "call": {
163                       "name": "print",
164                       "params": [
165                       {
166                               "string_lit": {
167                                   "val": "Hello, World!",
168                                   "datatype": "char[]"
169                               }
170                       }
171                       ],
172                       "index": 0,
173                       "datatype": "void"
174                       }
175                   },
176               "datatype": "void"
```

```
177                         }
178                         }
179                         ],
180                         "func_type": "user"
181                 }
182         },
183         "reserved": [
184             {
185                 "sfdecl": {
186                         "sfname": "print",
187                         "sreturnType": "void",
188                         "sformals": [ { "Many": "Any" } ],
189                         "sbody": [],
190                         "func_type": "reserved"
191                 }
192         },
193             {
194                 "sfdecl": {
195                         "sfname": "malloc",
196                         "sreturnType": "char[]",
197                         "sformals": [ { "name": "size", "datatype": "int" } ],
198                         "sbody": [],
199                         "func_type": "reserved"
200                 }
201         },
202             {
203                 "sfdecl": {
204                         "sfname": "cast",
205                         "sreturnType": "Any",
206                         "sformals": [ { "name": "in", "datatype": "Any" } ],
207                         "sbody": [],
208                         "func_type": "reserved"
209                 }
210         },
211             {
212                 "sfdecl": {
213                         "sfname": "sizeof",
214                         "sreturnType": "int",
215                         "sformals": [ { "name": "in", "datatype": "Any" } ],
216                         "sbody": [],
217                         "func_type": "reserved"
218                 }
219         },
220             {
221                 "sfdecl": {
222                         "sfname": "open",
223                         "sreturnType": "int",
224                         "sformals": [
225                         { "name": "path", "datatype": "char[]" },
```

```
226                    { "name": "flags", "datatype": "int" }
227                    ],
228                    "sbody": [],
229                    "func_type": "reserved"
230            }
231      },
232      {
233            "sfdecl": {
234                    "sfname": "close",
235                    "sreturnType": "int",
236                    "sformals": [ { "name": "fd", "datatype": "int" } ],
237                    "sbody": [],
238                    "func_type": "reserved"
239            }
240      },
241      {
242            "sfdecl": {
243                    "sfname": "read",
244                    "sreturnType": "int",
245                    "sformals": [
246                    { "name": "fd", "datatype": "int" },
247                    { "name": "buf", "datatype": "char[]" },
248                    { "name": "nbyte", "datatype": "int" }
249                    ],
250                    "sbody": [],
251                    "func_type": "reserved"
252            }
253      },
254      {
255            "sfdecl": {
256                    "sfname": "write",
257                    "sreturnType": "int",
258                    "sformals": [
259                    { "name": "fd", "datatype": "int" },
260                    { "name": "buf", "datatype": "char[]" },
261                    { "name": "nbyte", "datatype": "int" }
262                    ],
263                    "sbody": [],
264                    "func_type": "reserved"
265            }
266      },
267      {
268            "sfdecl": {
269                    "sfname": "lseek",
270                    "sreturnType": "int",
271                    "sformals": [
272                    { "name": "fd", "datatype": "int" },
273                    { "name": "offset", "datatype": "int" },
274                    { "name": "whence", "datatype": "int" }
```

```
275                        ],
276                        "sbody": [],
277                        "func_type": "reserved"
278                }
279        },
280        {
281            "sfdecl": {
282                        "sfname": "exit",
283                        "sreturnType": "void",
284                        "sformals": [ { "name": "status", "datatype": "int" } ],
285                        "sbody": [],
286                        "func_type": "reserved"
287                }
288        },
289        {
290            "sfdecl": {
291                        "sfname": "getchar",
292                        "sreturnType": "int",
293                        "sformals": [],
294                        "sbody": [],
295                        "func_type": "reserved"
296                }
297        },
298        {
299            "sfdecl": {
300                        "sfname": "input",
301                        "sreturnType": "char[]",
302                        "sformals": [],
303                        "sbody": [],
304                        "func_type": "reserved"
305                }
306        }
307        ]
308 }
309 }
```

Supplementary Code

The Standard Library

The standard library was written in order to provide the user with a solid foundation on which to start writing interesting programs. To that end we provide for basic file i/o and string and integer manipulation.

String

Provide useful functionality for string manipulation.

Fields

String has no public fields. Private fields include a char array my_string which stores the given string and an int to store the length of the string.

Constructors

String(char[] a) Accepts a char array, such as a string literal or a char array. This string is copied into the my_string field of the object and the private length() method is run to get the length of the input string.

Methods

private int length_internal(char[] input) Returns the length of the given char array.

private char[] copy_iternal(char[] input) Creates a new char array into which it copies the given char array.

public char[] string() Returns the char array contained in the my_string field.

public char getChar(int index) Returns the char contained at the given index in the my_string field.

public int length() Returns the length of the my_string field

public int toInteger() Converts the char array in the my_string field to an integer and returns that int. If the char array contained in the my_string field is not a string representation of an int, the behavior is undefined.

public int toDigit(char digit) Returns the integer corresponding to the character passed in.

public class String copy(class String input) Returns a copy of the current object.

public int indexOf(char input) Returns the index of the input character in the my_string field. Returns -1 if the character is not found in the field.

public class String reverse() Returns a string object with the my_string field containing the reverse of the current my_string char array.

public class String concat(class String temp) Returns a string object with the my_string field containing the concatenation of the current my_string field with the temp's my_string field.

public bool compare(class String input) Returns true if the my_string field of the input String is equal to the my_string field of the current String object.

public bool contains(class String check) Returns true if the my_string field of the input String is contained in the my_string field of the current String object.

public void free() Frees the memory for the my_string field of the current String object.

File

The File class constructor takes two arguments: a char[] that points to an already opened file on which the user wishes to operate and a boolean indicating whether the user wishes to open the file for writing. If the boolean is true the file is opened for reading and writing, and if false the file is opened as read only. The constructor stores the given path in a field and then calls open() on the given path and, if successful, sets the object's file descriptor field to the return of open(). If open() fails, the program exits with error.

Fields

File has no public fields. Private fields are the class String filePath, private bool isWriteEnabled, and the private int fd.

Constructors

File(char[] path, bool isWriteEnabled)   Accepts a char array to open a file on, then creates a file object with the file descriptor. isWriteEnabled is a parameter that is used to determine whether the file can be written to or just read from.

Methods

private int openfile(class String path, bool isWriteEnabled)   Returns the file descriptor of the opened file if successful, and -1 otherwise.

public char[] readfile(int num)   Reads num bytes from the open file and returns the bytes in a char array.

public int writefile(char[] arr, int offset)   Writes the contents of the char[] array to the file. If offset is -1 the write starts at the beginning of the file, if 0 it starts at the end of the file, and with any other positive integer it starts writing offset bytes from the beginning of the file.

public void closefile()   Closes the open file. On error, the program exits with error.

Integer

The Integer class provides for integers to be converted to char arrays.

Fields

Integer has no public fields. There is one private field my_int which stores the given integer.

Constructors

Integer(int input)   Accepts an integer which is stored in the field my_int.

Methods

public int num()   Returns the integer stored in the my_int field.

public char toChar(int digit)   Returns in teh input digit as a character.

public class String toString()   Converts the integer stored in the my_int field into a string using the toChar() method. Returns a string object.

Built-in Functions

These are functions which are mapped from Dice to the C standard library, which is accessed through LLVM IR. The following function names may not be declared by the user since they are reserved. These are the only functions in dice which are not called as the method of an object; instead the user calls them directly with no dot operator.

int print(...)

The print function can take a char array, int, float and boolean. For char arrays, the contents of the array are printed to stdout. For every other type, the type is converted to the proper variable identifier as used in the C standard library printf function, and then the identifier is replaced with the value of the passed in type when the string is printed to standard out. Arguments can be in any order and must be comma separated.

char[] malloc(int size)

Returns a char pointer to an area of allocated memory on the heap of size bytes.

int open(char[] path)

Attempts to open the file located at the path specified and, if successful, returns a file descriptor to the open file. Returns -1 on failure.

int close(int fd)

Closes the open file identified by the integer fd. Returns 0 if successful and -1 on error.

int read(int fd, char[] buf, int num)

Reads num bytes from the open file identified by fd and stores the resulting string in the char array buf. If successful the number of bytes read is returned. Otherwise returns -1.

int write(int fd, char[] buf, int size)

Writes the contents of the char array buf, which contains size bytes, to the open file identified by fd. If successful the number of bytes written is returned. Otherwise returns -1.

int lseek(int fd, int offset, int whence)

The lseek() function repositions the offset of the open file associated with the file descriptor fd to the argument offset according to the directive whence as follows: 0 - the offset is set to offset bytes, 1 - The offset is set to its current location plus offset bytes, 2 - The offset is set to the size of the file plus offset bytes.

void exit(int flag)

Exits the program. Program exits without error is flag is 0 and exits with error if flag is set to any other integer.

int getchar()

Gets a character from stdin. Returns the character cast to an int.

Functions Implemented in C

With LLVM IR dice is able to compile functions written in C to LLVM. The following functions for dice were written in C.

Declarations

char[] input()

The input function reads from stdin with the C standard library getchar() function, storing each character in a malloc'd char array, until a newline character is read. The resulting array is returned.

long[] init_arr(int[] dims, int dimc)

Takes a list of dimensions in the form of ints and initialize a dimc-dimensional array in a one-dimension malloc call. To access element arr[1][2], first dereference a[1], and cast the value to a long*, which is an address to the array at position 1. Then dereference arr[2] and then cast that to a long* and the value is located at that position. This function is implemented in bindings.c, but was never incorporated directly into the language.

6. TEST PLAN

We embodied a "Test Driven Development" approach while creating our programming language. This process entailed writing tests for specific features of our language before starting to implement them. Every test should start by failing in an automated script and then the script should be executed after every modification to any portion of the compiler (from scanner to code generation). This way the team members would know if any modifications made resulted in other tests failing that had previously passed.

The majority of the test cases in our suite check the code generation through a comparison of print statement outputs from the code and our expected output. We created a test for every component of our language from basic variable declaration and assignment to class inheritance and method overriding. If it's in our language, there's a test case for it.

Testing Phases

Unit Testing

In the beginning of the testing process, we set out to thoroughly check the scanner and parser; however, the course instructor suggested we focus on the overall output of the project because testing end-to-end flow was his recommendation. To simplify checking of the Abstract Syntax Tree (AST) and the semantically checked AST (SAST), our manager created a pretty printer that would output the trees in a Javascript Object Notation (JSON) format for quick visual confirmation of their structure. In addition to quick visual feedback JSON objects provide, we also considered using an OCaml JSON visualization package known as yojson to render a visual tree of the data. We then compared the results of this output to the expected results based on the input.

Integration Testing

In addition to running the test suite routinely, we streamlined creation of new test cases by allowing any member of the team to create a git issue (labeled with "Testing") whenever a test case idea came to mind. Khaled (Test Suite Creator) would then screen all the open testing issues and add/modify the test according to schedule set by the manager.

During the development process, we also realized that in addition to checking proper output from our programs, we should also check if our analyzer was correctly identifying semantically invalid code. For example, if trying to assign a float type number to an integer variable (a feature we do not support), the analyzer should throw the proper exception. We accounted for these cases and placed all the tests in a separate folder with an identifying prefix to easily determine the category of test case.

Automation

Testing was very simple using ./tester.sh. We can verify that a test works individually by running lli on the outputted ll file

Test Suites

We created a total of 121 tests divided into two categories. One checks that the compiler is properly recognizing invalid code. The other checks that the compiler accepts valid code and tests the output program.

Dice to LL IR

The following code examples are dice source files that compile to an associated LLVM IR file.

Hello World Example

The following "Hello, World!" program is the first program we got running in our language.

test-hello.dice

```
1  class test {
2          public void main(char[][] args) {
3                  print("Hello, World!");
4          }
5  }
```

test-hello.ll

```
1  ; ModuleID = 'Dice Codegen'
2  target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
3  target triple = "x86_64-pc-linux-gnu"
4
5  %test = type <{ i32 }>
6
7  @tmp = private unnamed_addr constant [14 x i8] c"Hello, World!\00"
8  @tmp.1 = private unnamed_addr constant [3 x i8] c"%s\00"
9
10 declare i32 @printf(i8*, ...)
11
12 declare noalias i8* @malloc(i32)
13
14 declare i32 @open(i8*, i32)
15
16 declare i32 @close(i32)
17
18 declare i32 @read(i32, i8*, i32)
19
20 declare i32 @write(i32, i8*, i32)
21
22 declare i32 @lseek(i32, i32, i32)
23
24 declare void @exit(i32)
25
26 declare i8* @realloc(i8*, i32)
27
```

```
28   declare i32 @getchar()
29
30   define i64* @lookup(i32 %c_index, i32 %f_index) {
31   entry:
32     %tmp = alloca i64**
33     %tmp1 = alloca i64*, i32 0
34     %tmp2 = getelementptr i64**, i64*** %tmp, i32 0
35     store i64** %tmp1, i64*** %tmp2
36     ret i64* null
37   }
38
39   define %test* @test.constructor() {
40   entry:
41     %this = alloca %test
42     %tmp = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to
        ↪  i32))
43     %tmp1 = bitcast i8* %tmp to %test*
44     %tmp2 = load %test, %test* %tmp1
45     store %test %tmp2, %test* %this
46     %.key = getelementptr inbounds %test, %test* %this, i32 0, i32 0
47     store i32 0, i32* %.key
48     ret %test* %this
49   }
50
51   define i32 @main(i32 %argc, i8** %argv) {
52   entry:
53     %arr_size = add i32 %argc, 1
54     %mallocsize = mul i32 %arr_size, ptrtoint (i1** getelementptr (i1*, i1** null, i32 1)
        ↪  to i32)
55     %malloccall = tail call i8* @malloc(i32 %mallocsize)
56     %args = bitcast i8* %malloccall to i8***
57     %args1 = bitcast i8*** %args to i8**
58     %argc_len = bitcast i8** %args1 to i32*
59     %arr_1 = getelementptr i8*, i8** %args1, i32 1
60     store i32 %argc, i32* %argc_len
61     br label %args.cond
62
63   args.cond:                                        ; preds = %args.init, %entry
64     %counter = phi i32 [ 0, %entry ], [ %tmp, %args.init ]
65     %tmp = add i32 %counter, 1
66     %tmp2 = icmp slt i32 %counter, %argc
67     br i1 %tmp2, label %args.init, label %args.done
68
69   args.init:                                        ; preds = %args.cond
70     %tmp3 = getelementptr i8*, i8** %arr_1, i32 %counter
71     %tmp4 = getelementptr i8*, i8** %argv, i32 %counter
72     %tmp5 = load i8*, i8** %tmp4
73     store i8* %tmp5, i8** %tmp3
74     br label %args.cond
```

```llvm
75
76  args.done:                                         ; preds = %args.cond
77      %this = alloca %test
78      %tmp6 = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to
        ↪  i32))
79      %tmp7 = bitcast i8* %tmp6 to %test*
80      %tmp8 = load %test, %test* %tmp7
81      store %test %tmp8, %test* %this
82      %.key = getelementptr inbounds %test, %test* %this, i32 0, i32 0
83      store i32 0, i32* %.key
84      %tmp9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]*
        ↪  @tmp.1, i32 0, i32 0), i8* getelementptr inbounds ([14 x i8], [14 x i8]* @tmp, i32
        ↪  0, i32 0))
85      ret i32 0
86  }
87
88  ; Function Attrs: nounwind uwtable
89  define i8* @input() #0 {
90      %initial_size = alloca i32, align 4
91      %str = alloca i8*, align 8
92      %index = alloca i32, align 4
93      %tmp = alloca i8, align 1
94      store i32 100, i32* %initial_size, align 4
95      %1 = load i32, i32* %initial_size, align 4
96      %2 = sext i32 %1 to i64
97      %3 = call noalias i8* bitcast (i8* (i32)* @malloc to i8* (i64)*)(i64 %2) #1
98      store i8* %3, i8** %str, align 8
99      store i32 0, i32* %index, align 4
100     store i8 48, i8* %tmp, align 1
101     br label %4
102
103 ; <label>:4                                         ; preds = %20, %0
104     %5 = call i32 @getchar()
105     %6 = trunc i32 %5 to i8
106     store i8 %6, i8* %tmp, align 1
107     %7 = sext i8 %6 to i32
108     %8 = icmp ne i32 %7, 10
109     br i1 %8, label %9, label %27
110
111 ; <label>:9                                         ; preds = %4
112     %10 = load i32, i32* %index, align 4
113     %11 = load i32, i32* %initial_size, align 4
114     %12 = sub nsw i32 %11, 1
115     %13 = icmp sge i32 %10, %12
116     br i1 %13, label %14, label %20
117
118 ; <label>:14                                        ; preds = %9
119     %15 = load i8*, i8** %str, align 8
120     %16 = load i32, i32* %initial_size, align 4
```

```
121    %17 = mul nsw i32 %16, 2
122    store i32 %17, i32* %initial_size, align 4
123    %18 = sext i32 %17 to i64
124    %19 = call i8* bitcast (i8* (i8*, i32)* @realloc to i8* (i8*, i64)*)(i8* %15, i64 %18)
    ↪   #1
125    store i8* %19, i8** %str, align 8
126    br label %20
127
128    ; <label>:20                                    ; preds = %14, %9
129    %21 = load i8, i8* %tmp, align 1
130    %22 = load i32, i32* %index, align 4
131    %23 = add nsw i32 %22, 1
132    store i32 %23, i32* %index, align 4
133    %24 = sext i32 %22 to i64
134    %25 = load i8*, i8** %str, align 8
135    %26 = getelementptr inbounds i8, i8* %25, i64 %24
136    store i8 %21, i8* %26, align 1
137    br label %4
138
139    ; <label>:27                                    ; preds = %4
140    %28 = load i32, i32* %index, align 4
141    %29 = sext i32 %28 to i64
142    %30 = load i8*, i8** %str, align 8
143    %31 = getelementptr inbounds i8, i8* %30, i64 %29
144    store i8 0, i8* %31, align 1
145    %32 = load i8*, i8** %str, align 8
146    ret i8* %32
147  }
148
149  ; Function Attrs: nounwind uwtable
150  define void @rec_init(i64* %arr, i32 %curr_offset, i32* %static_offsets, i32* %indexes,
    ↪   i32* %dims, i32 %dimc, i32 %dim_curr) #0 {
151    %1 = alloca i64*, align 8
152    %2 = alloca i32, align 4
153    %3 = alloca i32*, align 8
154    %4 = alloca i32*, align 8
155    %5 = alloca i32*, align 8
156    %6 = alloca i32, align 4
157    %7 = alloca i32, align 4
158    %static_offset = alloca i32, align 4
159    %dynamic_offset = alloca i32, align 4
160    %i = alloca i32, align 4
161    %tmp = alloca i32, align 4
162    %j = alloca i32, align 4
163    %i1 = alloca i32, align 4
164    %offset = alloca i32, align 4
165    %sub = alloca i64*, align 8
166    store i64* %arr, i64** %1, align 8
167    store i32 %curr_offset, i32* %2, align 4
```

```
168     store i32* %static_offsets, i32** %3, align 8
169     store i32* %indexes, i32** %4, align 8
170     store i32* %dims, i32** %5, align 8
171     store i32 %dimc, i32* %6, align 4
172     store i32 %dim_curr, i32* %7, align 4
173     %8 = load i32, i32* %7, align 4
174     %9 = sext i32 %8 to i64
175     %10 = load i32*, i32** %5, align 8
176     %11 = getelementptr inbounds i32, i32* %10, i64 %9
177     %12 = load i32, i32* %11, align 4
178     %13 = sext i32 %12 to i64
179     %14 = load i32, i32* %2, align 4
180     %15 = sext i32 %14 to i64
181     %16 = load i64*, i64** %1, align 8
182     %17 = getelementptr inbounds i64, i64* %16, i64 %15
183     store i64 %13, i64* %17, align 8
184     %18 = load i32, i32* %7, align 4
185     %19 = add nsw i32 %18, 1
186     %20 = load i32, i32* %6, align 4
187     %21 = icmp sge i32 %19, %20
188     br i1 %21, label %22, label %23
189
190     ; <label>:22                                       ; preds = %0
191     br label %115
192
193     ; <label>:23                                       ; preds = %0
194     %24 = load i32, i32* %7, align 4
195     %25 = sext i32 %24 to i64
196     %26 = load i32*, i32** %3, align 8
197     %27 = getelementptr inbounds i32, i32* %26, i64 %25
198     %28 = load i32, i32* %27, align 4
199     store i32 %28, i32* %static_offset, align 4
200     store i32 0, i32* %dynamic_offset, align 4
201     store i32 0, i32* %i, align 4
202     br label %29
203
204     ; <label>:29                                       ; preds = %60, %23
205     %30 = load i32, i32* %i, align 4
206     %31 = load i32, i32* %7, align 4
207     %32 = icmp slt i32 %30, %31
208     br i1 %32, label %33, label %63
209
210     ; <label>:33                                       ; preds = %29
211     %34 = load i32, i32* %i, align 4
212     %35 = sext i32 %34 to i64
213     %36 = load i32*, i32** %4, align 8
214     %37 = getelementptr inbounds i32, i32* %36, i64 %35
215     %38 = load i32, i32* %37, align 4
216     store i32 %38, i32* %tmp, align 4
```

```
217    %39 = load i32, i32* %i, align 4
218    %40 = add nsw i32 %39, 1
219    store i32 %40, i32* %j, align 4
220    br label %41
221
222  ; <label>:41                                    ; preds = %53, %33
223    %42 = load i32, i32* %j, align 4
224    %43 = load i32, i32* %7, align 4
225    %44 = icmp sle i32 %42, %43
226    br i1 %44, label %45, label %56
227
228  ; <label>:45                                    ; preds = %41
229    %46 = load i32, i32* %j, align 4
230    %47 = sext i32 %46 to i64
231    %48 = load i32*, i32** %5, align 8
232    %49 = getelementptr inbounds i32, i32* %48, i64 %47
233    %50 = load i32, i32* %49, align 4
234    %51 = load i32, i32* %tmp, align 4
235    %52 = mul nsw i32 %51, %50
236    store i32 %52, i32* %tmp, align 4
237    br label %53
238
239  ; <label>:53                                    ; preds = %45
240    %54 = load i32, i32* %j, align 4
241    %55 = add nsw i32 %54, 1
242    store i32 %55, i32* %j, align 4
243    br label %41
244
245  ; <label>:56                                    ; preds = %41
246    %57 = load i32, i32* %tmp, align 4
247    %58 = load i32, i32* %dynamic_offset, align 4
248    %59 = add nsw i32 %58, %57
249    store i32 %59, i32* %dynamic_offset, align 4
250    br label %60
251
252  ; <label>:60                                    ; preds = %56
253    %61 = load i32, i32* %i, align 4
254    %62 = add nsw i32 %61, 1
255    store i32 %62, i32* %i, align 4
256    br label %29
257
258  ; <label>:63                                    ; preds = %29
259    store i32 0, i32* %i1, align 4
260    br label %64
261
262  ; <label>:64                                    ; preds = %112, %63
263    %65 = load i32, i32* %i1, align 4
264    %66 = load i32, i32* %7, align 4
265    %67 = sext i32 %66 to i64
```

```
266    %68 = load i32*, i32** %5, align 8
267    %69 = getelementptr inbounds i32, i32* %68, i64 %67
268    %70 = load i32, i32* %69, align 4
269    %71 = icmp slt i32 %65, %70
270    br i1 %71, label %72, label %115
271
272  ; <label>:72                                         ; preds = %64
273    %73 = load i32, i32* %static_offset, align 4
274    %74 = load i32, i32* %dynamic_offset, align 4
275    %75 = load i32, i32* %i1, align 4
276    %76 = add nsw i32 %74, %75
277    %77 = load i32, i32* %7, align 4
278    %78 = add nsw i32 %77, 1
279    %79 = sext i32 %78 to i64
280    %80 = load i32*, i32** %5, align 8
281    %81 = getelementptr inbounds i32, i32* %80, i64 %79
282    %82 = load i32, i32* %81, align 4
283    %83 = add nsw i32 %82, 1
284    %84 = mul nsw i32 %76, %83
285    %85 = add nsw i32 %73, %84
286    store i32 %85, i32* %offset, align 4
287    %86 = load i64*, i64** %1, align 8
288    %87 = load i32, i32* %offset, align 4
289    %88 = sext i32 %87 to i64
290    %89 = getelementptr inbounds i64, i64* %86, i64 %88
291    store i64* %89, i64** %sub, align 8
292    %90 = load i64*, i64** %sub, align 8
293    %91 = ptrtoint i64* %90 to i64
294    %92 = load i32, i32* %2, align 4
295    %93 = add nsw i32 %92, 1
296    %94 = load i32, i32* %i1, align 4
297    %95 = add nsw i32 %93, %94
298    %96 = sext i32 %95 to i64
299    %97 = load i64*, i64** %1, align 8
300    %98 = getelementptr inbounds i64, i64* %97, i64 %96
301    store i64 %91, i64* %98, align 8
302    %99 = load i32, i32* %i1, align 4
303    %100 = load i32, i32* %7, align 4
304    %101 = sext i32 %100 to i64
305    %102 = load i32*, i32** %4, align 8
306    %103 = getelementptr inbounds i32, i32* %102, i64 %101
307    store i32 %99, i32* %103, align 4
308    %104 = load i64*, i64** %1, align 8
309    %105 = load i32, i32* %offset, align 4
310    %106 = load i32*, i32** %3, align 8
311    %107 = load i32*, i32** %4, align 8
312    %108 = load i32*, i32** %5, align 8
313    %109 = load i32, i32* %6, align 4
314    %110 = load i32, i32* %7, align 4
```

```
315    %111 = add nsw i32 %110, 1
316    call void @rec_init(i64* %104, i32 %105, i32* %106, i32* %107, i32* %108, i32 %109, i32
       ↪  %111)
317    br label %112
318
319  ; <label>:112                                        ; preds = %72
320    %113 = load i32, i32* %i1, align 4
321    %114 = add nsw i32 %113, 1
322    store i32 %114, i32* %i1, align 4
323    br label %64
324
325  ; <label>:115                                        ; preds = %22, %64
326    ret void
327  }
328
329  ; Function Attrs: nounwind uwtable
330  define i64* @init_arr(i32* %dims, i32 %dimc) #0 {
331    %1 = alloca i32*, align 8
332    %2 = alloca i32, align 4
333    %3 = alloca i8*
334    %total = alloca i32, align 4
335    %i = alloca i32, align 4
336    %j = alloca i32, align 4
337    %i1 = alloca i32, align 4
338    %length = alloca i32, align 4
339    %i2 = alloca i32, align 4
340    %tmp = alloca i32, align 4
341    %j3 = alloca i32, align 4
342    %arr = alloca i64*, align 8
343    %i4 = alloca i32, align 4
344    store i32* %dims, i32** %1, align 8
345    store i32 %dimc, i32* %2, align 4
346    %4 = load i32, i32* %2, align 4
347    %5 = zext i32 %4 to i64
348    %6 = call i8* @llvm.stacksave()
349    store i8* %6, i8** %3
350    %7 = alloca i32, i64 %5, align 16
351    store i32 0, i32* %total, align 4
352    store i32 0, i32* %i, align 4
353    br label %8
354
355  ; <label>:8                                           ; preds = %56, %0
356    %9 = load i32, i32* %i, align 4
357    %10 = load i32, i32* %2, align 4
358    %11 = icmp slt i32 %9, %10
359    br i1 %11, label %12, label %59
360
361  ; <label>:12                                          ; preds = %8
362    %13 = load i32, i32* %i, align 4
```

```
363    %14 = sext i32 %13 to i64
364    %15 = getelementptr inbounds i32, i32* %7, i64 %14
365    store i32 1, i32* %15, align 4
366    store i32 0, i32* %j, align 4
367    br label %16
368
369    ; <label>:16                                    ; preds = %31, %12
370    %17 = load i32, i32* %j, align 4
371    %18 = load i32, i32* %i, align 4
372    %19 = icmp slt i32 %17, %18
373    br i1 %19, label %20, label %34
374
375    ; <label>:20                                    ; preds = %16
376    %21 = load i32, i32* %j, align 4
377    %22 = sext i32 %21 to i64
378    %23 = load i32*, i32** %1, align 8
379    %24 = getelementptr inbounds i32, i32* %23, i64 %22
380    %25 = load i32, i32* %24, align 4
381    %26 = load i32, i32* %i, align 4
382    %27 = sext i32 %26 to i64
383    %28 = getelementptr inbounds i32, i32* %7, i64 %27
384    %29 = load i32, i32* %28, align 4
385    %30 = mul nsw i32 %29, %25
386    store i32 %30, i32* %28, align 4
387    br label %31
388
389    ; <label>:31                                    ; preds = %20
390    %32 = load i32, i32* %j, align 4
391    %33 = add nsw i32 %32, 1
392    store i32 %33, i32* %j, align 4
393    br label %16
394
395    ; <label>:34                                    ; preds = %16
396    %35 = load i32, i32* %i, align 4
397    %36 = sext i32 %35 to i64
398    %37 = load i32*, i32** %1, align 8
399    %38 = getelementptr inbounds i32, i32* %37, i64 %36
400    %39 = load i32, i32* %38, align 4
401    %40 = add nsw i32 %39, 1
402    %41 = load i32, i32* %i, align 4
403    %42 = sext i32 %41 to i64
404    %43 = getelementptr inbounds i32, i32* %7, i64 %42
405    %44 = load i32, i32* %43, align 4
406    %45 = mul nsw i32 %44, %40
407    store i32 %45, i32* %43, align 4
408    %46 = load i32, i32* %total, align 4
409    %47 = load i32, i32* %i, align 4
410    %48 = sext i32 %47 to i64
411    %49 = getelementptr inbounds i32, i32* %7, i64 %48
```

```
412    %50 = load i32, i32* %49, align 4
413    %51 = add nsw i32 %50, %46
414    store i32 %51, i32* %49, align 4
415    %52 = load i32, i32* %i, align 4
416    %53 = sext i32 %52 to i64
417    %54 = getelementptr inbounds i32, i32* %7, i64 %53
418    %55 = load i32, i32* %54, align 4
419    store i32 %55, i32* %total, align 4
420    br label %56
421
422    ; <label>:56                                    ; preds = %34
423    %57 = load i32, i32* %i, align 4
424    %58 = add nsw i32 %57, 1
425    store i32 %58, i32* %i, align 4
426    br label %8
427
428    ; <label>:59                                    ; preds = %8
429    %60 = load i32, i32* %2, align 4
430    %61 = zext i32 %60 to i64
431    %62 = alloca i32, i64 %61, align 16
432    store i32 0, i32* %i1, align 4
433    br label %63
434
435    ; <label>:63                                    ; preds = %71, %59
436    %64 = load i32, i32* %i1, align 4
437    %65 = load i32, i32* %2, align 4
438    %66 = icmp slt i32 %64, %65
439    br i1 %66, label %67, label %74
440
441    ; <label>:67                                    ; preds = %63
442    %68 = load i32, i32* %i1, align 4
443    %69 = sext i32 %68 to i64
444    %70 = getelementptr inbounds i32, i32* %62, i64 %69
445    store i32 0, i32* %70, align 4
446    br label %71
447
448    ; <label>:71                                    ; preds = %67
449    %72 = load i32, i32* %i1, align 4
450    %73 = add nsw i32 %72, 1
451    store i32 %73, i32* %i1, align 4
452    br label %63
453
454    ; <label>:74                                    ; preds = %63
455    store i32 0, i32* %length, align 4
456    store i32 0, i32* %i2, align 4
457    br label %75
458
459    ; <label>:75                                    ; preds = %108, %74
460    %76 = load i32, i32* %i2, align 4
```

```
461     %77 = load i32, i32* %2, align 4
462     %78 = icmp slt i32 %76, %77
463     br i1 %78, label %79, label %111
464
465     ; <label>:79                               ; preds = %75
466     store i32 1, i32* %tmp, align 4
467     %80 = load i32, i32* %i2, align 4
468     %81 = sub nsw i32 %80, 1
469     store i32 %81, i32* %j3, align 4
470     br label %82
471
472     ; <label>:82                               ; preds = %93, %79
473     %83 = load i32, i32* %j3, align 4
474     %84 = icmp sge i32 %83, 0
475     br i1 %84, label %85, label %96
476
477     ; <label>:85                               ; preds = %82
478     %86 = load i32, i32* %j3, align 4
479     %87 = sext i32 %86 to i64
480     %88 = load i32*, i32** %1, align 8
481     %89 = getelementptr inbounds i32, i32* %88, i64 %87
482     %90 = load i32, i32* %89, align 4
483     %91 = load i32, i32* %tmp, align 4
484     %92 = mul nsw i32 %91, %90
485     store i32 %92, i32* %tmp, align 4
486     br label %93
487
488     ; <label>:93                               ; preds = %85
489     %94 = load i32, i32* %j3, align 4
490     %95 = add nsw i32 %94, -1
491     store i32 %95, i32* %j3, align 4
492     br label %82
493
494     ; <label>:96                               ; preds = %82
495     %97 = load i32, i32* %i2, align 4
496     %98 = sext i32 %97 to i64
497     %99 = load i32*, i32** %1, align 8
498     %100 = getelementptr inbounds i32, i32* %99, i64 %98
499     %101 = load i32, i32* %100, align 4
500     %102 = add nsw i32 %101, 1
501     %103 = load i32, i32* %tmp, align 4
502     %104 = mul nsw i32 %103, %102
503     store i32 %104, i32* %tmp, align 4
504     %105 = load i32, i32* %tmp, align 4
505     %106 = load i32, i32* %length, align 4
506     %107 = add nsw i32 %106, %105
507     store i32 %107, i32* %length, align 4
508     br label %108
509
```

```
510   ; <label>:108                                    ; preds = %96
511     %109 = load i32, i32* %i2, align 4
512     %110 = add nsw i32 %109, 1
513     store i32 %110, i32* %i2, align 4
514     br label %75
515
516   ; <label>:111                                    ; preds = %75
517     %112 = load i32, i32* %length, align 4
518     %113 = sext i32 %112 to i64
519     %114 = call noalias i8* bitcast (i8* (i32)* @malloc to i8* (i64)*)(i64 %113) #1
520     %115 = bitcast i8* %114 to i64*
521     store i64* %115, i64** %arr, align 8
522     store i32 0, i32* %i4, align 4
523     br label %116
524
525   ; <label>:116                                    ; preds = %125, %111
526     %117 = load i32, i32* %i4, align 4
527     %118 = load i32, i32* %length, align 4
528     %119 = icmp slt i32 %117, %118
529     br i1 %119, label %120, label %128
530
531   ; <label>:120                                    ; preds = %116
532     %121 = load i32, i32* %i4, align 4
533     %122 = sext i32 %121 to i64
534     %123 = load i64*, i64** %arr, align 8
535     %124 = getelementptr inbounds i64, i64* %123, i64 %122
536     store i64 0, i64* %124, align 8
537     br label %125
538
539   ; <label>:125                                    ; preds = %120
540     %126 = load i32, i32* %i4, align 4
541     %127 = add nsw i32 %126, 1
542     store i32 %127, i32* %i4, align 4
543     br label %116
544
545   ; <label>:128                                    ; preds = %116
546     %129 = load i64*, i64** %arr, align 8
547     %130 = load i32*, i32** %1, align 8
548     %131 = load i32, i32* %2, align 4
549     call void @rec_init(i64* %129, i32 0, i32* %7, i32* %62, i32* %130, i32 %131, i32 0)
550     %132 = load i64*, i64** %arr, align 8
551     %133 = load i8*, i8** %3
552     call void @llvm.stackrestore(i8* %133)
553     ret i64* %132
554   }
555
556   ; Function Attrs: nounwind
557   declare i8* @llvm.stacksave() #1
558
```

```
559    ; Function Attrs: nounwind
560    declare void @llvm.stackrestore(i8*) #1
561
562    attributes #0 = { nounwind uwtable "disable-tail-calls"="false"
    ↪    "less-precise-fpmad"="false" "no-frame-pointer-elim"="true"
    ↪    "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false" "no-nans-fp-math"="false"
    ↪    "stack-protector-buffer-size"="8" "target-cpu"="x86-64"
    ↪    "target-features"="+sse,+sse2" "unsafe-fp-math"="false" "use-soft-float"="false" }
563    attributes #1 = { nounwind }
564
565    !llvm.ident = !{!0}
566
567    !0 = !{!"Ubuntu clang version 3.7.0-2ubuntu1 (tags/RELEASE_370/final) (based on LLVM
    ↪    3.7.0)"}
```

Class Extends Example

The following test checks if a child class inherits the parent's fields:

test-classExtends.dice

```
1  class shape {
2          public float xCoord;
3          public float yCoord;
4  }
5
6  class circle extends shape {
7          public float radius;
8  }
9
10  class test {
11          public void main(char[][] args) {
12                  class circle a = new circle();
13                  a.xCoord = 1.5;
14                  print(a.xCoord);
15          }
16  }
```

test-classExtends.ll

```
1          ; ModuleID = 'Dice Codegen'
2          target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
3          target triple = "x86_64-pc-linux-gnu"
4
5          %test = type <{ i32 }>
6          %circle = type <{ i32, double, double, double }>
7          %shape = type <{ i32, double, double }>
8
9          @tmp = private unnamed_addr constant [3 x i8] c"%f\00"
10
11          declare i32 @printf(i8*, ...)
12
13          declare noalias i8* @malloc(i32)
14
15          declare i32 @open(i8*, i32)
16
17          declare i32 @close(i32)
18
19          declare i32 @read(i32, i8*, i32)
20
21          declare i32 @write(i32, i8*, i32)
22
23          declare i32 @lseek(i32, i32, i32)
24
25          declare void @exit(i32)
```

```
26
27          declare i8* @realloc(i8*, i32)
28
29          declare i32 @getchar()
30
31          define i64* @lookup(i32 %c_index, i32 %f_index) {
32          entry:
33          %tmp = alloca i64**, i32 3
34          %tmp1 = alloca i64*, i32 0
35          %tmp2 = getelementptr i64**, i64*** %tmp, i32 2
36          store i64** %tmp1, i64*** %tmp2
37          %tmp3 = alloca i64*, i32 0
38          %tmp4 = getelementptr i64**, i64*** %tmp, i32 1
39          store i64** %tmp3, i64*** %tmp4
40          %tmp5 = alloca i64*, i32 0
41          %tmp6 = getelementptr i64**, i64*** %tmp, i32 0
42          store i64** %tmp5, i64*** %tmp6
43          ret i64* null
44      }
45
46      define %test* @test.constructor() {
47      entry:
48      %this = alloca %test
49      %tmp = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32))
50      %tmp1 = bitcast i8* %tmp to %test*
51      %tmp2 = load %test, %test* %tmp1
52      store %test %tmp2, %test* %this
53      %.key = getelementptr inbounds %test, %test* %this, i32 0, i32 0
54      store i32 2, i32* %.key
55      ret %test* %this
56      }
57
58      define %circle* @circle.constructor() {
59      entry:
60      %this = alloca %circle
61      %tmp = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32))
62      %tmp1 = bitcast i8* %tmp to %circle*
63      %tmp2 = load %circle, %circle* %tmp1
64      store %circle %tmp2, %circle* %this
65      %.key = getelementptr inbounds %circle, %circle* %this, i32 0, i32 0
66      store i32 1, i32* %.key
67      ret %circle* %this
68      }
69
70      define %shape* @shape.constructor() {
71      entry:
72      %this = alloca %shape
73      %tmp = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32))
74      %tmp1 = bitcast i8* %tmp to %shape*
```

```
75   %tmp2 = load %shape, %shape* %tmp1
76   store %shape %tmp2, %shape* %this
77   %.key = getelementptr inbounds %shape, %shape* %this, i32 0, i32 0
78   store i32 0, i32* %.key
79   ret %shape* %this
80   }
81
82   define i32 @main(i32 %argc, i8** %argv) {
83   entry:
84   %arr_size = add i32 %argc, 1
85   %mallocsize = mul i32 %arr_size, ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to
     ↪  i32)
86   %malloccall = tail call i8* @malloc(i32 %mallocsize)
87   %args = bitcast i8* %malloccall to i8***
88   %args1 = bitcast i8*** %args to i8**
89   %argc_len = bitcast i8** %args1 to i32*
90   %arr_1 = getelementptr i8*, i8** %args1, i32 1
91   store i32 %argc, i32* %argc_len
92   br label %args.cond
93
94   args.cond:                                        ; preds = %args.init, %entry
95   %counter = phi i32 [ 0, %entry ], [ %tmp, %args.init ]
96   %tmp = add i32 %counter, 1
97   %tmp2 = icmp slt i32 %counter, %argc
98   br i1 %tmp2, label %args.init, label %args.done
99
100  args.init:                                        ; preds = %args.cond
101  %tmp3 = getelementptr i8*, i8** %arr_1, i32 %counter
102  %tmp4 = getelementptr i8*, i8** %argv, i32 %counter
103  %tmp5 = load i8*, i8** %tmp4
104  store i8* %tmp5, i8** %tmp3
105  br label %args.cond
106
107  args.done:                                        ; preds = %args.cond
108  %this = alloca %test
109  %tmp6 = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to
     ↪  i32))
110  %tmp7 = bitcast i8* %tmp6 to %test*
111  %tmp8 = load %test, %test* %tmp7
112  store %test %tmp8, %test* %this
113  %.key = getelementptr inbounds %test, %test* %this, i32 0, i32 0
114  store i32 2, i32* %.key
115  %a = alloca %circle
116  %tmp9 = call %circle* @circle.constructor()
117  %tmp10 = load %circle, %circle* %tmp9
118  store %circle %tmp10, %circle* %a
119  %xCoord = getelementptr inbounds %circle, %circle* %a, i32 0, i32 2
120  store double 1.500000e+00, double* %xCoord
121  %xCoord11 = getelementptr inbounds %circle, %circle* %a, i32 0, i32 2
```

```
122   %xCoord12 = load double, double* %xCoord11
123   %tmp13 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]*
      ↪  @tmp, i32 0, i32 0), double %xCoord12)
124   ret i32 0
125   }
126
127   ; Function Attrs: nounwind uwtable
128   define i8* @input() #0 {
129         %initial_size = alloca i32, align 4
130         %str = alloca i8*, align 8
131         %index = alloca i32, align 4
132         %tmp = alloca i8, align 1
133         store i32 100, i32* %initial_size, align 4
134         %1 = load i32, i32* %initial_size, align 4
135         %2 = sext i32 %1 to i64
136         %3 = call noalias i8* bitcast (i8* (i32)* @malloc to i8* (i64)*)(i64 %2) #1
137         store i8* %3, i8** %str, align 8
138         store i32 0, i32* %index, align 4
139         store i8 48, i8* %tmp, align 1
140         br label %4
141
142         ; <label>:4                                        ; preds = %20, %0
143         %5 = call i32 @getchar()
144         %6 = trunc i32 %5 to i8
145         store i8 %6, i8* %tmp, align 1
146         %7 = sext i8 %6 to i32
147         %8 = icmp ne i32 %7, 10
148         br i1 %8, label %9, label %27
149
150         ; <label>:9                                        ; preds = %4
151         %10 = load i32, i32* %index, align 4
152         %11 = load i32, i32* %initial_size, align 4
153         %12 = sub nsw i32 %11, 1
154         %13 = icmp sge i32 %10, %12
155         br i1 %13, label %14, label %20
156
157         ; <label>:14                                       ; preds = %9
158         %15 = load i8*, i8** %str, align 8
159         %16 = load i32, i32* %initial_size, align 4
160         %17 = mul nsw i32 %16, 2
161         store i32 %17, i32* %initial_size, align 4
162         %18 = sext i32 %17 to i64
163         %19 = call i8* bitcast (i8* (i8*, i32)* @realloc to i8* (i8*, i64)*)(i8* %15, i64
            ↪  %18) #1
164         store i8* %19, i8** %str, align 8
165         br label %20
166
167         ; <label>:20                                       ; preds = %14, %9
168         %21 = load i8, i8* %tmp, align 1
```

```
169         %22 = load i32, i32* %index, align 4
170         %23 = add nsw i32 %22, 1
171         store i32 %23, i32* %index, align 4
172         %24 = sext i32 %22 to i64
173         %25 = load i8*, i8** %str, align 8
174         %26 = getelementptr inbounds i8, i8* %25, i64 %24
175         store i8 %21, i8* %26, align 1
176         br label %4
177
178         ; <label>:27                                    ; preds = %4
179         %28 = load i32, i32* %index, align 4
180         %29 = sext i32 %28 to i64
181         %30 = load i8*, i8** %str, align 8
182         %31 = getelementptr inbounds i8, i8* %30, i64 %29
183         store i8 0, i8* %31, align 1
184         %32 = load i8*, i8** %str, align 8
185         ret i8* %32
186     }
187
188     ; Function Attrs: nounwind uwtable
189     define void @rec_init(i64* %arr, i32 %curr_offset, i32* %static_offsets, i32* %indexes,
        ↪  i32* %dims, i32 %dimc, i32 %dim_curr) #0 {
190     %1 = alloca i64*, align 8
191     %2 = alloca i32, align 4
192     %3 = alloca i32*, align 8
193     %4 = alloca i32*, align 8
194     %5 = alloca i32*, align 8
195     %6 = alloca i32, align 4
196     %7 = alloca i32, align 4
197     %static_offset = alloca i32, align 4
198     %dynamic_offset = alloca i32, align 4
199     %i = alloca i32, align 4
200     %tmp = alloca i32, align 4
201     %j = alloca i32, align 4
202     %i1 = alloca i32, align 4
203     %offset = alloca i32, align 4
204     %sub = alloca i64*, align 8
205     store i64* %arr, i64** %1, align 8
206     store i32 %curr_offset, i32* %2, align 4
207     store i32* %static_offsets, i32** %3, align 8
208     store i32* %indexes, i32** %4, align 8
209     store i32* %dims, i32** %5, align 8
210     store i32 %dimc, i32* %6, align 4
211     store i32 %dim_curr, i32* %7, align 4
212     %8 = load i32, i32* %7, align 4
213     %9 = sext i32 %8 to i64
214     %10 = load i32*, i32** %5, align 8
215     %11 = getelementptr inbounds i32, i32* %10, i64 %9
216     %12 = load i32, i32* %11, align 4
```

```
217    %13 = sext i32 %12 to i64
218    %14 = load i32, i32* %2, align 4
219    %15 = sext i32 %14 to i64
220    %16 = load i64*, i64** %1, align 8
221    %17 = getelementptr inbounds i64, i64* %16, i64 %15
222    store i64 %13, i64* %17, align 8
223    %18 = load i32, i32* %7, align 4
224    %19 = add nsw i32 %18, 1
225    %20 = load i32, i32* %6, align 4
226    %21 = icmp sge i32 %19, %20
227    br i1 %21, label %22, label %23
228
229    ; <label>:22                                    ; preds = %0
230    br label %115
231
232    ; <label>:23                                    ; preds = %0
233    %24 = load i32, i32* %7, align 4
234    %25 = sext i32 %24 to i64
235    %26 = load i32*, i32** %3, align 8
236    %27 = getelementptr inbounds i32, i32* %26, i64 %25
237    %28 = load i32, i32* %27, align 4
238    store i32 %28, i32* %static_offset, align 4
239    store i32 0, i32* %dynamic_offset, align 4
240    store i32 0, i32* %i, align 4
241    br label %29
242
243    ; <label>:29                                    ; preds = %60, %23
244    %30 = load i32, i32* %i, align 4
245    %31 = load i32, i32* %7, align 4
246    %32 = icmp slt i32 %30, %31
247    br i1 %32, label %33, label %63
248
249    ; <label>:33                                    ; preds = %29
250    %34 = load i32, i32* %i, align 4
251    %35 = sext i32 %34 to i64
252    %36 = load i32*, i32** %4, align 8
253    %37 = getelementptr inbounds i32, i32* %36, i64 %35
254    %38 = load i32, i32* %37, align 4
255    store i32 %38, i32* %tmp, align 4
256    %39 = load i32, i32* %i, align 4
257    %40 = add nsw i32 %39, 1
258    store i32 %40, i32* %j, align 4
259    br label %41
260
261    ; <label>:41                                    ; preds = %53, %33
262    %42 = load i32, i32* %j, align 4
263    %43 = load i32, i32* %7, align 4
264    %44 = icmp sle i32 %42, %43
265    br i1 %44, label %45, label %56
```

```
266
267   ; <label>:45                                    ; preds = %41
268   %46 = load i32, i32* %j, align 4
269   %47 = sext i32 %46 to i64
270   %48 = load i32*, i32** %5, align 8
271   %49 = getelementptr inbounds i32, i32* %48, i64 %47
272   %50 = load i32, i32* %49, align 4
273   %51 = load i32, i32* %tmp, align 4
274   %52 = mul nsw i32 %51, %50
275   store i32 %52, i32* %tmp, align 4
276   br label %53
277
278   ; <label>:53                                    ; preds = %45
279   %54 = load i32, i32* %j, align 4
280   %55 = add nsw i32 %54, 1
281   store i32 %55, i32* %j, align 4
282   br label %41
283
284   ; <label>:56                                    ; preds = %41
285   %57 = load i32, i32* %tmp, align 4
286   %58 = load i32, i32* %dynamic_offset, align 4
287   %59 = add nsw i32 %58, %57
288   store i32 %59, i32* %dynamic_offset, align 4
289   br label %60
290
291   ; <label>:60                                    ; preds = %56
292   %61 = load i32, i32* %i, align 4
293   %62 = add nsw i32 %61, 1
294   store i32 %62, i32* %i, align 4
295   br label %29
296
297   ; <label>:63                                    ; preds = %29
298   store i32 0, i32* %i1, align 4
299   br label %64
300
301   ; <label>:64                                    ; preds = %112, %63
302   %65 = load i32, i32* %i1, align 4
303   %66 = load i32, i32* %7, align 4
304   %67 = sext i32 %66 to i64
305   %68 = load i32*, i32** %5, align 8
306   %69 = getelementptr inbounds i32, i32* %68, i64 %67
307   %70 = load i32, i32* %69, align 4
308   %71 = icmp slt i32 %65, %70
309   br i1 %71, label %72, label %115
310
311   ; <label>:72                                    ; preds = %64
312   %73 = load i32, i32* %static_offset, align 4
313   %74 = load i32, i32* %dynamic_offset, align 4
314   %75 = load i32, i32* %i1, align 4
```

```
315    %76 = add nsw i32 %74, %75
316    %77 = load i32, i32* %7, align 4
317    %78 = add nsw i32 %77, 1
318    %79 = sext i32 %78 to i64
319    %80 = load i32*, i32** %5, align 8
320    %81 = getelementptr inbounds i32, i32* %80, i64 %79
321    %82 = load i32, i32* %81, align 4
322    %83 = add nsw i32 %82, 1
323    %84 = mul nsw i32 %76, %83
324    %85 = add nsw i32 %73, %84
325    store i32 %85, i32* %offset, align 4
326    %86 = load i64*, i64** %1, align 8
327    %87 = load i32, i32* %offset, align 4
328    %88 = sext i32 %87 to i64
329    %89 = getelementptr inbounds i64, i64* %86, i64 %88
330    store i64* %89, i64** %sub, align 8
331    %90 = load i64*, i64** %sub, align 8
332    %91 = ptrtoint i64* %90 to i64
333    %92 = load i32, i32* %2, align 4
334    %93 = add nsw i32 %92, 1
335    %94 = load i32, i32* %i1, align 4
336    %95 = add nsw i32 %93, %94
337    %96 = sext i32 %95 to i64
338    %97 = load i64*, i64** %1, align 8
339    %98 = getelementptr inbounds i64, i64* %97, i64 %96
340    store i64 %91, i64* %98, align 8
341    %99 = load i32, i32* %i1, align 4
342    %100 = load i32, i32* %7, align 4
343    %101 = sext i32 %100 to i64
344    %102 = load i32*, i32** %4, align 8
345    %103 = getelementptr inbounds i32, i32* %102, i64 %101
346    store i32 %99, i32* %103, align 4
347    %104 = load i64*, i64** %1, align 8
348    %105 = load i32, i32* %offset, align 4
349    %106 = load i32*, i32** %3, align 8
350    %107 = load i32*, i32** %4, align 8
351    %108 = load i32*, i32** %5, align 8
352    %109 = load i32, i32* %6, align 4
353    %110 = load i32, i32* %7, align 4
354    %111 = add nsw i32 %110, 1
355    call void @rec_init(i64* %104, i32 %105, i32* %106, i32* %107, i32* %108, i32 %109, i32
       ↪  %111)
356    br label %112
357
358    ; <label>:112                                        ; preds = %72
359    %113 = load i32, i32* %i1, align 4
360    %114 = add nsw i32 %113, 1
361    store i32 %114, i32* %i1, align 4
362    br label %64
```

```
363
364  ; <label>:115                                    ; preds = %22, %64
365  ret void
366  }
367
368  ; Function Attrs: nounwind uwtable
369  define i64* @init_arr(i32* %dims, i32 %dimc) #0 {
370  %1 = alloca i32*, align 8
371  %2 = alloca i32, align 4
372  %3 = alloca i8*
373  %total = alloca i32, align 4
374  %i = alloca i32, align 4
375  %j = alloca i32, align 4
376  %i1 = alloca i32, align 4
377  %length = alloca i32, align 4
378  %i2 = alloca i32, align 4
379  %tmp = alloca i32, align 4
380  %j3 = alloca i32, align 4
381  %arr = alloca i64*, align 8
382  %i4 = alloca i32, align 4
383  store i32* %dims, i32** %1, align 8
384  store i32 %dimc, i32* %2, align 4
385  %4 = load i32, i32* %2, align 4
386  %5 = zext i32 %4 to i64
387  %6 = call i8* @llvm.stacksave()
388  store i8* %6, i8** %3
389  %7 = alloca i32, i64 %5, align 16
390  store i32 0, i32* %total, align 4
391  store i32 0, i32* %i, align 4
392  br label %8
393
394  ; <label>:8                                       ; preds = %56, %0
395  %9 = load i32, i32* %i, align 4
396  %10 = load i32, i32* %2, align 4
397  %11 = icmp slt i32 %9, %10
398  br i1 %11, label %12, label %59
399
400  ; <label>:12                                      ; preds = %8
401  %13 = load i32, i32* %i, align 4
402  %14 = sext i32 %13 to i64
403  %15 = getelementptr inbounds i32, i32* %7, i64 %14
404  store i32 1, i32* %15, align 4
405  store i32 0, i32* %j, align 4
406  br label %16
407
408  ; <label>:16                                      ; preds = %31, %12
409  %17 = load i32, i32* %j, align 4
410  %18 = load i32, i32* %i, align 4
411  %19 = icmp slt i32 %17, %18
```

```llvm
412  br i1 %19, label %20, label %34
413
414  ; <label>:20                                      ; preds = %16
415  %21 = load i32, i32* %j, align 4
416  %22 = sext i32 %21 to i64
417  %23 = load i32*, i32** %1, align 8
418  %24 = getelementptr inbounds i32, i32* %23, i64 %22
419  %25 = load i32, i32* %24, align 4
420  %26 = load i32, i32* %i, align 4
421  %27 = sext i32 %26 to i64
422  %28 = getelementptr inbounds i32, i32* %7, i64 %27
423  %29 = load i32, i32* %28, align 4
424  %30 = mul nsw i32 %29, %25
425  store i32 %30, i32* %28, align 4
426  br label %31
427
428  ; <label>:31                                      ; preds = %20
429  %32 = load i32, i32* %j, align 4
430  %33 = add nsw i32 %32, 1
431  store i32 %33, i32* %j, align 4
432  br label %16
433
434  ; <label>:34                                      ; preds = %16
435  %35 = load i32, i32* %i, align 4
436  %36 = sext i32 %35 to i64
437  %37 = load i32*, i32** %1, align 8
438  %38 = getelementptr inbounds i32, i32* %37, i64 %36
439  %39 = load i32, i32* %38, align 4
440  %40 = add nsw i32 %39, 1
441  %41 = load i32, i32* %i, align 4
442  %42 = sext i32 %41 to i64
443  %43 = getelementptr inbounds i32, i32* %7, i64 %42
444  %44 = load i32, i32* %43, align 4
445  %45 = mul nsw i32 %44, %40
446  store i32 %45, i32* %43, align 4
447  %46 = load i32, i32* %total, align 4
448  %47 = load i32, i32* %i, align 4
449  %48 = sext i32 %47 to i64
450  %49 = getelementptr inbounds i32, i32* %7, i64 %48
451  %50 = load i32, i32* %49, align 4
452  %51 = add nsw i32 %50, %46
453  store i32 %51, i32* %49, align 4
454  %52 = load i32, i32* %i, align 4
455  %53 = sext i32 %52 to i64
456  %54 = getelementptr inbounds i32, i32* %7, i64 %53
457  %55 = load i32, i32* %54, align 4
458  store i32 %55, i32* %total, align 4
459  br label %56
460
```

```
461   ; <label>:56                                    ; preds = %34
462   %57 = load i32, i32* %i, align 4
463   %58 = add nsw i32 %57, 1
464   store i32 %58, i32* %i, align 4
465   br label %8
466
467   ; <label>:59                                    ; preds = %8
468   %60 = load i32, i32* %2, align 4
469   %61 = zext i32 %60 to i64
470   %62 = alloca i32, i64 %61, align 16
471   store i32 0, i32* %i1, align 4
472   br label %63
473
474   ; <label>:63                                    ; preds = %71, %59
475   %64 = load i32, i32* %i1, align 4
476   %65 = load i32, i32* %2, align 4
477   %66 = icmp slt i32 %64, %65
478   br i1 %66, label %67, label %74
479
480   ; <label>:67                                    ; preds = %63
481   %68 = load i32, i32* %i1, align 4
482   %69 = sext i32 %68 to i64
483   %70 = getelementptr inbounds i32, i32* %62, i64 %69
484   store i32 0, i32* %70, align 4
485   br label %71
486
487   ; <label>:71                                    ; preds = %67
488   %72 = load i32, i32* %i1, align 4
489   %73 = add nsw i32 %72, 1
490   store i32 %73, i32* %i1, align 4
491   br label %63
492
493   ; <label>:74                                    ; preds = %63
494   store i32 0, i32* %length, align 4
495   store i32 0, i32* %i2, align 4
496   br label %75
497
498   ; <label>:75                                    ; preds = %108, %74
499   %76 = load i32, i32* %i2, align 4
500   %77 = load i32, i32* %2, align 4
501   %78 = icmp slt i32 %76, %77
502   br i1 %78, label %79, label %111
503
504   ; <label>:79                                    ; preds = %75
505   store i32 1, i32* %tmp, align 4
506   %80 = load i32, i32* %i2, align 4
507   %81 = sub nsw i32 %80, 1
508   store i32 %81, i32* %j3, align 4
509   br label %82
```

```
510
511    ; <label>:82                                    ; preds = %93, %79
512    %83 = load i32, i32* %j3, align 4
513    %84 = icmp sge i32 %83, 0
514    br i1 %84, label %85, label %96
515
516    ; <label>:85                                    ; preds = %82
517    %86 = load i32, i32* %j3, align 4
518    %87 = sext i32 %86 to i64
519    %88 = load i32*, i32** %1, align 8
520    %89 = getelementptr inbounds i32, i32* %88, i64 %87
521    %90 = load i32, i32* %89, align 4
522    %91 = load i32, i32* %tmp, align 4
523    %92 = mul nsw i32 %91, %90
524    store i32 %92, i32* %tmp, align 4
525    br label %93
526
527    ; <label>:93                                    ; preds = %85
528    %94 = load i32, i32* %j3, align 4
529    %95 = add nsw i32 %94, -1
530    store i32 %95, i32* %j3, align 4
531    br label %82
532
533    ; <label>:96                                    ; preds = %82
534    %97 = load i32, i32* %i2, align 4
535    %98 = sext i32 %97 to i64
536    %99 = load i32*, i32** %1, align 8
537    %100 = getelementptr inbounds i32, i32* %99, i64 %98
538    %101 = load i32, i32* %100, align 4
539    %102 = add nsw i32 %101, 1
540    %103 = load i32, i32* %tmp, align 4
541    %104 = mul nsw i32 %103, %102
542    store i32 %104, i32* %tmp, align 4
543    %105 = load i32, i32* %tmp, align 4
544    %106 = load i32, i32* %length, align 4
545    %107 = add nsw i32 %106, %105
546    store i32 %107, i32* %length, align 4
547    br label %108
548
549    ; <label>:108                                   ; preds = %96
550    %109 = load i32, i32* %i2, align 4
551    %110 = add nsw i32 %109, 1
552    store i32 %110, i32* %i2, align 4
553    br label %75
554
555    ; <label>:111                                   ; preds = %75
556    %112 = load i32, i32* %length, align 4
557    %113 = sext i32 %112 to i64
558    %114 = call noalias i8* bitcast (i8* (i32)* @malloc to i8* (i64)*)(i64 %113) #1
```

```
559   %115 = bitcast i8* %114 to i64*
560   store i64* %115, i64** %arr, align 8
561   store i32 0, i32* %i4, align 4
562   br label %116
563
564   ; <label>:116                                    ; preds = %125, %111
565   %117 = load i32, i32* %i4, align 4
566   %118 = load i32, i32* %length, align 4
567   %119 = icmp slt i32 %117, %118
568   br i1 %119, label %120, label %128
569
570   ; <label>:120                                    ; preds = %116
571   %121 = load i32, i32* %i4, align 4
572   %122 = sext i32 %121 to i64
573   %123 = load i64*, i64** %arr, align 8
574   %124 = getelementptr inbounds i64, i64* %123, i64 %122
575   store i64 0, i64* %124, align 8
576   br label %125
577
578   ; <label>:125                                    ; preds = %120
579   %126 = load i32, i32* %i4, align 4
580   %127 = add nsw i32 %126, 1
581   store i32 %127, i32* %i4, align 4
582   br label %116
583
584   ; <label>:128                                    ; preds = %116
585   %129 = load i64*, i64** %arr, align 8
586   %130 = load i32*, i32** %1, align 8
587   %131 = load i32, i32* %2, align 4
588   call void @rec_init(i64* %129, i32 0, i32* %7, i32* %62, i32* %130, i32 %131, i32 0)
589   %132 = load i64*, i64** %arr, align 8
590   %133 = load i8*, i8** %3
591   call void @llvm.stackrestore(i8* %133)
592   ret i64* %132
593   }
594
595   ; Function Attrs: nounwind
596   declare i8* @llvm.stacksave() #1
597
598   ; Function Attrs: nounwind
599   declare void @llvm.stackrestore(i8*) #1
600
601   attributes #0 = { nounwind uwtable "disable-tail-calls"="false"
      ↪   "less-precise-fpmad"="false" "no-frame-pointer-elim"="true"
      ↪   "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false" "no-nans-fp-math"="false"
      ↪   "stack-protector-buffer-size"="8" "target-cpu"="x86-64"
      ↪   "target-features"="+sse,+sse2" "unsafe-fp-math"="false" "use-soft-float"="false" }
602   attributes #1 = { nounwind }
603
```

```
604   !llvm.ident = !{!0}
605
606   !0 = !{!"Ubuntu clang version 3.7.0-2ubuntu1 (tags/RELEASE_370/final) (based on LLVM
      ↪   3.7.0)"}
```

For Loop Test

The following test is the first of several for loop checks. This one ensures that the correct amount of iterations are complete for the specified block within the curly braces:

test-for1.dice

```
1   class test {
2         public void main(char[][] args) {
3                 int i;
4                 for (i = 0 ; i < 5 ; i = i + 1) {
5                         print(i);
6                 }
7                 print(42);
8         }
9   }
```

test-for1.ll

```
1       ; ModuleID = 'Dice Codegen'
2       target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
3       target triple = "x86_64-pc-linux-gnu"
4
5       %test = type <{ i32 }>
6
7       @tmp = private unnamed_addr constant [3 x i8] c"%d\00"
8       @tmp.1 = private unnamed_addr constant [3 x i8] c"%d\00"
9
10      declare i32 @printf(i8*, ...)
11
12      declare noalias i8* @malloc(i32)
13
14      declare i32 @open(i8*, i32)
15
16      declare i32 @close(i32)
17
18      declare i32 @read(i32, i8*, i32)
19
20      declare i32 @write(i32, i8*, i32)
21
22      declare i32 @lseek(i32, i32, i32)
23
24      declare void @exit(i32)
25
26      declare i8* @realloc(i8*, i32)
27
28      declare i32 @getchar()
29
30      define i64* @lookup(i32 %c_index, i32 %f_index) {
31      entry:
```

```llvm
32          %tmp = alloca i64**
33          %tmp1 = alloca i64*, i32 0
34          %tmp2 = getelementptr i64**, i64*** %tmp, i32 0
35          store i64** %tmp1, i64*** %tmp2
36          ret i64* null
37  }
38
39  define %test* @test.constructor() {
40  entry:
41  %this = alloca %test
42  %tmp = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32))
43  %tmp1 = bitcast i8* %tmp to %test*
44  %tmp2 = load %test, %test* %tmp1
45  store %test %tmp2, %test* %this
46  %.key = getelementptr inbounds %test, %test* %this, i32 0, i32 0
47  store i32 0, i32* %.key
48  ret %test* %this
49  }
50
51  define i32 @main(i32 %argc, i8** %argv) {
52  entry:
53  %arr_size = add i32 %argc, 1
54  %mallocsize = mul i32 %arr_size, ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to
        ↪ i32)
55  %malloccall = tail call i8* @malloc(i32 %mallocsize)
56  %args = bitcast i8* %malloccall to i8***
57  %args1 = bitcast i8*** %args to i8**
58  %argc_len = bitcast i8** %args1 to i32*
59  %arr_1 = getelementptr i8*, i8** %args1, i32 1
60  store i32 %argc, i32* %argc_len
61  br label %args.cond
62
63  args.cond:                                      ; preds = %args.init, %entry
64  %counter = phi i32 [ 0, %entry ], [ %tmp, %args.init ]
65  %tmp = add i32 %counter, 1
66  %tmp2 = icmp slt i32 %counter, %argc
67  br i1 %tmp2, label %args.init, label %args.done
68
69  args.init:                                      ; preds = %args.cond
70  %tmp3 = getelementptr i8*, i8** %arr_1, i32 %counter
71  %tmp4 = getelementptr i8*, i8** %argv, i32 %counter
72  %tmp5 = load i8*, i8** %tmp4
73  store i8* %tmp5, i8** %tmp3
74  br label %args.cond
75
76  args.done:                                      ; preds = %args.cond
77  %this = alloca %test
78  %tmp6 = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to
        ↪ i32))
```

```
79    %tmp7 = bitcast i8* %tmp6 to %test*
80    %tmp8 = load %test, %test* %tmp7
81    store %test %tmp8, %test* %this
82    %.key = getelementptr inbounds %test, %test* %this, i32 0, i32 0
83    store i32 0, i32* %.key
84    %i = alloca i32
85    store i32 0, i32* %i
86    br label %cond
87
88    loop:                                              ; preds = %cond
89    %i9 = load i32, i32* %i
90    %tmp10 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]*
      ↪  @tmp, i32 0, i32 0), i32 %i9)
91    br label %inc
92
93    inc:                                               ; preds = %loop
94    %i11 = load i32, i32* %i
95    %addtmp = add i32 %i11, 1
96    store i32 %addtmp, i32* %i
97    br label %cond
98
99    cond:                                              ; preds = %inc, %args.done
100   %i12 = load i32, i32* %i
101   %lesstmp = icmp slt i32 %i12, 5
102   br i1 %lesstmp, label %loop, label %afterloop
103
104   afterloop:                                         ; preds = %cond
105   %tmp13 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]*
      ↪  @tmp.1, i32 0, i32 0), i32 42)
106   ret i32 0
107   }
108
109   ; Function Attrs: nounwind uwtable
110   define i8* @input() #0 {
111          %initial_size = alloca i32, align 4
112          %str = alloca i8*, align 8
113          %index = alloca i32, align 4
114          %tmp = alloca i8, align 1
115          store i32 100, i32* %initial_size, align 4
116          %1 = load i32, i32* %initial_size, align 4
117          %2 = sext i32 %1 to i64
118          %3 = call noalias i8* bitcast (i8* (i32)* @malloc to i8* (i64)*)(i64 %2) #1
119          store i8* %3, i8** %str, align 8
120          store i32 0, i32* %index, align 4
121          store i8 48, i8* %tmp, align 1
122          br label %4
123
124          ; <label>:4                                 ; preds = %20, %0
125          %5 = call i32 @getchar()
```

```
126        %6 = trunc i32 %5 to i8
127        store i8 %6, i8* %tmp, align 1
128        %7 = sext i8 %6 to i32
129        %8 = icmp ne i32 %7, 10
130        br i1 %8, label %9, label %27
131
132        ; <label>:9                                  ; preds = %4
133        %10 = load i32, i32* %index, align 4
134        %11 = load i32, i32* %initial_size, align 4
135        %12 = sub nsw i32 %11, 1
136        %13 = icmp sge i32 %10, %12
137        br i1 %13, label %14, label %20
138
139        ; <label>:14                                 ; preds = %9
140        %15 = load i8*, i8** %str, align 8
141        %16 = load i32, i32* %initial_size, align 4
142        %17 = mul nsw i32 %16, 2
143        store i32 %17, i32* %initial_size, align 4
144        %18 = sext i32 %17 to i64
145        %19 = call i8* bitcast (i8* (i8*, i32)* @realloc to i8* (i8*, i64)*)(i8* %15, i64
           ↪  %18) #1
146        store i8* %19, i8** %str, align 8
147        br label %20
148
149        ; <label>:20                                 ; preds = %14, %9
150        %21 = load i8, i8* %tmp, align 1
151        %22 = load i32, i32* %index, align 4
152        %23 = add nsw i32 %22, 1
153        store i32 %23, i32* %index, align 4
154        %24 = sext i32 %22 to i64
155        %25 = load i8*, i8** %str, align 8
156        %26 = getelementptr inbounds i8, i8* %25, i64 %24
157        store i8 %21, i8* %26, align 1
158        br label %4
159
160        ; <label>:27                                 ; preds = %4
161        %28 = load i32, i32* %index, align 4
162        %29 = sext i32 %28 to i64
163        %30 = load i8*, i8** %str, align 8
164        %31 = getelementptr inbounds i8, i8* %30, i64 %29
165        store i8 0, i8* %31, align 1
166        %32 = load i8*, i8** %str, align 8
167        ret i8* %32
168    }
169
170    ; Function Attrs: nounwind uwtable
171    define void @rec_init(i64* %arr, i32 %curr_offset, i32* %static_offsets, i32* %indexes,
      ↪  i32* %dims, i32 %dimc, i32 %dim_curr) #0 {
172    %1 = alloca i64*, align 8
```

```
173   %2 = alloca i32, align 4
174   %3 = alloca i32*, align 8
175   %4 = alloca i32*, align 8
176   %5 = alloca i32*, align 8
177   %6 = alloca i32, align 4
178   %7 = alloca i32, align 4
179   %static_offset = alloca i32, align 4
180   %dynamic_offset = alloca i32, align 4
181   %i = alloca i32, align 4
182   %tmp = alloca i32, align 4
183   %j = alloca i32, align 4
184   %i1 = alloca i32, align 4
185   %offset = alloca i32, align 4
186   %sub = alloca i64*, align 8
187   store i64* %arr, i64** %1, align 8
188   store i32 %curr_offset, i32* %2, align 4
189   store i32* %static_offsets, i32** %3, align 8
190   store i32* %indexes, i32** %4, align 8
191   store i32* %dims, i32** %5, align 8
192   store i32 %dimc, i32* %6, align 4
193   store i32 %dim_curr, i32* %7, align 4
194   %8 = load i32, i32* %7, align 4
195   %9 = sext i32 %8 to i64
196   %10 = load i32*, i32** %5, align 8
197   %11 = getelementptr inbounds i32, i32* %10, i64 %9
198   %12 = load i32, i32* %11, align 4
199   %13 = sext i32 %12 to i64
200   %14 = load i32, i32* %2, align 4
201   %15 = sext i32 %14 to i64
202   %16 = load i64*, i64** %1, align 8
203   %17 = getelementptr inbounds i64, i64* %16, i64 %15
204   store i64 %13, i64* %17, align 8
205   %18 = load i32, i32* %7, align 4
206   %19 = add nsw i32 %18, 1
207   %20 = load i32, i32* %6, align 4
208   %21 = icmp sge i32 %19, %20
209   br i1 %21, label %22, label %23
210
211   ; <label>:22                                     ; preds = %0
212   br label %115
213
214   ; <label>:23                                     ; preds = %0
215   %24 = load i32, i32* %7, align 4
216   %25 = sext i32 %24 to i64
217   %26 = load i32*, i32** %3, align 8
218   %27 = getelementptr inbounds i32, i32* %26, i64 %25
219   %28 = load i32, i32* %27, align 4
220   store i32 %28, i32* %static_offset, align 4
221   store i32 0, i32* %dynamic_offset, align 4
```

```
222   store i32 0, i32* %i, align 4
223   br label %29
224
225   ; <label>:29                                      ; preds = %60, %23
226   %30 = load i32, i32* %i, align 4
227   %31 = load i32, i32* %7, align 4
228   %32 = icmp slt i32 %30, %31
229   br i1 %32, label %33, label %63
230
231   ; <label>:33                                      ; preds = %29
232   %34 = load i32, i32* %i, align 4
233   %35 = sext i32 %34 to i64
234   %36 = load i32*, i32** %4, align 8
235   %37 = getelementptr inbounds i32, i32* %36, i64 %35
236   %38 = load i32, i32* %37, align 4
237   store i32 %38, i32* %tmp, align 4
238   %39 = load i32, i32* %i, align 4
239   %40 = add nsw i32 %39, 1
240   store i32 %40, i32* %j, align 4
241   br label %41
242
243   ; <label>:41                                      ; preds = %53, %33
244   %42 = load i32, i32* %j, align 4
245   %43 = load i32, i32* %7, align 4
246   %44 = icmp sle i32 %42, %43
247   br i1 %44, label %45, label %56
248
249   ; <label>:45                                      ; preds = %41
250   %46 = load i32, i32* %j, align 4
251   %47 = sext i32 %46 to i64
252   %48 = load i32*, i32** %5, align 8
253   %49 = getelementptr inbounds i32, i32* %48, i64 %47
254   %50 = load i32, i32* %49, align 4
255   %51 = load i32, i32* %tmp, align 4
256   %52 = mul nsw i32 %51, %50
257   store i32 %52, i32* %tmp, align 4
258   br label %53
259
260   ; <label>:53                                      ; preds = %45
261   %54 = load i32, i32* %j, align 4
262   %55 = add nsw i32 %54, 1
263   store i32 %55, i32* %j, align 4
264   br label %41
265
266   ; <label>:56                                      ; preds = %41
267   %57 = load i32, i32* %tmp, align 4
268   %58 = load i32, i32* %dynamic_offset, align 4
269   %59 = add nsw i32 %58, %57
270   store i32 %59, i32* %dynamic_offset, align 4
```

```
271   br label %60
272
273   ; <label>:60                                      ; preds = %56
274   %61 = load i32, i32* %i, align 4
275   %62 = add nsw i32 %61, 1
276   store i32 %62, i32* %i, align 4
277   br label %29
278
279   ; <label>:63                                      ; preds = %29
280   store i32 0, i32* %i1, align 4
281   br label %64
282
283   ; <label>:64                                      ; preds = %112, %63
284   %65 = load i32, i32* %i1, align 4
285   %66 = load i32, i32* %7, align 4
286   %67 = sext i32 %66 to i64
287   %68 = load i32*, i32** %5, align 8
288   %69 = getelementptr inbounds i32, i32* %68, i64 %67
289   %70 = load i32, i32* %69, align 4
290   %71 = icmp slt i32 %65, %70
291   br i1 %71, label %72, label %115
292
293   ; <label>:72                                      ; preds = %64
294   %73 = load i32, i32* %static_offset, align 4
295   %74 = load i32, i32* %dynamic_offset, align 4
296   %75 = load i32, i32* %i1, align 4
297   %76 = add nsw i32 %74, %75
298   %77 = load i32, i32* %7, align 4
299   %78 = add nsw i32 %77, 1
300   %79 = sext i32 %78 to i64
301   %80 = load i32*, i32** %5, align 8
302   %81 = getelementptr inbounds i32, i32* %80, i64 %79
303   %82 = load i32, i32* %81, align 4
304   %83 = add nsw i32 %82, 1
305   %84 = mul nsw i32 %76, %83
306   %85 = add nsw i32 %73, %84
307   store i32 %85, i32* %offset, align 4
308   %86 = load i64*, i64** %1, align 8
309   %87 = load i32, i32* %offset, align 4
310   %88 = sext i32 %87 to i64
311   %89 = getelementptr inbounds i64, i64* %86, i64 %88
312   store i64* %89, i64** %sub, align 8
313   %90 = load i64*, i64** %sub, align 8
314   %91 = ptrtoint i64* %90 to i64
315   %92 = load i32, i32* %2, align 4
316   %93 = add nsw i32 %92, 1
317   %94 = load i32, i32* %i1, align 4
318   %95 = add nsw i32 %93, %94
319   %96 = sext i32 %95 to i64
```

```llvm
320  %97 = load i64*, i64** %1, align 8
321  %98 = getelementptr inbounds i64, i64* %97, i64 %96
322  store i64 %91, i64* %98, align 8
323  %99 = load i32, i32* %i1, align 4
324  %100 = load i32, i32* %7, align 4
325  %101 = sext i32 %100 to i64
326  %102 = load i32*, i32** %4, align 8
327  %103 = getelementptr inbounds i32, i32* %102, i64 %101
328  store i32 %99, i32* %103, align 4
329  %104 = load i64*, i64** %1, align 8
330  %105 = load i32, i32* %offset, align 4
331  %106 = load i32*, i32** %3, align 8
332  %107 = load i32*, i32** %4, align 8
333  %108 = load i32*, i32** %5, align 8
334  %109 = load i32, i32* %6, align 4
335  %110 = load i32, i32* %7, align 4
336  %111 = add nsw i32 %110, 1
337  call void @rec_init(i64* %104, i32 %105, i32* %106, i32* %107, i32* %108, i32 %109, i32
     ↪  %111)
338  br label %112
339
340  ; <label>:112                                          ; preds = %72
341  %113 = load i32, i32* %i1, align 4
342  %114 = add nsw i32 %113, 1
343  store i32 %114, i32* %i1, align 4
344  br label %64
345
346  ; <label>:115                                          ; preds = %22, %64
347  ret void
348  }
349
350  ; Function Attrs: nounwind uwtable
351  define i64* @init_arr(i32* %dims, i32 %dimc) #0 {
352  %1 = alloca i32*, align 8
353  %2 = alloca i32, align 4
354  %3 = alloca i8*
355  %total = alloca i32, align 4
356  %i = alloca i32, align 4
357  %j = alloca i32, align 4
358  %i1 = alloca i32, align 4
359  %length = alloca i32, align 4
360  %i2 = alloca i32, align 4
361  %tmp = alloca i32, align 4
362  %j3 = alloca i32, align 4
363  %arr = alloca i64*, align 8
364  %i4 = alloca i32, align 4
365  store i32* %dims, i32** %1, align 8
366  store i32 %dimc, i32* %2, align 4
367  %4 = load i32, i32* %2, align 4
```

```
368   %5 = zext i32 %4 to i64
369   %6 = call i8* @llvm.stacksave()
370   store i8* %6, i8** %3
371   %7 = alloca i32, i64 %5, align 16
372   store i32 0, i32* %total, align 4
373   store i32 0, i32* %i, align 4
374   br label %8
375
376   ; <label>:8                                          ; preds = %56, %0
377   %9 = load i32, i32* %i, align 4
378   %10 = load i32, i32* %2, align 4
379   %11 = icmp slt i32 %9, %10
380   br i1 %11, label %12, label %59
381
382   ; <label>:12                                         ; preds = %8
383   %13 = load i32, i32* %i, align 4
384   %14 = sext i32 %13 to i64
385   %15 = getelementptr inbounds i32, i32* %7, i64 %14
386   store i32 1, i32* %15, align 4
387   store i32 0, i32* %j, align 4
388   br label %16
389
390   ; <label>:16                                         ; preds = %31, %12
391   %17 = load i32, i32* %j, align 4
392   %18 = load i32, i32* %i, align 4
393   %19 = icmp slt i32 %17, %18
394   br i1 %19, label %20, label %34
395
396   ; <label>:20                                         ; preds = %16
397   %21 = load i32, i32* %j, align 4
398   %22 = sext i32 %21 to i64
399   %23 = load i32*, i32** %1, align 8
400   %24 = getelementptr inbounds i32, i32* %23, i64 %22
401   %25 = load i32, i32* %24, align 4
402   %26 = load i32, i32* %i, align 4
403   %27 = sext i32 %26 to i64
404   %28 = getelementptr inbounds i32, i32* %7, i64 %27
405   %29 = load i32, i32* %28, align 4
406   %30 = mul nsw i32 %29, %25
407   store i32 %30, i32* %28, align 4
408   br label %31
409
410   ; <label>:31                                         ; preds = %20
411   %32 = load i32, i32* %j, align 4
412   %33 = add nsw i32 %32, 1
413   store i32 %33, i32* %j, align 4
414   br label %16
415
416   ; <label>:34                                         ; preds = %16
```

```
417   %35 = load i32, i32* %i, align 4
418   %36 = sext i32 %35 to i64
419   %37 = load i32*, i32** %1, align 8
420   %38 = getelementptr inbounds i32, i32* %37, i64 %36
421   %39 = load i32, i32* %38, align 4
422   %40 = add nsw i32 %39, 1
423   %41 = load i32, i32* %i, align 4
424   %42 = sext i32 %41 to i64
425   %43 = getelementptr inbounds i32, i32* %7, i64 %42
426   %44 = load i32, i32* %43, align 4
427   %45 = mul nsw i32 %44, %40
428   store i32 %45, i32* %43, align 4
429   %46 = load i32, i32* %total, align 4
430   %47 = load i32, i32* %i, align 4
431   %48 = sext i32 %47 to i64
432   %49 = getelementptr inbounds i32, i32* %7, i64 %48
433   %50 = load i32, i32* %49, align 4
434   %51 = add nsw i32 %50, %46
435   store i32 %51, i32* %49, align 4
436   %52 = load i32, i32* %i, align 4
437   %53 = sext i32 %52 to i64
438   %54 = getelementptr inbounds i32, i32* %7, i64 %53
439   %55 = load i32, i32* %54, align 4
440   store i32 %55, i32* %total, align 4
441   br label %56
442
443   ; <label>:56                                   ; preds = %34
444   %57 = load i32, i32* %i, align 4
445   %58 = add nsw i32 %57, 1
446   store i32 %58, i32* %i, align 4
447   br label %8
448
449   ; <label>:59                                   ; preds = %8
450   %60 = load i32, i32* %2, align 4
451   %61 = zext i32 %60 to i64
452   %62 = alloca i32, i64 %61, align 16
453   store i32 0, i32* %i1, align 4
454   br label %63
455
456   ; <label>:63                                   ; preds = %71, %59
457   %64 = load i32, i32* %i1, align 4
458   %65 = load i32, i32* %2, align 4
459   %66 = icmp slt i32 %64, %65
460   br i1 %66, label %67, label %74
461
462   ; <label>:67                                   ; preds = %63
463   %68 = load i32, i32* %i1, align 4
464   %69 = sext i32 %68 to i64
465   %70 = getelementptr inbounds i32, i32* %62, i64 %69
```

```
466   store i32 0, i32* %70, align 4
467   br label %71
468
469   ; <label>:71                                    ; preds = %67
470   %72 = load i32, i32* %i1, align 4
471   %73 = add nsw i32 %72, 1
472   store i32 %73, i32* %i1, align 4
473   br label %63
474
475   ; <label>:74                                    ; preds = %63
476   store i32 0, i32* %length, align 4
477   store i32 0, i32* %i2, align 4
478   br label %75
479
480   ; <label>:75                                    ; preds = %108, %74
481   %76 = load i32, i32* %i2, align 4
482   %77 = load i32, i32* %2, align 4
483   %78 = icmp slt i32 %76, %77
484   br i1 %78, label %79, label %111
485
486   ; <label>:79                                    ; preds = %75
487   store i32 1, i32* %tmp, align 4
488   %80 = load i32, i32* %i2, align 4
489   %81 = sub nsw i32 %80, 1
490   store i32 %81, i32* %j3, align 4
491   br label %82
492
493   ; <label>:82                                    ; preds = %93, %79
494   %83 = load i32, i32* %j3, align 4
495   %84 = icmp sge i32 %83, 0
496   br i1 %84, label %85, label %96
497
498   ; <label>:85                                    ; preds = %82
499   %86 = load i32, i32* %j3, align 4
500   %87 = sext i32 %86 to i64
501   %88 = load i32*, i32** %1, align 8
502   %89 = getelementptr inbounds i32, i32* %88, i64 %87
503   %90 = load i32, i32* %89, align 4
504   %91 = load i32, i32* %tmp, align 4
505   %92 = mul nsw i32 %91, %90
506   store i32 %92, i32* %tmp, align 4
507   br label %93
508
509   ; <label>:93                                    ; preds = %85
510   %94 = load i32, i32* %j3, align 4
511   %95 = add nsw i32 %94, -1
512   store i32 %95, i32* %j3, align 4
513   br label %82
514
```

```
515   ; <label>:96                                    ; preds = %82
516   %97 = load i32, i32* %i2, align 4
517   %98 = sext i32 %97 to i64
518   %99 = load i32*, i32** %1, align 8
519   %100 = getelementptr inbounds i32, i32* %99, i64 %98
520   %101 = load i32, i32* %100, align 4
521   %102 = add nsw i32 %101, 1
522   %103 = load i32, i32* %tmp, align 4
523   %104 = mul nsw i32 %103, %102
524   store i32 %104, i32* %tmp, align 4
525   %105 = load i32, i32* %tmp, align 4
526   %106 = load i32, i32* %length, align 4
527   %107 = add nsw i32 %106, %105
528   store i32 %107, i32* %length, align 4
529   br label %108
530
531   ; <label>:108                                   ; preds = %96
532   %109 = load i32, i32* %i2, align 4
533   %110 = add nsw i32 %109, 1
534   store i32 %110, i32* %i2, align 4
535   br label %75
536
537   ; <label>:111                                   ; preds = %75
538   %112 = load i32, i32* %length, align 4
539   %113 = sext i32 %112 to i64
540   %114 = call noalias i8* bitcast (i8* (i32)* @malloc to i8* (i64)*)(i64 %113) #1
541   %115 = bitcast i8* %114 to i64*
542   store i64* %115, i64** %arr, align 8
543   store i32 0, i32* %i4, align 4
544   br label %116
545
546   ; <label>:116                                   ; preds = %125, %111
547   %117 = load i32, i32* %i4, align 4
548   %118 = load i32, i32* %length, align 4
549   %119 = icmp slt i32 %117, %118
550   br i1 %119, label %120, label %128
551
552   ; <label>:120                                   ; preds = %116
553   %121 = load i32, i32* %i4, align 4
554   %122 = sext i32 %121 to i64
555   %123 = load i64*, i64** %arr, align 8
556   %124 = getelementptr inbounds i64, i64* %123, i64 %122
557   store i64 0, i64* %124, align 8
558   br label %125
559
560   ; <label>:125                                   ; preds = %120
561   %126 = load i32, i32* %i4, align 4
562   %127 = add nsw i32 %126, 1
563   store i32 %127, i32* %i4, align 4
```

```
564    br label %116
565
566    ; <label>:128                                    ; preds = %116
567    %129 = load i64*, i64** %arr, align 8
568    %130 = load i32*, i32** %1, align 8
569    %131 = load i32, i32* %2, align 4
570    call void @rec_init(i64* %129, i32 0, i32* %7, i32* %62, i32* %130, i32 %131, i32 0)
571    %132 = load i64*, i64** %arr, align 8
572    %133 = load i8*, i8** %3
573    call void @llvm.stackrestore(i8* %133)
574    ret i64* %132
575    }
576
577    ; Function Attrs: nounwind
578    declare i8* @llvm.stacksave() #1
579
580    ; Function Attrs: nounwind
581    declare void @llvm.stackrestore(i8*) #1
582
583    attributes #0 = { nounwind uwtable "disable-tail-calls"="false"
       ↪   "less-precise-fpmad"="false" "no-frame-pointer-elim"="true"
       ↪   "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false" "no-nans-fp-math"="false"
       ↪   "stack-protector-buffer-size"="8" "target-cpu"="x86-64"
       ↪   "target-features"="+sse,+sse2" "unsafe-fp-math"="false" "use-soft-float"="false" }
584    attributes #1 = { nounwind }
585
586    !llvm.ident = !{!0}
587
588    !0 = !{!"Ubuntu clang version 3.7.0-2ubuntu1 (tags/RELEASE_370/final) (based on LLVM
       ↪   3.7.0)"}
```

7. Lessons Learned

David

Most critically I learned that if you want to make something good, put as much effort as physically possible into it. I was told frequently "get started early" with respect to this project. After starting early I also learned that working often and with purpose helped not only myself get through the project but also the rest of my team.

As project manager the most critical decision I made was to gain consensus on the development environment that each team member was using. My main takeaway was to make sure that everyone agrees to use the same tools and systems. Having incompatible hardware/software can create unnecessary tension in what is already a stressful situation.

One final note is that I really did not know what to expect from OCaml coming into this class. It seemed very mysterious at first, but after looking through previous examples of compilers from other groups and writing out the Analyzer for my language, I quickly grew to enjoy the language. It certainly was not as daunting as it seemed at first.

Emily

Khaled

Read the lessons learned from previous projects and prioritize (with your group) which of them you will implement. You will not be able to do them all, but if you can agree as a group on which mistakes you can avoid, you're already ahead. For our group, we determined that we will ACTUALLY start early, which we we did.

Fortunately, we had a very organized and decisive manager that made sure we were all on track throughout the semester. Make sure you nominate a person with same qualities if you don't want to spend the last week of the semester pulling all-nighters for this project (save that for your other exams).

Track tasks with Github's issue tracking. Keep this issue tracker open during meetings with the Professor/TAs in order to avoid forgetting discussed to-do items. Ensure the manager of the group delegates through this system.

To spare your team members pain, don't use the diff command's output in your test script. Just label the program's output and your expected output and place them on top of each other for easy reading.

Phillip

8. Code Listing

_tags

```
1   <filepath.*> or <*/*.native> or <*/*.byte>: package(unix)
```

analyzer.ml

```
1   open Sast
2   open Ast
3   open Processor
4   open Utils
5   open Filepath
6   open Conf
7
8   module StringMap = Map.Make (String)
9
10  module StringSet = Set.Make (String)
11
12  let struct_indexes:(string, int) Hashtbl.t = Hashtbl.create 10
13  let predecessors:(string, string list) Hashtbl.t = Hashtbl.create 10
14
15  module SS = Set.Make(
16  struct
17  let compare = Pervasives.compare
18  type t = datatype
19  end )
20
21  type class_map = {
22          field_map        : Ast.field StringMap.t;
23          func_map         : Ast.func_decl StringMap.t;
24          constructor_map  : Ast.func_decl StringMap.t;
25          reserved_map         : sfunc_decl StringMap.t;
26          cdecl                       : Ast.class_decl;
27  }
28
29  type env = {
30          env_class_maps: class_map StringMap.t;
31          env_name       : string;
32          env_cmap            : class_map;
33          env_locals     : datatype StringMap.t;
34          env_parameters: Ast.formal StringMap.t;
35          env_returnType: datatype;
36          env_in_for     : bool;
37          env_in_while   : bool;
38          env_reserved   : sfunc_decl list;
39  }
40
41  let update_env_name env env_name =
42  {
43          env_class_maps = env.env_class_maps;
44          env_name       = env_name;
45          env_cmap            = env.env_cmap;
46          env_locals     = env.env_locals;
47          env_parameters = env.env_parameters;
```

```
48          env_returnType = env.env_returnType;
49          env_in_for     = env.env_in_for;
50          env_in_while   = env.env_in_while;
51          env_reserved   = env.env_reserved;
52  }
53
54  let update_call_stack env in_for in_while =
55  {
56          env_class_maps = env.env_class_maps;
57          env_name       = env.env_name;
58          env_cmap           = env.env_cmap;
59          env_locals     = env.env_locals;
60          env_parameters = env.env_parameters;
61          env_returnType = env.env_returnType;
62          env_in_for     = in_for;
63          env_in_while   = in_while;
64          env_reserved   = env.env_reserved;
65  }
66
67  let append_code_to_constructor fbody cname ret_type =
68  let key = Hashtbl.find struct_indexes cname in
69  let init_this = [SLocal(
70  ret_type,
71  "this",
72  SCall(        "cast",
73  [SCall("malloc",
74  [
75  SCall("sizeof", [SId("ignore", ret_type)], Datatype(Int_t), 0)
76  ],
77  Arraytype(Char_t, 1), 0)
78  ],
79  ret_type,
80  0
81  )
82  );
83  SExpr(
84  SAssign(
85  SObjAccess(
86  SId("this", ret_type),
87  SId(".key", Datatype(Int_t)),
88  Datatype(Int_t)
89  ),
90  SInt_Lit(key),
91  Datatype(Int_t)
92  ),
93  Datatype(Int_t)
94  )
95  ]
96  in
```

```
97   let ret_this =
98   [
99   SReturn(
100  SId("this", ret_type),
101  ret_type
102  )
103  ]
104  in
105  (* Need to check for duplicate default constructs *)
106  (* Also need to add malloc around other constructors *)
107  init_this @ fbody @ ret_this
108
109  let default_constructor_body cname =
110  let ret_type = Datatype(Objecttype(cname)) in
111  let fbody = [] in
112  append_code_to_constructor fbody cname ret_type
113
114  let default_sc cname =
115  {
116          sfname                          = Ast.FName (cname ^ "." ^ "constructor");
117          sreturnType         = Datatype(Objecttype(cname));
118          sformals                = [];
119          sbody                           = default_constructor_body cname;
120          func_type              = Sast.User;
121          overrides        = false;
122          source                          = "NA";
123  }
124
125  let default_c cname =
126  {
127          scope                           = Ast.Public;
128          fname                           = Ast.Constructor;
129          returnType                  = Datatype(ConstructorType);
130          formals               = [];
131          body                          = [];
132          overrides                = false;
133          root_cname                   = None;
134  }
135
136  let process_includes filename includes classes =
137  (* Bring in each include  *)
138  let processInclude include_statement =
139  let file_in = open_in include_statement in
140  let lexbuf = Lexing.from_channel file_in in
141  let token_list = Processor.build_token_list lexbuf in
142  let program = Processor.parser include_statement token_list in
143  ignore(close_in file_in);
144  program
145  in
```

```
146  let rec iterate_includes classes m = function
147  [] -> classes
148  | (Include h) :: t ->
149  let h = if h = "stdlib" then Conf.stdlib_path else h in
150  (* Check each include against the map *)
151  let realpath = Filepath.realpath h in
152  if StringMap.mem realpath m then
153  iterate_includes (classes) (m) (t)
154  else
155  let result = processInclude realpath in
156  match result with Program(i,c) ->
157  iterate_includes (classes @ c) (StringMap.add realpath 1 m) (i @ t)
158  in
159  iterate_includes classes (StringMap.add (Filepath.realpath filename) 1 StringMap.empty)
     ↪   includes
160
161  let get_name cname fdecl =
162  (* We use '.' to separate types so llvm will recognize the function name and it won't
     ↪   conflict *)
163  (* let params = List.fold_left (fun s -> (function Formal(t, _) -> s ^ "." ^
     ↪   Utils.string_of_datatype t | _ -> "" )) "" fdecl.formals in *)
164  let name = Utils.string_of_fname fdecl.fname in
165  if name = "main"
166  then "main"
167  else cname ^ "." ^ name(*  ^ params *)
168
169  let get_constructor_name cname fdecl =
170  let params = List.fold_left (fun s -> (function Formal(t, _) -> s ^ "." ^
     ↪   Utils.string_of_datatype t | _ -> "" )) "" fdecl.formals in
171  let name = Utils.string_of_fname fdecl.fname in
172  cname ^ "." ^ name ^ params
173
174  let get_name_without_class fdecl =
175  (* We use '.' to separate types so llvm will recognize the function name and it won't
     ↪   conflict *)
176  let params = List.fold_left (fun s -> (function Formal(t, _) -> s ^ "." ^
     ↪   Utils.string_of_datatype t | _ -> "" )) "" fdecl.formals in
177  let name = Utils.string_of_fname fdecl.fname in
178  let ret_type = Utils.string_of_datatype fdecl.returnType in
179  ret_type ^ "." ^ name ^ "." ^ params
180
181  (* Generate list of all classes to be used for semantic checking *)
182  let build_class_maps reserved cdecls =
183  let reserved_map = List.fold_left (fun m f -> StringMap.add (Utils.string_of_fname
     ↪   f.sfname) f m) StringMap.empty reserved in
184  let helper m (cdecl:Ast.class_decl) =
185  let fieldfun = (fun m -> (function Field(s, d, n) -> if (StringMap.mem (n) m) then
     ↪   raise(Exceptions.DuplicateField) else (StringMap.add n (Field(s, d, n)) m))) in
186  let funcname = get_name cdecl.cname in
```

```ocaml
187    let funcfun m fdecl =
188    if (StringMap.mem (funcname fdecl) m)
189    then raise(Exceptions.DuplicateFunction(funcname fdecl))
190    else if (StringMap.mem (Utils.string_of_fname fdecl.fname) reserved_map)
191    then raise(Exceptions.CannotUseReservedFuncName(Utils.string_of_fname fdecl.fname))
192    else (StringMap.add (funcname fdecl) fdecl m)
193    in
194    let constructor_name = get_constructor_name cdecl.cname in
195    let constructorfun m fdecl =
196    if fdecl.formals = [] then m
197    else if StringMap.mem (constructor_name fdecl) m
198    then raise(Exceptions.DuplicateConstructor)
199    else (StringMap.add (constructor_name fdecl) fdecl m)
200    in
201    let default_c = default_c cdecl.cname in
202    let constructor_map = StringMap.add (get_constructor_name cdecl.cname default_c)
       ↪  default_c StringMap.empty in
203    (if (StringMap.mem cdecl.cname m) then raise (Exceptions.DuplicateClassName(cdecl.cname))
       ↪  else
204    StringMap.add cdecl.cname
205    {        field_map = List.fold_left fieldfun StringMap.empty cdecl.cbody.fields;
206             func_map = List.fold_left funcfun StringMap.empty cdecl.cbody.methods;
207             constructor_map = List.fold_left constructorfun constructor_map
                 ↪  cdecl.cbody.constructors;
208             reserved_map = reserved_map;
209             cdecl = cdecl }
210    m) in
211    List.fold_left helper StringMap.empty cdecls
212
213    let rec get_all_descendants cname accum =
214    if Hashtbl.mem predecessors cname then
215    let direct_descendants = Hashtbl.find predecessors cname in
216    let add_childs_descendants desc_set direct_descendant =
217    get_all_descendants direct_descendant (StringSet.add direct_descendant desc_set)
218    in
219    List.fold_left add_childs_descendants accum direct_descendants
220    else accum
221
222    let inherited potential_predec potential_child =
223    match potential_predec, potential_child with
224    Datatype(Objecttype(predec_cname)), Datatype(Objecttype(child_cname)) ->
225    let descendants = get_all_descendants predec_cname StringSet.empty in
226    if (predec_cname = child_cname) || (StringSet.mem child_cname descendants) then true
227    else raise (Exceptions.LocalAssignTypeMismatch(predec_cname, child_cname))
228    | _ , _ -> false
229
230    let get_equality_binop_type type1 type2 se1 se2 op =
231    (* Equality op not supported for float operands. The correct way to test floats
232    for equality is to check the difference between the operands in question *)
```

```
233    if (type1 = Datatype(Float_t) || type2 = Datatype(Float_t)) then raise
       ↪  (Exceptions.InvalidBinopExpression "Equality operation is not supported for Float
       ↪  types")
234    else
235    match type1, type2 with
236    Datatype(Char_t), Datatype(Int_t)
237    |       Datatype(Int_t), Datatype(Char_t)
238    |       Datatype(Objecttype(_)), Datatype(Null_t)
239    |       Datatype(Null_t), Datatype(Objecttype(_))
240    |       Datatype(Null_t), Arraytype(_, _)
241    |       Arraytype(_, _), Datatype(Null_t) -> SBinop(se1, op, se2, Datatype(Bool_t))
242    | _ ->
243    if type1 = type2 then SBinop(se1, op, se2, Datatype(Bool_t))
244    else raise (Exceptions.InvalidBinopExpression "Equality operator can't operate on
       ↪  different types, with the exception of Int_t and Char_t")
245
246    let get_logical_binop_type se1 se2 op = function
247    (Datatype(Bool_t), Datatype(Bool_t)) -> SBinop(se1, op, se2, Datatype(Bool_t))
248    | _ -> raise (Exceptions.InvalidBinopExpression "Logical operators only operate on Bool_t
       ↪  types")
249
250    let get_comparison_binop_type type1 type2 se1 se2 op =
251    let numerics = SS.of_list [Datatype(Int_t); Datatype(Char_t); Datatype(Float_t)]
252    in
253    if SS.mem type1 numerics && SS.mem type2 numerics
254    then SBinop(se1, op, se2, Datatype(Bool_t))
255    else raise (Exceptions.InvalidBinopExpression "Comparison operators operate on numeric
       ↪  types only")
256
257
258    let get_arithmetic_binop_type se1 se2 op = function
259    (Datatype(Int_t), Datatype(Float_t))
260    |       (Datatype(Float_t), Datatype(Int_t))
261    |       (Datatype(Float_t), Datatype(Float_t))           -> SBinop(se1, op, se2,
       ↪  Datatype(Float_t))
262
263    |       (Datatype(Int_t), Datatype(Char_t))
264    |       (Datatype(Char_t), Datatype(Int_t))
265    |       (Datatype(Char_t), Datatype(Char_t))           -> SBinop(se1, op, se2,
       ↪  Datatype(Char_t))
266
267    |       (Datatype(Int_t), Datatype(Int_t))                   -> SBinop(se1, op, se2,
       ↪  Datatype(Int_t))
268
269    | _ -> raise (Exceptions.InvalidBinopExpression "Arithmetic operators don't support these
       ↪  types")
270
271    let rec get_ID_type env s =
272    try StringMap.find s env.env_locals
```

```
273   with | Not_found ->
274   try let formal = StringMap.find s env.env_parameters in
275   (function Formal(t, _) -> t | Many t -> t ) formal
276   with | Not_found -> raise (Exceptions.UndefinedID s)
277
278   and check_array_primitive env el =
279   let rec iter t sel = function
280   [] -> sel, t
281   |         e :: el ->
282   let se, _ = expr_to_sexpr env e in
283   let se_t = get_type_from_sexpr se in
284   if t = se_t
285   then iter t (se :: sel) el
286   else
287   let t1 = Utils.string_of_datatype t in
288   let t2 = Utils.string_of_datatype se_t in
289   raise(Exceptions.InvalidArrayPrimitiveConsecutiveTypes(t1, t2))
290   in
291   let se, _ = expr_to_sexpr env (List.hd el) in
292   let el = List.tl el in
293   let se_t = get_type_from_sexpr se in
294   let sel, t = iter se_t ([se]) el in
295   let se_t = match t with
296   Datatype(x) -> Arraytype(x, 1)
297   |         Arraytype(x, n) -> Arraytype(x, n+1)
298   |         _ as t -> raise(Exceptions.InvalidArrayPrimitiveType(Utils.string_of_datatype
      ↪  t))
299   in
300   SArrayPrimitive(sel, se_t)
301
302   and check_array_init env d el =
303   (* Get dimension size for the array being created *)
304   let array_complexity = List.length el in
305   let check_elem_type e =
306   let sexpr, _ = expr_to_sexpr env e in
307   let sexpr_type = get_type_from_sexpr sexpr in
308   if sexpr_type = Datatype(Int_t)
309   then sexpr
310   else raise(Exceptions.MustPassIntegerTypeToArrayCreate)
311   in
312   let convert_d_to_arraytype = function
313   Datatype(x) -> Arraytype(x, array_complexity)
314   |         _ as t ->
315   let error_msg = Utils.string_of_datatype t in
316   raise (Exceptions.ArrayInitTypeInvalid(error_msg))
317   in
318   let sexpr_type = convert_d_to_arraytype d in
319   let sel = List.map check_elem_type el in
320   SArrayCreate(d, sel, sexpr_type)
```

```
321
322  and check_array_access env e el =
323  (* Get dimensions of array, ex: foo[10][4][2] is dimen=3 *)
324  let array_dimensions = List.length el in
325  (* Check every e in el is of type Datatype(Int_t). Ensure all indices are ints *)
326  let check_elem_type arg =
327  let sexpr, _ = expr_to_sexpr env arg in
328  let sexpr_type = get_type_from_sexpr sexpr in
329  if sexpr_type = Datatype(Int_t)
330  then sexpr
331  else raise(Exceptions.MustPassIntegerTypeToArrayAccess)
332  in
333  (* converting e to se also checks if the array id has been declared  *)
334  let se, _ = expr_to_sexpr env e in
335  let se_type = get_type_from_sexpr se in
336
337  (* Check that e has enough dimens as e's in el. Return overall datatype of access*)
338  let check_array_dim_vs_params num_params = function
339  Arraytype(t, n) ->
340  if num_params < n then
341  Arraytype(t, (n-num_params))
342  else if num_params = n then
343  Datatype(t)
344  else
345  raise (Exceptions.ArrayAccessInvalidParamLength(string_of_int num_params, string_of_int
       ↪  n))
346  |         _ as t ->
347  let error_msg = Utils.string_of_datatype t in
348  raise (Exceptions.ArrayAccessExpressionNotArray(error_msg))
349  in
350  let sexpr_type = check_array_dim_vs_params array_dimensions se_type in
351  let sel = List.map check_elem_type el in
352
353  SArrayAccess(se, sel, sexpr_type)
354
355  and check_obj_access env lhs rhs =
356  let check_lhs = function
357  This                        -> SId("this", Datatype(Objecttype(env.env_name)))
358  |       Id s                        -> SId(s, get_ID_type env s)
359  |         ArrayAccess(e, el)        -> check_array_access env e el
360  |        _ as e          -> raise (Exceptions.LHSofRootAccessMustBeIDorFunc
       ↪  (Utils.string_of_expr e))
361  in
362  let ptype_name parent_type = match parent_type with
363  Datatype(Objecttype(name))         -> name
364  |        _ as d                                      -> raise
       ↪  (Exceptions.ObjAccessMustHaveObjectType (Utils.string_of_datatype d))
365  in
366  let rec check_rhs (env) parent_type (top_level_env) =
```

```
367  let pt_name = ptype_name parent_type in
368  let get_id_type_from_object env (id) cname tlenv =
369  let cmap = StringMap.find cname env.env_class_maps in
370  let match_field f = match f with
371  Field(scope, d, n) ->
372  (* Have to update this with all parent classes checks *)
373  if scope = Ast.Private && tlenv.env_name <> env.env_name then
374  raise(Exceptions.CannotAccessPrivateFieldInNonProperScope(n, env.env_name,
     ↪  tlenv.env_name))
375  else d
376  in
377  try match_field (StringMap.find id cmap.field_map)
378  with | Not_found -> raise (Exceptions.UnknownIdentifierForClass(id, cname))
379  in
380  function
381  (* Check fields in parent *)
382  Id s                                -> SId(s, (get_id_type_from_object env s pt_name
     ↪  top_level_env)), env
383  (* Check functions in parent *)
384  |        Call(fname, el)        ->
385  let env = update_env_name env pt_name in
386  check_call_type top_level_env true env fname el, env
387  (* Set parent, check if base is field *)
388  |        ObjAccess(e1, e2)        ->
389  let old_env = env in
390  let lhs, env = check_rhs env parent_type top_level_env e1 in
391  let lhs_type = get_type_from_sexpr lhs in
392
393  let pt_name = ptype_name lhs_type in
394  let lhs_env = update_env_name env pt_name in
395
396  let rhs, env = check_rhs lhs_env lhs_type top_level_env e2 in
397  let rhs_type = get_type_from_sexpr rhs in
398  SObjAccess(lhs, rhs, rhs_type), old_env
399  |        _ as e                        -> raise (Exceptions.InvalidAccessLHS
     ↪  (Utils.string_of_expr e))
400  in
401  let arr_lhs, _ = expr_to_sexpr env lhs in
402  let arr_lhs_type = get_type_from_sexpr arr_lhs in
403  match arr_lhs_type with
404  Arraytype(Char_t, 1) -> raise(Exceptions.CannotAccessLengthOfCharArray)
405  |        Arraytype(_, _) ->
406  let rhs = match rhs with
407  Id("length") -> SId("length", Datatype(Int_t))
408  |        _ -> raise(Exceptions.CanOnlyAccessLengthOfArray)
409  in
410  SObjAccess(arr_lhs, rhs, Datatype(Int_t))
411  | _ ->
412  let lhs = check_lhs lhs in
```

```ocaml
413    let lhs_type = get_type_from_sexpr lhs in
414
415    let ptype_name = ptype_name lhs_type in
416    let lhs_env = update_env_name env ptype_name in
417
418    let rhs, _ = check_rhs lhs_env lhs_type env rhs in
419    let rhs_type = get_type_from_sexpr rhs in
420    SObjAccess(lhs, rhs, rhs_type)
421
422    and check_call_type top_level_env isObjAccess env fname el =
423    let sel, env = exprl_to_sexprl env el in
424    (* check that 'env.env_name' is in the list of defined classes *)
425    let cmap =
426    try StringMap.find env.env_name env.env_class_maps
427    with | Not_found -> raise (Exceptions.UndefinedClass env.env_name)
428    in
429
430    let handle_param formal param =
431    let fty = match formal with Formal(d, _) -> d | _ -> Datatype(Void_t) in
432    let pty = get_type_from_sexpr param in
433    match fty, pty with
434    Datatype(Objecttype(f)), Datatype(Objecttype(p)) ->
435    if f <> p then
436    try let descendants = Hashtbl.find predecessors f in
437    let _ = try List.find (fun d -> p = d) descendants
438    with | Not_found -> raise(Exceptions.CannotPassNonInheritedClassesInPlaceOfOthers(f, p))
439    in
440    let rt = Datatype(Objecttype(f)) in
441    SCall("cast", [param; SId("ignore", rt)], rt, 0)
442    with | Not_found -> raise(Exceptions.ClassIsNotExtendedBy(f, p))
443    else param
444    |        _ -> if fty = pty then param else
       raise(Exceptions.IncorrectTypePassedToFunction(fname, Utils.string_of_datatype pty))
445    in
446
447    let index fdecl fname =
448    let cdecl = cmap.cdecl in
449    (* Have to update this with all parent classes checks *)
450    let _ =
451    if fdecl.scope = Ast.Private && top_level_env.env_name <> env.env_name then
452    raise(Exceptions.CannotAccessPrivateFunctionInNonProperScope(get_name env.env_name fdecl,
       env.env_name, top_level_env.env_name))
453    in
454    (* Not exactly sure why there needs to be a list.rev *)
455    let fns = List.rev cdecl.cbody.methods in
456    let rec find x lst =
457    match lst with
458    | [] -> raise (Failure ("Could not find " ^ fname))
459    | fdecl :: t ->
```

```
460   let search_name = (get_name env.env_name fdecl) in
461   if x = search_name then 0
462   else if search_name = "main" then find x t
463   else 1 + find x t
464   in
465   find fname fns
466   in
467
468   let handle_params (formals) params =
469   match formals, params with
470   [Many(Any)], _ -> params
471   |         [], [] -> []
472   |         [], _
473   |         _, [] -> raise(Exceptions.IncorrectTypePassedToFunction(fname,
      ↪   Utils.string_of_datatype (Datatype(Void_t))))
474   |            _ ->
475   let len1 = List.length formals in
476   let len2 = List.length params in
477   if len1 <> len2 then raise(Exceptions.IncorrectNumberOfArguments(fname, len1, len2))
478   else
479   List.map2 handle_param formals sel
480   in
481
482   let sfname = env.env_name ^ "." ^ fname in
483   try let func = StringMap.find fname cmap.reserved_map in
484   let actuals = handle_params func.sformals sel in
485   SCall(fname, actuals, func.sreturnType, 0)
486   with | Not_found ->
487   try let f = StringMap.find sfname cmap.func_map in
488   let actuals = handle_params f.formals sel in
489   let index = index f sfname in
490   SCall(sfname, actuals, f.returnType, index)
491   with | Not_found -> raise(Exceptions.FunctionNotFound(env.env_name, sfname)) | _ as ex ->
      ↪   raise ex
492
493   and check_object_constructor env s el =
494   let sel, env = exprl_to_sexprl env el in
495   (* check that 'env.env_name' is in the list of defined classes *)
496   let cmap =
497   try StringMap.find s env.env_class_maps
498   with | Not_found -> raise (Exceptions.UndefinedClass s)
499   in
500   (* get a list of the types of the actuals to match against defined function formals *)
501   let params = List.fold_left (fun s e -> s ^ "." ^ (Utils.string_of_datatype
      ↪   (get_type_from_sexpr e))) "" sel in
502   let constructor_name = s ^ "." ^ "constructor" ^ params in
503   let _ =
504   try StringMap.find constructor_name cmap.constructor_map
505   with | Not_found -> raise (Exceptions.ConstructorNotFound constructor_name)
```

```
506    in
507    let ftype = Datatype(Objecttype(s)) in
508    (* Add a reference to the class in front of the function call *)
509    (* Must properly handle the case where this is a reserved function *)
510    SObjectCreate(constructor_name, sel, ftype)
511
512    and check_assign env e1 e2 =
513    let se1, env = expr_to_sexpr env e1 in
514    let se2, env = expr_to_sexpr env e2 in
515    let type1 = get_type_from_sexpr se1 in
516    let type2 = get_type_from_sexpr se2 in
517    match (type1, se2) with
518    Datatype(Objecttype(_)), SNull
519    |        Arraytype(_, _), SNull -> SAssign(se1, se2, type1)
520    |    _ ->
521    match type1, type2 with
522    Datatype(Char_t), Datatype(Int_t)
523    |        Datatype(Int_t), Datatype(Char_t) -> SAssign(se1, se2, type1)
524    |        Datatype(Objecttype(d)), Datatype(Objecttype(t)) ->
525    if d = t then SAssign(se1, se2, type1)
526    else if inherited type1 type2 then
527    SAssign(se1, SCall("cast", [se2; SId("ignore", type1)], type1, 0), type1)
528    else raise (Exceptions.AssignmentTypeMismatch(Utils.string_of_datatype type1,
       ↪  Utils.string_of_datatype type2))
529    | _ ->
530    if type1 = type2
531    then SAssign(se1, se2, type1)
532    else raise (Exceptions.AssignmentTypeMismatch(Utils.string_of_datatype type1,
       ↪  Utils.string_of_datatype type2))
533
534    and check_unop env op e =
535    let check_num_unop t = function
536    Sub        -> t
537    |        _                  -> raise(Exceptions.InvalidUnaryOperation)
538    in
539    let check_bool_unop = function
540    Not        -> Datatype(Bool_t)
541    |        _                  -> raise(Exceptions.InvalidUnaryOperation)
542    in
543    let se, env = expr_to_sexpr env e in
544    let t = get_type_from_sexpr se in
545    match t with
546    Datatype(Int_t)
547    |      Datatype(Float_t)          -> SUnop(op, se, check_num_unop t op)
548    |       Datatype(Bool_t)          -> SUnop(op, se, check_bool_unop op)
549    |       _ -> raise(Exceptions.InvalidUnaryOperation)
550
551    and check_binop env e1 op e2 =
552    let se1, env = expr_to_sexpr env e1 in
```

```
553    let se2, env = expr_to_sexpr env e2 in
554    let type1 = get_type_from_sexpr se1 in
555    let type2 = get_type_from_sexpr se2 in
556    match op with
557    Equal | Neq -> get_equality_binop_type type1 type2 se1 se2 op
558    | And | Or -> get_logical_binop_type se1 se2 op (type1, type2)
559    | Less | Leq | Greater | Geq -> get_comparison_binop_type type1 type2 se1 se2 op
560    | Add | Mult | Sub | Div | Mod -> get_arithmetic_binop_type se1 se2 op (type1, type2)
561    | _ -> raise (Exceptions.InvalidBinopExpression ((Utils.string_of_op op) ^ " is not a
    ↪   supported binary op"))
562
563    and check_delete env e =
564    let se, _ = expr_to_sexpr env e in
565    let t = get_type_from_sexpr se in
566    match t with
567    Arraytype(_, _) | Datatype(Objecttype(_)) -> SDelete(se)
568    |         _ -> raise(Exceptions.CanOnlyDeleteObjectsOrArrays)
569
570    and expr_to_sexpr env = function
571    Int_Lit i           -> SInt_Lit(i), env
572    |    Boolean_Lit b      -> SBoolean_Lit(b), env
573    |    Float_Lit f        -> SFloat_Lit(f), env
574    |    String_Lit s       -> SString_Lit(s), env
575    |    Char_Lit c         -> SChar_Lit(c), env
576    |    This               -> SId("this", Datatype(Objecttype(env.env_name))), env
577    |    Id s               -> SId(s, get_ID_type env s), env
578    |    Null               -> SNull, env
579    |    Noexpr             -> SNoexpr, env
580
581    |    ObjAccess(e1, e2)   -> check_obj_access env e1 e2, env
582    |    ObjectCreate(s, el) -> check_object_constructor env s el, env
583    |    Call(s, el)         -> check_call_type env false env s el, env
584
585    |    ArrayCreate(d, el)  -> check_array_init env d el, env
586    |    ArrayAccess(e, el)  -> check_array_access env e el, env
587    |    ArrayPrimitive el   -> check_array_primitive env el, env
588
589    |    Assign(e1, e2)      -> check_assign env e1 e2, env
590    |    Unop(op, e)         -> check_unop env op e, env
591    |    Binop(e1, op, e2)   -> check_binop env e1 op e2, env
592    |        Delete(e)                       -> check_delete env e, env
593
594
595    and get_type_from_sexpr = function
596    SInt_Lit(_)                               -> Datatype(Int_t)
597    |        SBoolean_Lit(_)                     -> Datatype(Bool_t)
598    |        SFloat_Lit(_)                      -> Datatype(Float_t)
599    |        SString_Lit(_)                      -> Arraytype(Char_t, 1)
600    |        SChar_Lit(_)                        -> Datatype(Char_t)
```

```
601  |           SId(_, d)                                        -> d
602  |           SBinop(_, _, _, d)                    -> d
603  |           SAssign(_, _, d)                  -> d
604  |           SNoexpr                                      -> Datatype(Void_t)
605  |           SArrayCreate(_, _, d)           -> d
606  |           SArrayAccess(_, _, d)             -> d
607  |           SObjAccess(_, _, d)                  -> d
608  |           SCall(_, _, d, _)                  -> d
609  |     SObjectCreate(_, _, d)           -> d
610  |           SArrayPrimitive(_, d)           -> d
611  |            SUnop(_, _, d)                        -> d
612  |           SNull                                         -> Datatype(Null_t)
613  |           SDelete _                               -> Datatype(Void_t)
614
615  and exprl_to_sexprl env el =
616  let env_ref = ref(env) in
617  let rec helper = function
618  head::tail ->
619  let a_head, env = expr_to_sexpr !env_ref head in
620  env_ref := env;
621  a_head::(helper tail)
622  | [] -> []
623  in (helper el), !env_ref
624
625  let rec local_handler d s e env =
626  if StringMap.mem s env.env_locals
627  then raise (Exceptions.DuplicateLocal s)
628  else
629  let se, env = expr_to_sexpr env e in
630  let t = get_type_from_sexpr se in
631  if t = Datatype(Void_t) || t = Datatype(Null_t) || t = d || (inherited d t)
632  then
633  let new_env = {
634          env_class_maps = env.env_class_maps;
635          env_name = env.env_name;
636          env_cmap = env.env_cmap;
637          env_locals = StringMap.add s d env.env_locals;
638          env_parameters = env.env_parameters;
639          env_returnType = env.env_returnType;
640          env_in_for = env.env_in_for;
641          env_in_while = env.env_in_while;
642          env_reserved = env.env_reserved;
643  } in
644  (* if the user-defined type being declared is not in global classes map, it is an
     ↪  undefined class *)
645  (match d with
646  Datatype(Objecttype(x)) ->
647  (if not (StringMap.mem (Utils.string_of_object d) env.env_class_maps)
648  then raise (Exceptions.UndefinedClass (Utils.string_of_object d))
```

```
649    else
650    let local = if inherited d t then SLocal(t, s, se) else SLocal(d, s, se)
651    in local, new_env)
652    | _ -> SLocal(d, s, se), new_env)
653    else
654    (let type1 = (Utils.string_of_datatype t) in
655    let type2 = (Utils.string_of_datatype d) in
656    let ex = Exceptions.LocalAssignTypeMismatch(type1, type2) in
657    raise ex)
658
659    let rec check_sblock sl env = match sl with
660    [] -> SBlock([SExpr(SNoexpr, Datatype(Void_t))]), env
661    |         _ ->
662    let sl, _ = convert_stmt_list_to_sstmt_list env sl in
663    SBlock(sl), env
664
665    and check_expr_stmt e env =
666    let se, env = expr_to_sexpr env e in
667    let t = get_type_from_sexpr se in
668    SExpr(se, t), env
669
670    and check_return e env =
671    let se, _ = expr_to_sexpr env e in
672    let t = get_type_from_sexpr se in
673    match t, env.env_returnType with
674    Datatype(Null_t), Datatype(Objecttype(_))
675    |         Datatype(Null_t), Arraytype(_, _) -> SReturn(se, t), env
676    |         _ ->
677    if t = env.env_returnType
678    then SReturn(se, t), env
679    else raise (Exceptions.ReturnTypeMismatch(Utils.string_of_datatype t,
       ↪  Utils.string_of_datatype env.env_returnType))
680
681    and check_if e s1 s2 env =
682    let se, _ = expr_to_sexpr env e in
683    let t = get_type_from_sexpr se in
684    let ifbody, _ = parse_stmt env s1 in
685    let elsebody, _ = parse_stmt env s2 in
686    if t = Datatype(Bool_t)
687    then SIf(se, ifbody, elsebody), env
688    else raise Exceptions.InvalidIfStatementType
689
690    and check_for e1 e2 e3 s env =
691    let old_val = env.env_in_for in
692    let env = update_call_stack env true env.env_in_while in
693
694    let se1, _ = expr_to_sexpr env e1 in
695    let se2, _ = expr_to_sexpr env e2 in
696    let se3, _ = expr_to_sexpr env e3 in
```

```
697    let forbody, _ = parse_stmt env s in
698    let conditional = get_type_from_sexpr se2 in
699    let sfor =
700    if (conditional = Datatype(Bool_t) || conditional = Datatype(Void_t))
701    then SFor(se1, se2, se3, forbody)
702    else raise Exceptions.InvalidForStatementType
703    in
704
705    let env = update_call_stack env old_val env.env_in_while in
706    sfor, env
707
708    and check_while e s env =
709    let old_val = env.env_in_while in
710    let env = update_call_stack env env.env_in_for true in
711
712    let se, _ = expr_to_sexpr env e in
713    let t = get_type_from_sexpr se in
714    let sstmt, _ = parse_stmt env s in
715    let swhile =
716    if (t = Datatype(Bool_t) || t = Datatype(Void_t))
717    then SWhile(se, sstmt)
718    else raise Exceptions.InvalidWhileStatementType
719    in
720
721    let env = update_call_stack env env.env_in_for old_val in
722    swhile, env
723
724    and check_break env =
725    if env.env_in_for || env.env_in_while then
726    SBreak, env
727    else
728    raise Exceptions.CannotCallBreakOutsideOfLoop
729
730    and check_continue env =
731    if env.env_in_for || env.env_in_while then
732    SContinue, env
733    else
734    raise Exceptions.CannotCallContinueOutsideOfLoop
735
736    and parse_stmt env = function
737    Block sl                              -> check_sblock sl env
738    |        Expr e                              -> check_expr_stmt e env
739    |        Return e                        -> check_return e env
740    |        If(e, s1, s2)              -> check_if e s1 s2        env
741    |        For(e1, e2, e3, e4)      -> check_for e1 e2 e3 e4 env
742    |        While(e, s)                      -> check_while e s env
743    |         Break                            -> check_break env (* Need to
       ↪  check if in right context *)
```

```
744  |    Continue                                        -> check_continue env (* Need to check if in
     ↪  right context *)
745  |    Local(d, s, e)                                  -> local_handler d s e env
746
747  (* Update this function to return an env object *)
748  and convert_stmt_list_to_sstmt_list env stmt_list =
749  let env_ref = ref(env) in
750  let rec iter = function
751  head::tail ->
752  let a_head, env = parse_stmt !env_ref head in
753  env_ref := env;
754  a_head::(iter tail)
755  | [] -> []
756  in
757  let sstmt_list = (iter stmt_list), !env_ref in
758  sstmt_list
759
760  let append_code_to_main fbody cname ret_type =
761  let key = Hashtbl.find struct_indexes cname in
762  let init_this = [SLocal(
763  ret_type,
764  "this",
765  SCall(        "cast",
766  [SCall("malloc",
767  [
768  SCall("sizeof", [SId("ignore", ret_type)], Datatype(Int_t), 0)
769  ],
770  Arraytype(Char_t, 1), 0)
771  ],
772  ret_type, 0
773  )
774  );
775  SExpr(
776  SAssign(
777  SObjAccess(
778  SId("this", ret_type),
779  SId(".key", Datatype(Int_t)),
780  Datatype(Int_t)
781  ),
782  SInt_Lit(key),
783  Datatype(Int_t)
784  ),
785  Datatype(Int_t)
786  )
787  ]
788  in
789  init_this @ fbody
790
791  let convert_constructor_to_sfdecl class_maps reserved class_map cname constructor =
```

```
792  let env = {
793          env_class_maps          = class_maps;
794          env_name                = cname;
795          env_cmap                 = class_map;
796          env_locals               = StringMap.empty;
797          env_parameters           = List.fold_left (fun m f -> match f with Formal(d, s) ->
        ↪  (StringMap.add s f m) | _ -> m) StringMap.empty constructor.formals;
798          env_returnType           = Datatype(Objecttype(cname));
799          env_in_for                  = false;
800          env_in_while             = false;
801          env_reserved             = reserved;
802  } in
803  let fbody = fst (convert_stmt_list_to_sstmt_list env constructor.body) in
804  {
805          sfname                   = Ast.FName (get_constructor_name cname constructor);
806          sreturnType = Datatype(Objecttype(cname));
807          sformals          = constructor.formals;
808          sbody                    = append_code_to_constructor fbody cname
        ↪  (Datatype(Objecttype(cname)));
809          func_type         = Sast.User;
810          overrides          = false;
811          source                    = "NA";
812  }
813
814  let check_fbody fname fbody returnType =
815  let len = List.length fbody in
816  if len = 0 then () else
817  let final_stmt = List.hd (List.rev fbody) in
818  match returnType, final_stmt with
819  Datatype(Void_t), _ -> ()
820  |         _, SReturn(_, _) -> ()
821  |         _ -> raise(Exceptions.AllNonVoidFunctionsMustEndWithReturn(fname))
822
823  let convert_fdecl_to_sfdecl class_maps reserved class_map cname fdecl =
824  let root_cname = match fdecl.root_cname with
825  Some(x) -> x
826  | None -> cname
827  in
828  let class_formal =
829  if fdecl.overrides then
830  Ast.Formal(Datatype(Objecttype(root_cname)), "this")
831  else
832  Ast.Formal(Datatype(Objecttype(cname)), "this")
833  in
834  let env_param_helper m fname = match fname with
835  Formal(d, s) -> (StringMap.add s fname m)
836  |         _ -> m
837  in
```

```ocaml
838  let env_params = List.fold_left env_param_helper StringMap.empty (class_formal ::
     ↪  fdecl.formals) in
839  let env = {
840          env_class_maps         = class_maps;
841          env_name              = cname;
842          env_cmap               = class_map;
843          env_locals            = StringMap.empty;
844          env_parameters        = env_params;
845          env_returnType        = fdecl.returnType;
846          env_in_for               = false;
847          env_in_while          = false;
848          env_reserved          = reserved;
849  }
850  in
851  let fbody = fst (convert_stmt_list_to_sstmt_list env fdecl.body) in
852  let fname = (get_name cname fdecl) in
853  ignore(check_fbody fname fbody fdecl.returnType);
854  let fbody = if fname = "main"
855  then (append_code_to_main fbody cname (Datatype(Objecttype(cname))))
856  else fbody
857  in
858  (* We add the class as the first parameter to the function for codegen *)
859  {
860          sfname                          = Ast.FName (get_name cname fdecl);
861          sreturnType         = fdecl.returnType;
862          sformals                = class_formal :: fdecl.formals;
863          sbody                      = fbody;
864          func_type             = Sast.User;
865          overrides        = fdecl.overrides;
866          source                    = cname;
867  }
868
869  let convert_cdecl_to_sast sfdecls (cdecl:Ast.class_decl) =
870  {
871          scname = cdecl.cname;
872          sfields = cdecl.cbody.fields;
873          sfuncs = sfdecls;
874  }
875
876  (*
877  * Given a list of func_decls for the base class and a single func_decl
878  * for the child class, replaces func_decls for the base class if any of them
879  * have the same method signature
880  *)
881  let replace_fdecl_in_base_methods base_cname base_methods child_fdecl =
882  let replace base_fdecl accum =
883  let get_root_cname = function
884  None -> Some(base_cname)
885  | Some(x) -> Some(x)
```

```
886   in
887   let modify_child_fdecl =
888   {
889           scope = child_fdecl.scope;
890           fname = child_fdecl.fname;
891           returnType = child_fdecl.returnType;
892           formals = child_fdecl.formals;
893           body = child_fdecl.body;
894           overrides = true;
895           root_cname = get_root_cname base_fdecl.root_cname;
896   }
897   in
898   if (get_name_without_class base_fdecl) = (get_name_without_class child_fdecl)
899   then modify_child_fdecl::accum
900   else base_fdecl::accum
901   in
902   List.fold_right replace base_methods []
903
904   let merge_methods base_cname base_methods child_methods =
905   let check_overrides child_fdecl accum =
906   let base_checked_for_overrides =
907   replace_fdecl_in_base_methods base_cname (fst accum) child_fdecl
908   in
909   if (fst accum) = base_checked_for_overrides
910   then ((fst accum), child_fdecl::(snd accum))
911   else (base_checked_for_overrides, (snd accum))
912   in
913   let updated_base_and_child_fdecls =
914   List.fold_right check_overrides child_methods (base_methods, [])
915   in
916   (fst updated_base_and_child_fdecls) @ (snd updated_base_and_child_fdecls)
917
918   let merge_cdecls base_cdecl child_cdecl =
919   (* return a cdecl in which cdecl.cbody.fields contains the fields of
920   the extended class, concatenated by the fields of the child class *)
921   let child_cbody =
922   {
923           fields = base_cdecl.cbody.fields @ child_cdecl.cbody.fields;
924           constructors = child_cdecl.cbody.constructors;
925           methods = merge_methods base_cdecl.cname base_cdecl.cbody.methods
926             ↪   child_cdecl.cbody.methods
926   }
927   in
928   {
929           cname = child_cdecl.cname;
930           extends = child_cdecl.extends;
931           cbody = child_cbody
932   }
933
```

```ocaml
934   (* returns a list of cdecls that contains inherited fields *)
935   let inherit_fields_cdecls cdecls inheritance_forest =
936   (* iterate through cdecls to make a map for lookup *)
937   let cdecl_lookup = List.fold_left (fun a litem -> StringMap.add litem.cname litem a)
      ↪  StringMap.empty cdecls in
938   let add_key key pred maps =
939   let elem1 = StringSet.add key (fst maps) in
940   let accum acc child = StringSet.add child acc in
941   let elem2 = List.fold_left (accum) (snd maps) pred in
942   (elem1, elem2)
943   in
944   let empty_s = StringSet.empty in
945   let res = StringMap.fold add_key inheritance_forest (empty_s, empty_s) in
946   let roots = StringSet.diff (fst res) (snd res) in
947   let rec add_inherited_fields predec desc map_to_update =
948   let merge_fields accum descendant =
949   let updated_predec_cdecl = StringMap.find predec accum in
950   let descendant_cdecl_to_update = StringMap.find descendant cdecl_lookup in
951   let merged = merge_cdecls updated_predec_cdecl descendant_cdecl_to_update in
952   let updated = (StringMap.add descendant merged accum) in
953   if (StringMap.mem descendant inheritance_forest) then
954   let descendants_of_descendant = StringMap.find descendant inheritance_forest in
955   add_inherited_fields descendant descendants_of_descendant updated
956   else updated
957   in
958   List.fold_left merge_fields map_to_update desc
959   in
960   (* map class name of every class_decl in 'cdecls' to its inherited cdecl *)
961   let inherited_cdecls =
962   let traverse_tree tree_root accum =
963   let tree_root_descendant = StringMap.find tree_root inheritance_forest in
964   let accum_with_tree_root_mapping = StringMap.add tree_root (StringMap.find tree_root
      ↪  cdecl_lookup) accum in
965   add_inherited_fields tree_root tree_root_descendant accum_with_tree_root_mapping
966   in
967   StringSet.fold traverse_tree roots StringMap.empty
968   in
969   (* build a list of updated cdecls corresponding to the sequence of cdecls in 'cdecls' *)
970   let add_inherited_cdecl cdecl accum =
971   let inherited_cdecl =
972   try StringMap.find cdecl.cname inherited_cdecls
973   with | Not_found -> cdecl
974   in
975   inherited_cdecl::accum
976   in
977   let result = List.fold_right add_inherited_cdecl cdecls [] in
978   result
979
980   let convert_cdecls_to_sast class_maps reserved (cdecls:Ast.class_decl list) =
```

```
981   let find_main = (fun f -> match f.sfname with FName n -> n = "main" | _ -> false) in
982   let get_main func_list =
983   let mains = (List.find_all find_main func_list) in
984   if List.length mains < 1 then
985   raise Exceptions.MainNotDefined
986   else if List.length mains > 1 then
987   raise Exceptions.MultipleMainsDefined
988   else List.hd mains
989   in
990   let remove_main func_list =
991   List.filter (fun f -> not (find_main f)) func_list
992   in
993   let find_default_constructor cdecl clist =
994   let default_cname = cdecl.cname ^ "." ^ "constructor" in
995   let find_default_c f =
996   match f.sfname with FName n -> n = default_cname | _ -> false
997   in
998   try let _ = List.find find_default_c clist in
999   clist
1000  with | Not_found ->
1001  let default_c = default_sc cdecl.cname in
1002  default_c :: clist
1003  in
1004  let handle_cdecl cdecl =
1005  let class_map = StringMap.find cdecl.cname class_maps in
1006  let sconstructor_list = List.fold_left (fun l c -> (convert_constructor_to_sfdecl
      ↪  class_maps reserved class_map cdecl.cname c) :: l) [] cdecl.cbody.constructors in
1007  let sconstructor_list = find_default_constructor cdecl sconstructor_list in
1008  let func_list = List.fold_left (fun l f -> (convert_fdecl_to_sfdecl class_maps reserved
      ↪  class_map cdecl.cname f) :: l) [] cdecl.cbody.methods in
1009  let sfunc_list = remove_main func_list in
1010  let scdecl = convert_cdecl_to_sast sfunc_list cdecl in
1011  (scdecl, func_list @ sconstructor_list)
1012  in
1013  let iter_cdecls t c =
1014  let scdecl = handle_cdecl c in
1015  (fst scdecl :: fst t, snd scdecl @ snd t)
1016  in
1017  let scdecl_list, func_list = List.fold_left iter_cdecls ([], []) cdecls in
1018  let main = get_main func_list in
1019  let funcs = remove_main func_list in
1020  (* let funcs = (add_default_constructors cdecls class_maps) @ funcs in *)
1021  {
1022          classes               = scdecl_list;
1023          functions             = funcs;
1024          main                  = main;
1025          reserved              = reserved;
1026  }
1027
```

```
1028  let add_reserved_functions =
1029  let reserved_stub name return_type formals =
1030  {
1031          sfname                          = FName(name);
1032          sreturnType         = return_type;
1033          sformals                = formals;
1034          sbody                       = [];
1035          func_type               = Sast.Reserved;
1036          overrides                = false;
1037          source                      = "NA";
1038  }
1039  in
1040  let i32_t = Datatype(Int_t) in
1041  let void_t = Datatype(Void_t) in
1042  let str_t = Arraytype(Char_t, 1) in
1043  let mf t n = Formal(t, n) in (* Make formal *)
1044  let reserved = [
1045  reserved_stub "print"        (void_t)          ([Many(Any)]);
1046  reserved_stub "malloc"        (str_t)          ([mf i32_t "size"]);
1047  reserved_stub "cast"        (Any)                ([mf Any "in"]);
1048  reserved_stub "sizeof"        (i32_t)          ([mf Any "in"]);
1049  reserved_stub "open"        (i32_t)          ([mf str_t "path"; mf i32_t "flags"]);
1050  reserved_stub "close"        (i32_t)          ([mf i32_t "fd"]);
1051  reserved_stub "read"        (i32_t)          ([mf i32_t "fd"; mf str_t "buf"; mf i32_t
      ↪  "nbyte"]);
1052  reserved_stub "write"        (i32_t)          ([mf i32_t "fd"; mf str_t "buf"; mf i32_t
      ↪  "nbyte"]);
1053  reserved_stub "lseek"        (i32_t)          ([mf i32_t "fd"; mf i32_t "offset"; mf
      ↪  i32_t "whence"]);
1054  reserved_stub "exit"        (void_t)          ([mf i32_t "status"]);
1055  reserved_stub "getchar" (i32_t)          ([]);
1056  reserved_stub "input"        (str_t)          ([]);
1057  ] in
1058  reserved
1059
1060  let build_inheritance_forest cdecls cmap =
1061  let handler a cdecl =
1062  match cdecl.extends with
1063  Parent(s)          ->
1064  let new_list = if (StringMap.mem s a) then
1065  cdecl.cname::(StringMap.find s a)
1066  else
1067  [cdecl.cname]
1068  in
1069  Hashtbl.add predecessors s new_list;
1070  (StringMap.add s new_list a)
1071  |        NoParent          -> a
1072  in
1073  let forest = List.fold_left handler StringMap.empty cdecls in
```

```
1074
1075   let handler key value =
1076   if not (StringMap.mem key cmap) then
1077   raise (Exceptions.UndefinedClass key)
1078   in
1079   ignore(StringMap.iter handler forest);
1080   forest
1081
1082   let merge_maps m1 m2 =
1083   StringMap.fold (fun k v a -> StringMap.add k v a) m1 m2
1084
1085   let update_class_maps map_type cmap_val cname cmap_to_update =
1086   let update m map_type =
1087   if map_type = "field_map" then
1088   {
1089           field_map = cmap_val;
1090           func_map = m.func_map;
1091           constructor_map = m.constructor_map;
1092           reserved_map = m.reserved_map;
1093           cdecl = m.cdecl;
1094   }
1095   else m
1096   in
1097   let updated = StringMap.find cname cmap_to_update in
1098   let updated = update updated map_type in
1099   let updated = StringMap.add cname updated cmap_to_update in
1100   updated
1101
1102   let inherit_fields class_maps predecessors =
1103   (* Get basic inheritance map *)
1104   let add_key key pred map = StringMap.add key pred map in
1105   let cmaps_inherit = StringMap.fold add_key class_maps StringMap.empty in
1106   (* Perform accumulation of child classes *)
1107   let add_key key pred maps =
1108   let elem1 = StringSet.add key (fst maps) in
1109   let accum acc child = StringSet.add child acc in
1110   let elem2 = List.fold_left (accum) (snd maps) pred in
1111   (elem1, elem2)
1112   in
1113   let empty_s = StringSet.empty in
1114   let res = StringMap.fold add_key predecessors (empty_s, empty_s) in
1115   let roots = StringSet.diff (fst res) (snd res) in
1116   (*in let _ = print_set_members roots*)
1117   let rec add_inherited_fields predec desc cmap_to_update =
1118   let cmap_inherit accum descendant =
1119   let predec_field_map = (StringMap.find predec accum).field_map in
1120   let desc_field_map = (StringMap.find descendant accum).field_map in
1121   let merged = merge_maps predec_field_map desc_field_map in
1122   let updated = update_class_maps "field_map" merged descendant accum in
```

```
1123   if (StringMap.mem descendant predecessors) then
1124   let descendants_of_descendant = StringMap.find descendant predecessors in
1125   add_inherited_fields descendant descendants_of_descendant updated
1126   else updated
1127   in
1128   List.fold_left cmap_inherit cmap_to_update desc
1129   (* end of add_inherited_fields *)
1130   in
1131   let result = StringSet.fold (fun x a -> add_inherited_fields x (StringMap.find x
       ↪   predecessors) a) roots cmaps_inherit
1132   (*in let _ = print_map result*)
1133   in result
1134
1135   (* TODO Check that this actually works *)
1136   let check_cyclical_inheritance cdecls predecessors =
1137   let handle_predecessor cdecl parent predecessor =
1138   if cdecl.cname = predecessor then
1139   raise(Exceptions.CyclicalDependencyBetween(cdecl.cname, parent))
1140   in
1141   let handle_cdecl cdecl =
1142   if StringMap.mem cdecl.cname predecessors
1143   then
1144   let pred_list = StringMap.find cdecl.cname predecessors in
1145   List.iter (handle_predecessor cdecl (List.hd pred_list)) pred_list
1146   else ()
1147   in
1148   List.iter handle_cdecl cdecls
1149
1150   let build_func_map_inherited_lookup cdecls_inherited =
1151   let build_func_map cdecl =
1152   let add_func m fdecl = StringMap.add (get_name cdecl.cname fdecl) fdecl m in
1153   List.fold_left add_func StringMap.empty cdecl.cbody.methods
1154   in
1155   let add_class_func_map m cdecl = StringMap.add cdecl.cname (build_func_map cdecl) m in
1156   List.fold_left add_class_func_map StringMap.empty cdecls_inherited
1157
1158   let add_inherited_methods cmaps cdecls func_maps_inherited =
1159   let find_cdecl cname =
1160   try List.find (fun cdecl -> cdecl.cname = cname) cdecls
1161   with | Not_found -> raise Not_found
1162   in
1163   let update_with_inherited_methods cname cmap =
1164   let fmap = StringMap.find cname func_maps_inherited in
1165   let cdecl = find_cdecl cname in
1166   {
1167           field_map = cmap.field_map;
1168           func_map = fmap;
1169           constructor_map = cmap.constructor_map;
1170           reserved_map = cmap.reserved_map;
```

```
1171          cdecl = cdecl;
1172  }
1173  in
1174  let add_updated_cmap cname cmap accum = StringMap.add cname
      ↪  (update_with_inherited_methods cname cmap) accum in
1175  StringMap.fold add_updated_cmap cmaps StringMap.empty
1176
1177  let handle_inheritance cdecls class_maps =
1178  let predecessors = build_inheritance_forest cdecls class_maps in
1179  let cdecls_inherited = inherit_fields_cdecls cdecls predecessors in
1180  let func_maps_inherited = build_func_map_inherited_lookup cdecls_inherited in
1181  ignore(check_cyclical_inheritance cdecls predecessors);
1182  let cmaps_with_inherited_fields = inherit_fields class_maps predecessors in
1183  let cmaps_inherited = add_inherited_methods cmaps_with_inherited_fields cdecls_inherited
      ↪  func_maps_inherited in
1184  cmaps_inherited, cdecls_inherited
1185
1186  let generate_struct_indexes cdecls =
1187  let cdecl_handler index cdecl =
1188  Hashtbl.add struct_indexes cdecl.cname index
1189  in
1190  List.iteri cdecl_handler cdecls
1191
1192  (* Main method for analyzer *)
1193  let analyze filename program = match program with
1194  Program(includes, classes) ->
1195  (* Include code from external files *)
1196  let cdecls = process_includes filename includes classes in
1197  ignore(generate_struct_indexes cdecls);
1198
1199  (* Add built-in functions *)
1200  let reserved = add_reserved_functions in
1201  (* Generate the class_maps for look up in checking functions *)
1202  let class_maps = build_class_maps reserved cdecls in
1203  let class_maps, cdecls = handle_inheritance cdecls class_maps in
1204  let sast = convert_cdecls_to_sast class_maps reserved cdecls in
1205  sast
```

ast.ml

```
1  type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq | And | Not |
   ↪  Or | Mod
2  type scope = Private | Public
3  type primitive = Int_t | Float_t | Void_t | Bool_t | Char_t | Objecttype of string |
   ↪  ConstructorType | Null_t
4  type datatype = Arraytype of primitive * int | Datatype of primitive | Any
5
6  type extends = NoParent | Parent of string
7  type fname = Constructor | FName of string
8  type formal = Formal of datatype * string | Many of datatype
9
10 type expr =
11 Int_Lit of int
12 |         Boolean_Lit of bool
13 |         Float_Lit of float
14 |         String_Lit of string
15 |         Char_Lit of char
16 |         This
17 |         Id of string
18 |         Binop of expr * op * expr
19 |         Assign of expr * expr
20 |         Noexpr
21 |         ArrayCreate of datatype * expr list
22 |         ArrayAccess of expr * expr list
23 |         ObjAccess of expr * expr
24 |         Call of string * expr list
25 |    ObjectCreate of string * expr list
26 |         ArrayPrimitive of expr list
27 |          Unop of op * expr
28 |         Null
29 |         Delete of expr
30
31 type stmt =
32 Block of stmt list
33 |         Expr of expr
34 |         Return of expr
35 |         If of expr * stmt * stmt
36 |         For of expr * expr * expr * stmt
37 |         While of expr * stmt
38 |          Break
39 |    Continue
40 |    Local of datatype * string * expr
41
42 type field = Field of scope * datatype * string
43 type include_stmt = Include of string
44
45 type func_decl = {
```

```
46          scope : scope;
47          fname : fname;
48          returnType : datatype;
49          formals : formal list;
50          body : stmt list;
51          overrides : bool;
52          root_cname : string option;
53  }
54
55  type cbody = {
56          fields : field list;
57          constructors : func_decl list;
58          methods : func_decl list;
59  }
60
61  type class_decl = {
62          cname : string;
63          extends : extends;
64          cbody: cbody;
65  }
66
67  type program = Program of include_stmt list * class_decl list
```

bindings.c

```c
#include <stdio.h>
#include <stdlib.h>

#define INIT_SIZE 100

struct s {
        int x;
        int y;
};

char* input() {
        int initial_size = INIT_SIZE;
        char* str = malloc(initial_size);
        int index = 0;
        char tmp = '0';
        while((tmp = getchar() )!= '\n') {
                if(index >= initial_size - 1) {
                        str = realloc(str, initial_size *= 2);
                }
                str[index++] = tmp;
        }
        str[index] = '\0';
        return str;
}

void rec_init(long* arr, int curr_offset, int* static_offsets, int* indexes, int* dims,
   int dimc, int dim_curr) {

        //Assign length
        arr[curr_offset] = dims[dim_curr];

        if(dim_curr + 1 >= dimc)
        return;

        //Determine the static offset and the dynamic offset
        int static_offset = static_offsets[dim_curr];
        int dynamic_offset = 0;
        for(int i = 0; i < dim_curr; i++) {
                int tmp = indexes[i];
                for(int j = i + 1; j <= dim_curr; j++) {
                        tmp *= dims[j];
                }
                dynamic_offset += tmp;
        }

        //Iterate through position and iniitalize subarrays
        //Set local indexes to pointers to the subarrays
```

```
47          for(int i = 0; i < dims[dim_curr]; i++) {
48                  int offset = (static_offset + (dynamic_offset + i) * (dims[dim_curr + 1]
                    ↪  + 1));
49
50                  long* sub = arr + offset;
51                  arr[curr_offset + 1 + i] = (long) sub;
52
53                  indexes[dim_curr] = i;
54                  rec_init(arr, offset, static_offsets, indexes, dims, dimc, dim_curr + 1);
55          }
56  }
57
58  long* init_arr(int* dims, int dimc) {
59
60          int static_offsets[dimc];
61          int total = 0;
62          for(int i = 0; i < dimc; i++) {
63                  static_offsets[i] = 1;
64                  for(int j = 0; j < i; j++) {
65                          static_offsets[i] *= dims[j];
66                  }
67                  static_offsets[i] *= dims[i] + 1;
68                  static_offsets[i] += total;
69                  total = static_offsets[i];
70          }
71
72          int indexes[dimc];
73          for(int i = 0; i < dimc; i++) {
74                  indexes[i] = 0;
75          }
76
77          //Get total length of array
78          int length = 0;
79          for(int i = 0; i < dimc; i++) {
80                  int tmp = 1;
81                  for(int j = i - 1; j >= 0; j--) {
82                          tmp *= dims[j];
83                  }
84                  tmp *= dims[i] + 1;
85                  length += tmp;
86          }
87
88          //Malloc array
89          long* arr = malloc(length);
90
91          //Set all values to 0 initially
92          for(int i = 0 ; i < length; i++) {
93                  arr[i] = 0;
94          }
```

```
95
96          //Initialize the entire array
97          rec_init(arr, 0, static_offsets, indexes, dims, dimc, 0);
98
99          return arr;
100   }
101
102   // int main() {
103
104          //          //Array creation
105          //          int dims[5] = {2, 3, 4, 5, 6};
106          //          int dimc = 5;
107
108          //          long* arr = init_arr(dims, dimc);
109
110          //          //Get total length of array
111          //          int length = 0;
112          //          for(int i = 0; i < dimc; i++) {
113                 //                 int tmp = 1;
114                 //                 for(int j = i - 1; j >= 0; j--) {
115                        //                        tmp *= dims[j];
116                        //                 }
117                 //                 tmp *= dims[i] + 1;
118                 //                 length += tmp;
119                 //          }
120
121          //          for(int i = 0; i < length; i++) {
122                 //                 printf("val: %ld | addr: %ld\n", arr[i], (long) arr +
               ↪  i);
123                 //          }
124          //          printf("\n");
125          // }
```

codegen.ml

```ocaml
(* ===----------------------------------------------------------------===
 * Code Generation
 *===----------------------------------------------------------------===*)

open Llvm
open Ast
open Sast
open Analyzer
open Exceptions
open Batteries
open Hashtbl
open Conf

open Llvm.MemoryBuffer
open Llvm_bitreader

let context = global_context ()
let the_module = create_module context "Dice Codegen"
let builder = builder context
let named_values:(string, llvalue) Hashtbl.t = Hashtbl.create 50
let named_params:(string, llvalue) Hashtbl.t = Hashtbl.create 50
let struct_types:(string, lltype) Hashtbl.t = Hashtbl.create 10
let struct_field_indexes:(string, int) Hashtbl.t = Hashtbl.create 50

let i32_t = i32_type context;;
let i8_t = i8_type context;;
let f_t = double_type context;;
let i1_t = i1_type context;;
let str_t = pointer_type i8_t;;
let i64_t = i64_type context;;
let void_t = void_type context;;

let str_type = Arraytype(Char_t, 1)

let (br_block) = ref (block_of_value (const_int i32_t 0))
let (cont_block) = ref (block_of_value (const_int i32_t 0))
let is_loop = ref false

let debug = fun s ->
print_endline ("''''''''''''''''''''''''''''''''''''''''''''"^s);
dump_module the_module;
print_endline ("''''''''''''''''''''''''''''''''''''''''''''"^s);
()

let rec get_ptr_type datatype = match datatype with
Arraytype(t, 0) -> get_type (Datatype(t))
|        Arraytype(t, 1) -> pointer_type (get_type (Datatype(t)))
```

```
48  |           Arraytype(t, i) -> pointer_type (get_ptr_type (Arraytype(t, (i-1)))))
49  |           _ -> raise(Exceptions.InvalidStructType "Array Pointer Type")

51  and find_struct name =
52  try Hashtbl.find struct_types name
53  with | Not_found -> raise(Exceptions.InvalidStructType name)

55  and get_type (datatype:Ast.datatype) = match datatype with
56  Datatype(Int_t) -> i32_t
57  |           Datatype(Float_t) -> f_t
58  |           Datatype(Bool_t) -> i1_t
59  |           Datatype(Char_t) -> i8_t
60  |           Datatype(Void_t) -> void_t
61  |           Datatype(Null_t) -> i32_t
62  |           Datatype(Objecttype(name)) -> pointer_type(find_struct name)
63  |           Arraytype(t, i) -> get_ptr_type (Arraytype(t, (i)))
64  |           d -> raise(Exceptions.InvalidStructType (Utils.string_of_datatype d))

66  (* cast will return an llvalue of the desired type *)
67  (* The commented out casts are unsupported actions in Dice *)
68  let cast lhs rhs lhsType rhsType llbuilder =
69  match (lhsType, rhsType) with
70  (* int to,__ ) ( using const_sitofp for signed ints *)
71  (Datatype(Int_t), Datatype(Int_t))                              -> (lhs, rhs),
    ↪  Datatype(Int_t)
72  |           (Datatype(Int_t), Datatype(Char_t))                        ->
    ↪  (build_uitofp lhs i8_t "tmp" llbuilder, rhs), Datatype(Char_t)
73  (* |          (Datatype(Int_t), Datatype(Bool_t))                        ->
    ↪  (lhs, const_zext rhs i32_t) *)
74  |   (Datatype(Int_t), Datatype(Float_t))                      -> (build_sitofp lhs f_t
    ↪  "tmp" llbuilder, rhs), Datatype(Float_t)

76  (* char to,__)  ( using uitofp since char isn't signed *)
77  |   (Datatype(Char_t), Datatype(Int_t))                       -> (lhs, build_uitofp rhs
    ↪  i8_t "tmp" llbuilder), Datatype(Char_t)
78  |   (Datatype(Char_t), Datatype(Char_t))                      -> (lhs, rhs),
    ↪  Datatype(Char_t)
79  (* |          (Datatype(Char_t), Datatype(Bool_t))                      -> (lhs,
    ↪  const_zext rhs i8_t) *)
80  (* |          (Datatype(Char_t), Datatype(Float_t))                     ->
    ↪  (const_uitofp lhs f_t, rhs) *)

82  (* bool to,__)  ( zext fills the empty bits with zeros, zero extension *)
83  (* |          (Datatype(Bool_t), Datatype(Int_t))                       -> (const_zext
    ↪  lhs i32_t, rhs) *)
84  (* |          (Datatype(Bool_t), Datatype(Char_t))                      -> (const_zext
    ↪  lhs i8_t, rhs) *)
85  |           (Datatype(Bool_t), Datatype(Bool_t))                     -> (lhs, rhs),
    ↪  Datatype(Bool_t)
```

```
86   (* |            (Datatype(Bool_t), Datatype(Float_t))                    ->
  ↪   (const_uitofp lhs f_t, rhs) *)
87
88   (* float to,__) ( using fptosi for signed ints *)
89   |    (Datatype(Float_t), Datatype(Int_t))                     -> (lhs, build_sitofp
  ↪   rhs f_t "tmp" llbuilder), Datatype(Float_t)
90   (* |           (Datatype(Float_t), Datatype(Char_t))                  -> (lhs,
  ↪   const_uitofp rhs f_t) *)
91   (* |           (Datatype(Float_t), Datatype(Bool_t))                  -> (lhs,
  ↪   const_uitofp rhs f_t) *)
92   |    (Datatype(Float_t), Datatype(Float_t))               -> (lhs, rhs),
  ↪   Datatype(Float_t)
93
94   | Datatype(Objecttype(d)), Datatype(Null_t)                   -> (lhs, rhs), lhsType
95   | Datatype(Null_t), Datatype(Objecttype(d))           -> (rhs, lhs), rhsType
96   | Datatype(Objecttype(d)), t                              ->
  ↪   raise(Exceptions.CanOnlyCompareObjectsWithNull(d, (Utils.string_of_datatype t)))
97
98   | Arraytype(d, s), Datatype(Null_t)                       -> (lhs, rhs),
  ↪   lhsType
99   | Datatype(Null_t), Arraytype(d, s)                       -> (rhs, lhs),
  ↪   rhsType
100  | Arraytype(d, _), t                                          ->
  ↪   raise(Exceptions.CanOnlyCompareArraysWithNull(Utils.string_of_primitive d,
  ↪   (Utils.string_of_datatype t)))
101
102  |        _
  ↪                                                                                ->
  ↪   raise (Exceptions.CannotCastTypeException(Utils.string_of_datatype lhsType,
  ↪   Utils.string_of_datatype rhsType))
103
104  let rec handle_binop e1 op e2 d llbuilder =
105  (* Get the types of e1 and e2 *)
106  let type1 = Analyzer.get_type_from_sexpr e1 in
107  let type2 = Analyzer.get_type_from_sexpr e2 in
108
109  (* Generate llvalues from e1 and e2 *)
110
111  let e1 = codegen_sexpr llbuilder e1 in
112  let e2 = codegen_sexpr llbuilder e2 in
113
114  let float_ops op e1 e2 =
115  match op with
116  Add                -> build_fadd e1 e2 "flt_addtmp" llbuilder
117  |       Sub             -> build_fsub e1 e2 "flt_subtmp" llbuilder
118  |       Mult             -> build_fmul e1 e2 "flt_multmp" llbuilder
119  |       Div             -> build_fdiv e1 e2 "flt_divtmp" llbuilder
120  |       Mod             -> build_frem e1 e2 "flt_sremtmp" llbuilder
121  |       Equal              -> build_fcmp Fcmp.Oeq e1 e2 "flt_eqtmp" llbuilder
```

```
122  |         Neq                   -> build_fcmp Fcmp.One e1 e2 "flt_neqtmp" llbuilder
123  |         Less                   -> build_fcmp Fcmp.Ult e1 e2 "flt_lesstmp" llbuilder
124  |         Leq                   -> build_fcmp Fcmp.Ole e1 e2 "flt_leqtmp" llbuilder
125  |         Greater                 -> build_fcmp Fcmp.Ogt e1 e2 "flt_sgttmp" llbuilder
126  |         Geq                   -> build_fcmp Fcmp.Oge e1 e2 "flt_sgetmp" llbuilder
127  |         _                         -> raise Exceptions.FloatOpNotSupported
128
129  in
130
131  (* chars are considered ints, so they will use int_ops as well*)
132  let int_ops op e1 e2 =
133  match op with
134  Add                 -> build_add e1 e2 "addtmp" llbuilder
135  |         Sub                 -> build_sub e1 e2 "subtmp" llbuilder
136  |         Mult                 -> build_mul e1 e2 "multmp" llbuilder
137  |         Div                 -> build_sdiv e1 e2 "divtmp" llbuilder
138  |         Mod                 -> build_srem e1 e2 "sremtmp" llbuilder
139  |         Equal                 -> build_icmp Icmp.Eq e1 e2 "eqtmp" llbuilder
140  |         Neq                 -> build_icmp Icmp.Ne e1 e2 "neqtmp" llbuilder
141  |         Less                 -> build_icmp Icmp.Slt e1 e2 "lesstmp" llbuilder
142  |         Leq                 -> build_icmp Icmp.Sle e1 e2 "leqtmp" llbuilder
143  |         Greater                 -> build_icmp Icmp.Sgt e1 e2 "sgttmp" llbuilder
144  |         Geq                 -> build_icmp Icmp.Sge e1 e2 "sgetmp" llbuilder
145  |         And                 -> build_and e1 e2 "andtmp" llbuilder
146  |         Or                       -> build_or  e1 e2 "ortmp" llbuilder
147  |         _                         -> raise Exceptions.IntOpNotSupported
148  in
149
150  let obj_ops op e1 e2 =
151  match op with
152  Equal -> build_is_null e1 "tmp" llbuilder
153  |         Neq -> build_is_not_null e1 "tmp" llbuilder
154  |         _             -> raise (Exceptions.ObjOpNotSupported(Utils.string_of_op op))
155  in
156
157  let (e1, e2), d = cast e1 e2 type1 type2 llbuilder in
158
159  let type_handler d = match d with
160  Datatype(Float_t)   -> float_ops op e1 e2
161  |         Datatype(Int_t)
162  |    Datatype(Bool_t)
163  |         Datatype(Char_t)         -> int_ops op e1 e2
164  |         Datatype(Objecttype(_))
165  |         Arraytype(_, _) -> obj_ops op e1 e2
166  |   _ -> raise Exceptions.InvalidBinopEvaluationType
167  in
168
169  type_handler d
170
```

```
171  and handle_unop op e d llbuilder =
172  (* Get the type of e *)
173  let eType = Analyzer.get_type_from_sexpr e in
174  (* Get llvalue  *)
175  let e = codegen_sexpr llbuilder e in
176
177  let unops op eType e = match (op, eType) with
178  (Sub, Datatype(Int_t))                   ->  build_neg e "int_unoptmp" llbuilder
179  |  (Sub, Datatype(Float_t))        ->         build_fneg e "flt_unoptmp" llbuilder
180  |   (Not, Datatype(Bool_t))        ->  build_not e "bool_unoptmp" llbuilder
181  |     _           -> raise Exceptions.UnopNotSupported        in
182
183  let unop_type_handler d = match d with
184  Datatype(Float_t)
185  |       Datatype(Int_t)
186  |   Datatype(Bool_t)         -> unops op eType e
187  |   _ -> raise Exceptions.InvalidUnopEvaluationType
188  in
189
190  unop_type_handler d
191
192  and func_lookup fname =
193  match (lookup_function fname the_module) with
194  None          -> raise (Exceptions.LLVMFunctionNotFound fname)
195  |        Some f         -> f
196
197  and codegen_print el llbuilder =
198  let printf = func_lookup "printf" in
199  let tmp_count = ref 0 in
200  let incr_tmp = fun x -> incr tmp_count in
201
202  let map_expr_to_printfexpr expr =
203  let exprType = Analyzer.get_type_from_sexpr expr in
204  match exprType with
205  Datatype(Bool_t) ->
206  incr_tmp ();
207  let tmp_var = "tmp" ^ (string_of_int !tmp_count) in
208  let trueStr = SString_Lit("true") in
209  let falseStr = SString_Lit("false") in
210  let id = SId(tmp_var, str_type) in
211  ignore(codegen_stmt llbuilder (SLocal(str_type, tmp_var, SNoexpr)));
212  ignore(codegen_stmt llbuilder (SIf(expr,
213  SExpr(SAssign(id, trueStr, str_type), str_type),
214  SExpr(SAssign(id, falseStr, str_type), str_type)
215  )));
216  codegen_sexpr llbuilder id
217  | _ -> codegen_sexpr llbuilder expr
218  in
219
```

```
220  let params = List.map map_expr_to_printfexpr el in
221  let param_types = List.map (Analyzer.get_type_from_sexpr) el in
222
223  let map_param_to_string = function
224  Arraytype(Char_t, 1)          -> "%s"
225  |        Datatype(Int_t)              -> "%d"
226  |        Datatype(Float_t)             -> "%f"
227  |        Datatype(Bool_t)             -> "%s"
228  |        Datatype(Char_t)             -> "%c"
229  |            _                                      -> raise
     ↪  (Exceptions.InvalidTypePassedToPrintf)
230  in
231  let const_str = List.fold_left (fun s t -> s ^ map_param_to_string t) "" param_types in
232  let s = codegen_sexpr llbuilder (SString_Lit(const_str)) in
233  let zero = const_int i32_t 0 in
234  let s = build_in_bounds_gep s [| zero |] "tmp" llbuilder in
235  build_call printf (Array.of_list (s :: params)) "tmp" llbuilder
236
237  and codegen_func_call fname el d llbuilder =
238  let f = func_lookup fname in
239  let params = List.map (codegen_sexpr llbuilder) el in
240  match d with
241  Datatype(Void_t) -> build_call f (Array.of_list params) "" llbuilder
242  |         _ ->                         build_call f (Array.of_list params) "tmp"
     ↪  llbuilder
243
244  and codegen_sizeof el llbuilder =
245  let type_of = Analyzer.get_type_from_sexpr (List.hd el) in
246  let type_of = get_type type_of in
247  let size_of = size_of type_of in
248  build_bitcast size_of i32_t "tmp" llbuilder
249
250  and codegen_cast el d llbuilder =
251  let cast_malloc_to_objtype lhs currType newType llbuilder = match newType with
252  Datatype(Objecttype(x)) ->
253  let obj_type = get_type (Datatype(Objecttype(x))) in
254  build_pointercast lhs obj_type "tmp" llbuilder
255  |          _ as t -> raise (Exceptions.CannotCastTypeException(Utils.string_of_datatype
     ↪  currType, Utils.string_of_datatype t))
256  in
257  let expr = List.hd el in
258  let t = Analyzer.get_type_from_sexpr expr in
259  let lhs = match expr with
260  |        Sast.SId(id, d) -> codegen_id false false id d llbuilder
261  |         SObjAccess(e1, e2, d) -> codegen_obj_access false e1 e2 d llbuilder
262  |        SArrayAccess(se, sel, d) -> codegen_array_access true se sel d llbuilder
263  | _ -> codegen_sexpr llbuilder expr
264  in
265  cast_malloc_to_objtype lhs t d llbuilder
```

```
266
267  and codegen_call llbuilder d el = function
268  "print"          -> codegen_print el llbuilder
269  |        "sizeof"        -> codegen_sizeof el llbuilder
270  |        "cast"            -> codegen_cast el d llbuilder
271  |        "malloc"          -> codegen_func_call "malloc" el d llbuilder
272  |        "open"            -> codegen_func_call "open" el d llbuilder
273  |        "write"           -> codegen_func_call "write" el d llbuilder
274  |        "close"           -> codegen_func_call "close" el d llbuilder
275  |        "read"            -> codegen_func_call "read" el d llbuilder
276  |        "lseek"         -> codegen_func_call "lseek" el d llbuilder
277  |        "exit"            -> codegen_func_call "exit" el d llbuilder
278  |        "input"         -> codegen_func_call "input" el d llbuilder
279  |    "getchar"   -> codegen_func_call "getchar" el d llbuilder
280  |         _ as fname          -> raise (Exceptions.UnableToCallFunctionWithoutParent
     ↪ fname)(* codegen_func_call fname el llbuilder *)
281
282  and codegen_id isDeref checkParam id d llbuilder =
283  if isDeref then
284  try Hashtbl.find named_params id
285  with | Not_found ->
286  try let _val = Hashtbl.find named_values id in
287  build_load _val id llbuilder
288  with | Not_found -> raise (Exceptions.UnknownVariable id)
289  else
290  try Hashtbl.find named_values id
291  with | Not_found ->
292  try
293  let _val = Hashtbl.find named_params id in
294  if checkParam then raise (Exceptions.CannotAssignParam id)
295  else _val
296  with | Not_found -> raise (Exceptions.UnknownVariable id)
297
298  and codegen_assign lhs rhs d llbuilder =
299  let rhsType = Analyzer.get_type_from_sexpr rhs in
300  (* Special case '=' because we don't want to emit the LHS as an
301  * expression. *)
302  let lhs, isObjAccess = match lhs with
303  |        Sast.SId(id, d) -> codegen_id false false id d llbuilder, false
304  |         SObjAccess(e1, e2, d) -> codegen_obj_access false e1 e2 d llbuilder, true
305  |        SArrayAccess(se, sel, d) -> codegen_array_access true se sel d llbuilder, true
306  | _ -> raise Exceptions.AssignLHSMustBeAssignable
307  in
308  (* Codegen the rhs. *)
309  let rhs = match rhs with
310  |        Sast.SId(id, d) -> codegen_id false false id d llbuilder
311  |         SObjAccess(e1, e2, d) -> codegen_obj_access true e1 e2 d llbuilder
312  | _ -> codegen_sexpr llbuilder rhs
313  in
```

```
314   let rhs = match d with
315     Datatype(Objecttype(_))          ->
316     if isObjAccess then rhs
317     else build_load rhs "tmp" llbuilder
318   |           Datatype(Null_t) -> const_null (get_type d)
319   | _ -> rhs
320   in
321   let rhs = match d, rhsType with
322     Datatype(Char_t), Datatype(Int_t) -> build_uitofp rhs i8_t "tmp" llbuilder
323   |           Datatype(Int_t), Datatype(Char_t) -> build_uitofp rhs i32_t "tmp" llbuilder
324   |           _ -> rhs
325   in
326   (* Lookup the name. *)
327   ignore(build_store rhs lhs llbuilder);
328   rhs
329
330   and deref ptr t llbuilder =
331   build_gep ptr (Array.of_list [ptr]) "tmp" llbuilder
332
333   and codegen_obj_access isAssign lhs rhs d llbuilder =
334   let codegen_func_call param_ty fptr parent_expr el d llbuilder =
335   let match_sexpr se = match se with
336     SId(id, d) -> let isDeref = match d with
337     Datatype(Objecttype(_)) -> false
338   |           _ -> true
339   in codegen_id isDeref false id d llbuilder
340   |           se -> codegen_sexpr llbuilder se
341   in
342   let parent_expr = build_pointercast parent_expr param_ty "tmp" llbuilder in
343   let params = List.map match_sexpr el in
344   match d with
345     Datatype(Void_t) -> build_call fptr (Array.of_list (parent_expr :: params)) "" llbuilder
346   |           _ -> build_call fptr (Array.of_list (parent_expr :: params)) "tmp" llbuilder
347   in
348   let check_lhs = function
349     SId(s, d)                               -> codegen_id false false s d llbuilder
350   |           SArrayAccess(e, el, d)        -> codegen_array_access false e el d llbuilder
351   |           se           -> raise (Exceptions.LHSofRootAccessMustBeIDorFunc
    ↪ (Utils.string_of_sexpr se))
352   in
353   (* Needs to be changed *)
354   let rec check_rhs isLHS parent_expr parent_type =
355   let parent_str = Utils.string_of_object parent_type in
356   function
357   (* Check fields in parent *)
358     SId(field, d) ->
359   let search_term = (parent_str ^ "." ^ field) in
360   let field_index = Hashtbl.find struct_field_indexes search_term in
361   let _val = build_struct_gep parent_expr field_index field llbuilder in
```

```
362    let _val = match d with
363    Datatype(Objecttype(_)) ->
364    if not isAssign then _val
365    else build_load _val field llbuilder
366    | _ ->
367    if not isAssign then
368    _val
369    else
370    build_load _val field llbuilder
371    in
372    _val
373
374    |        SArrayAccess(e, el, d) ->
375
376    let ce = check_rhs false parent_expr parent_type e in
377    let index = codegen_sexpr llbuilder (List.hd el) in
378    let index = match d with
379    Datatype(Char_t) -> index
380    |        _ -> build_add index (const_int i32_t 1) "tmp" llbuilder
381    in
382    let _val = build_gep ce [| index |] "tmp" llbuilder in
383    if isLHS && isAssign
384    then _val
385    else build_load _val "tmp" llbuilder
386
387    (* Check functions in parent *)
388    |        SCall(fname, el, d, index)         ->
389    let index = const_int i32_t index in
390    let c_index = build_struct_gep parent_expr 0 "cindex" llbuilder in
391    let c_index = build_load c_index "cindex" llbuilder in
392    let lookup = func_lookup "lookup" in
393    let fptr = build_call lookup [| c_index; index |] "fptr" llbuilder in
394    let fptr2 = func_lookup fname in
395    let f_ty = type_of fptr2 in
396    let param1 = param fptr2 0 in
397    let param_ty = type_of param1 in
398    let fptr = build_pointercast fptr f_ty fname llbuilder in
399    let ret = codegen_func_call param_ty fptr parent_expr el d llbuilder in
400    let ret = ret
401    (* if not isLHS && not isAssign then
402    build_load ret "tmp" llbuilder
403    else
404    ret *)
405    in
406    ret
407    (* Set parent, check if base is field *)
408    |        SObjAccess(e1, e2, d)         ->
409    let e1_type = Analyzer.get_type_from_sexpr e1 in
410    let e1 = check_rhs true parent_expr parent_type e1 in
```

```
411    let e2 = check_rhs true e1 e1_type e2 in
412    e2
413    |        _ as e -> raise (Exceptions.InvalidAccessLHS (Utils.string_of_sexpr e))
414    in
415    let lhs_type = Analyzer.get_type_from_sexpr lhs in
416    match lhs_type with
417    Arraytype(_, _) ->
418    let lhs = codegen_sexpr llbuilder lhs in
419    let _ = match rhs with
420    SId("length", _) -> "length"
421    |        _ -> raise(Exceptions.CanOnlyAccessLengthOfArray)
422    in
423    let _val = build_gep lhs [| (const_int i32_t 0) |] "tmp" llbuilder in
424    build_load _val "tmp" llbuilder
425    |        _ ->
426    let lhs = check_lhs lhs in
427    let rhs = check_rhs true lhs lhs_type rhs in
428    rhs
429
430    and codegen_obj_create fname el d llbuilder =
431    let f = func_lookup fname in
432    let params = List.map (codegen_sexpr llbuilder) el in
433    let obj = build_call f (Array.of_list params) "tmp" llbuilder in
434    obj
435
436    and codegen_string_lit s llbuilder =
437    if s = "true" then build_global_stringptr "true" "tmp" llbuilder
438    else if s = "false" then build_global_stringptr "false" "tmp" llbuilder
439    else build_global_stringptr s "tmp" llbuilder
440
441    and codegen_array_access isAssign e el d llbuilder =
442    let index = codegen_sexpr llbuilder (List.hd el) in
443    let index = match d with
444    Datatype(Char_t) -> index
445    |        _ -> build_add index (const_int i32_t 1) "tmp" llbuilder
446    in
447    let arr = codegen_sexpr llbuilder e in
448    let _val = build_gep arr [| index |] "tmp" llbuilder in
449    if isAssign
450    then _val
451    else build_load _val "tmp" llbuilder
452
453    and initialise_array arr arr_len init_val start_pos llbuilder =
454    let new_block label =
455    let f = block_parent (insertion_block llbuilder) in
456    append_block (global_context ()) label f
457    in
458    let bbcurr = insertion_block llbuilder in
459    let bbcond = new_block "array.cond" in
```

```
460   let bbbody = new_block "array.init" in
461   let bbdone = new_block "array.done" in
462   ignore (build_br bbcond llbuilder);
463   position_at_end bbcond llbuilder;
464
465   (* Counter into the length of the array *)
466   let counter = build_phi [const_int i32_t start_pos, bbcurr] "counter" llbuilder in
467   add_incoming ((build_add counter (const_int i32_t 1) "tmp" llbuilder), bbbody) counter;
468   let cmp = build_icmp Icmp.Slt counter arr_len "tmp" llbuilder in
469   ignore (build_cond_br cmp bbbody bbdone llbuilder);
470   position_at_end bbbody llbuilder;
471
472   (* Assign array position to init_val *)
473   let arr_ptr = build_gep arr [| counter |] "tmp" llbuilder in
474   ignore (build_store init_val arr_ptr llbuilder);
475   ignore (build_br bbcond llbuilder);
476   position_at_end bbdone llbuilder
477
478   and codegen_array_create llbuilder t expr_type el =
479   if(List.length el > 1) then raise(Exceptions.ArrayLargerThan1Unsupported)
480   else
481   match expr_type with
482   Arraytype(Char_t, 1) ->
483   let e = List.hd el in
484   let size = (codegen_sexpr llbuilder e) in
485   let t = get_type t in
486   let arr = build_array_malloc t size "tmp" llbuilder in
487   let arr = build_pointercast arr (pointer_type t) "tmp" llbuilder in
488   (* initialise_array arr size (const_int i32_t 0) 0 llbuilder; *)
489   arr
490   |           _ ->
491   let e = List.hd el in
492   let t = get_type t in
493
494   (* This will not work for arrays of objects *)
495   let size = (codegen_sexpr llbuilder e) in
496   let size_t = build_intcast (size_of t) i32_t "tmp" llbuilder in
497   let size = build_mul size_t size "tmp" llbuilder in
498   let size_real = build_add size (const_int i32_t 1) "arr_size" llbuilder in
499
500   let arr = build_array_malloc t size_real "tmp" llbuilder in
501   let arr = build_pointercast arr (pointer_type t) "tmp" llbuilder in
502
503   let arr_len_ptr = build_pointercast arr (pointer_type i32_t) "tmp" llbuilder in
504
505   (* Store length at this position *)
506   ignore(build_store size_real arr_len_ptr llbuilder);
507   initialise_array arr_len_ptr size_real (const_int i32_t 0) 0 llbuilder;
508   arr
```

```
509
510   and codegen_array_prim d el llbuilder =
511   let t = d in
512   let size = (const_int i32_t ((List.length el))) in
513   let size_real = (const_int i32_t ((List.length el) + 1)) in
514   let t = get_type t in
515   let arr = build_array_malloc t size_real "tmp" llbuilder in
516   let arr = build_pointercast arr t "tmp" llbuilder in
517   let size_casted = build_bitcast size t "tmp" llbuilder in
518   ignore(if d = Arraytype(Char_t, 1) then ignore(build_store size_casted arr llbuilder););
      ↪  (* Store length at this position *)
519   (* initialise_array arr size_real (const_int i32_t 0) 1 llbuilder; *)
520
521   let llvalues = List.map (codegen_sexpr llbuilder) el in
522   List.iteri (fun i llval ->
523   let arr_ptr = build_gep arr [| (const_int i32_t (i+1)) |] "tmp" llbuilder in
524   ignore(build_store llval arr_ptr llbuilder);  ) llvalues;
525   arr
526
527   and codegen_delete e llbuilder =
528   let ce = match e with
529   SId(id, d) -> codegen_id false false id d llbuilder
530   |        _ -> codegen_sexpr llbuilder e
531   in
532   build_free ce llbuilder
533
534   and codegen_sexpr llbuilder = function
535   SInt_Lit(i)                        -> const_int i32_t i
536   |    SBoolean_Lit(b)                   -> if b then const_int i1_t 1 else const_int
      ↪  i1_t 0
537   |    SFloat_Lit(f)                     -> const_float f_t f
538   |    SString_Lit(s)                    -> codegen_string_lit s llbuilder
539   |    SChar_Lit(c)                      -> const_int i8_t (Char.code c)
540   |    SId(id, d)                   -> codegen_id true false id d llbuilder
541   |    SBinop(e1, op, e2, d)         -> handle_binop e1 op e2 d llbuilder
542   |    SAssign(e1, e2, d)            -> codegen_assign e1 e2 d llbuilder
543   |    SNoexpr                     -> build_add (const_int i32_t 0) (const_int i32_t 0)
      ↪  "nop" llbuilder
544   |    SArrayCreate(t, el, d)        -> codegen_array_create llbuilder t d el
545   |    SArrayAccess(e, el, d)        -> codegen_array_access false e el d llbuilder
546   |    SObjAccess(e1, e2, d)         -> codegen_obj_access true e1 e2 d llbuilder
547   |    SCall(fname, el, d, _)         -> codegen_call llbuilder d el fname
548   |    SObjectCreate(id, el, d)      -> codegen_obj_create id el d llbuilder
549   |    SArrayPrimitive(el, d)        -> codegen_array_prim d el llbuilder
550   |    SUnop(op, e, d)               -> handle_unop op e d llbuilder
551   |    SNull                        -> const_null i32_t
552   |        SDelete e                              -> codegen_delete e
      ↪  llbuilder
553
```

```
554  and codegen_if_stmt exp then_ (else_:Sast.sstmt) llbuilder =
555  let cond_val = codegen_sexpr llbuilder exp in
556
557  (* Grab the first block so that we might later add the conditional branch
558   * to it at the end of the function. *)
559  let start_bb = insertion_block llbuilder in
560  let the_function = block_parent start_bb in
561
562  let then_bb = append_block context "then" the_function in
563
564  (* Emit 'then' value. *)
565  position_at_end then_bb llbuilder;
566  let _(* then_val *) = codegen_stmt llbuilder then_ in
567
568  (* Codegen of 'then' can change the current block, update then_bb for the
569   * phi. We create a new name because one is used for the phi node, and the
570   * other is used for the conditional branch. *)
571  let new_then_bb = insertion_block llbuilder in
572
573  (* Emit 'else' value. *)
574  let else_bb = append_block context "else" the_function in
575  position_at_end else_bb llbuilder;
576  let _ (* else_val *) = codegen_stmt llbuilder else_ in
577
578  (* Codegen of 'else' can change the current block, update else_bb for the
579   * phi. *)
580  let new_else_bb = insertion_block llbuilder in
581
582
583  let merge_bb = append_block context "ifcont" the_function in
584  position_at_end merge_bb llbuilder;
585  (* let then_bb_val = value_of_block new_then_bb in *)
586  let else_bb_val = value_of_block new_else_bb in
587  (* let incoming = [(then_bb_val, new_then_bb); (else_bb_val, new_else_bb)] in *)
588  (* let phi = build_phi incoming "iftmp" llbuilder in *)
589
590  (* Return to the start block to add the conditional branch. *)
591  position_at_end start_bb llbuilder;
592  ignore (build_cond_br cond_val then_bb else_bb llbuilder);
593
594  (* Set a unconditional branch at the end of the 'then' block and the
595   * 'else' block to the 'merge' block. *)
596  position_at_end new_then_bb llbuilder; ignore (build_br merge_bb llbuilder);
597  position_at_end new_else_bb llbuilder; ignore (build_br merge_bb llbuilder);
598
599  (* Finally, set the builder to the end of the merge block. *)
600  position_at_end merge_bb llbuilder;
601
602  else_bb_val (* phi *)
```

```
603
604   and codegen_for init_ cond_ inc_ body_ llbuilder =
605   let old_val = !is_loop in
606   is_loop := true;
607
608   let the_function = block_parent (insertion_block llbuilder) in
609
610   (* Emit the start code first, without 'variable' in scope. *)
611   let _ = codegen_sexpr llbuilder init_ in
612
613   (* Make the new basic block for the loop header, inserting after current
614    * block. *)
615   let loop_bb = append_block context "loop" the_function in
616   (* Insert maintenance block *)
617   let inc_bb = append_block context "inc" the_function in
618   (* Insert condition block *)
619   let cond_bb = append_block context "cond" the_function in
620   (* Create the "after loop" block and insert it. *)
621   let after_bb = append_block context "afterloop" the_function in
622
623   let _ = if not old_val then
624   cont_block := inc_bb;
625   br_block := after_bb;
626   in
627
628   (* Insert an explicit fall through from the current block to the
629    * loop_bb. *)
630   ignore (build_br cond_bb llbuilder);
631
632   (* Start insertion in loop_bb. *)
633   position_at_end loop_bb llbuilder;
634
635   (* Emit the body of the loop.  This, like any other expr, can change the
636    * current BB.  Note that we ignore the value computed by the body, but
637    * don't allow an error *)
638   ignore (codegen_stmt llbuilder body_);
639
640   let bb = insertion_block llbuilder in
641   move_block_after bb inc_bb;
642   move_block_after inc_bb cond_bb;
643   move_block_after cond_bb after_bb;
644   ignore(build_br inc_bb llbuilder);
645
646   (* Start insertion in loop_bb. *)
647   position_at_end inc_bb llbuilder;
648   (* Emit the step value. *)
649   let _ = codegen_sexpr llbuilder inc_ in
650   ignore(build_br cond_bb llbuilder);
651
```

```
652    position_at_end cond_bb llbuilder;
653
654    let cond_val = codegen_sexpr llbuilder cond_ in
655    ignore (build_cond_br cond_val loop_bb after_bb llbuilder);
656
657    (* Any new code will be inserted in after_bb. *)
658    position_at_end after_bb llbuilder;
659
660    is_loop := old_val;
661
662    (* for expr always returns 0.0. *)
663    const_null f_t
664
665    and codegen_while cond_ body_ llbuilder =
666    let null_sexpr = SInt_Lit(0) in
667    codegen_for null_sexpr cond_ null_sexpr body_ llbuilder
668
669    and codegen_alloca datatype var_name expr llbuilder =
670    let t = match datatype with
671    Datatype(Objecttype(name)) -> find_struct name
672    |           _ -> get_type datatype
673    in
674    let alloca = build_alloca t var_name llbuilder in
675    Hashtbl.add named_values var_name alloca;
676    let lhs = SId(var_name, datatype) in
677    match expr with
678    SNoexpr -> alloca
679    |           _ -> codegen_assign lhs expr datatype llbuilder
680
681    and codegen_ret d expr llbuilder =
682    match expr with
683    SId(name, d) ->
684    (match d with
685    | Datatype(Objecttype(_)) -> build_ret (codegen_id false false name d llbuilder)
           ↪  llbuilder
686    | _ -> build_ret (codegen_id true true name d llbuilder) llbuilder)
687    | SObjAccess(e1, e2, d) -> build_ret (codegen_obj_access true e1 e2 d llbuilder)
           ↪  llbuilder
688    | SNoexpr -> build_ret_void llbuilder
689    | _ -> build_ret (codegen_sexpr llbuilder expr) llbuilder
690
691    and codegen_break llbuilder =
692    let block = fun () -> !br_block in
693    build_br (block ()) llbuilder
694
695    and codegen_continue llbuilder =
696    let block = fun () -> !cont_block in
697    build_br (block ()) llbuilder
698
```

```ocaml
699  and codegen_stmt llbuilder = function
700    SBlock sl                                    -> List.hd(List.map (codegen_stmt llbuilder) sl)
701    |   SExpr(e, d)                              -> codegen_sexpr llbuilder e
702    |   SReturn(e, d)                            -> codegen_ret d e llbuilder
703    |   SIf (e, s1, s2)                          -> codegen_if_stmt e s1 s2 llbuilder
704    |   SFor (e1, e2, e3, s)                     -> codegen_for e1 e2 e3 s llbuilder
705    |   SWhile (e, s)                            -> codegen_while e s llbuilder
706    |   SBreak                                   -> codegen_break llbuilder
707    |   SContinue                                -> codegen_continue llbuilder
708    |   SLocal(d, s, e)                          -> codegen_alloca d s e llbuilder
709
710    let codegen_funcstub sfdecl =
711    let fname = (Utils.string_of_fname sfdecl.sfname) in
712    let is_var_arg = ref false in
713    let params = List.rev (List.fold_left (fun l -> (function Formal(t, _) -> get_type t :: l
     ↪   | _ -> is_var_arg := true; l )) [] sfdecl.sformals) in
714    let fty = if !is_var_arg
715    then var_arg_function_type (get_type sfdecl.sreturnType) (Array.of_list params)
716    else function_type (get_type sfdecl.sreturnType) (Array.of_list params)
717    in
718    define_function fname fty the_module
719
720    let init_params f formals =
721    let formals = Array.of_list (formals) in
722    Array.iteri (fun i a ->
723    let n = formals.(i) in
724    let n = Utils.string_of_formal_name n in
725    set_value_name n a;
726    Hashtbl.add named_params n a;
727    ) (params f)
728
729    let codegen_func sfdecl =
730    Hashtbl.clear named_values;
731    Hashtbl.clear named_params;
732    let fname = (Utils.string_of_fname sfdecl.sfname) in
733    let f = func_lookup fname in
734    let llbuilder = builder_at_end context (entry_block f) in
735    let _ = init_params f sfdecl.sformals in
736    let _ = if sfdecl.overrides then
737    let this_param = Hashtbl.find named_params "this" in
738    let source = Datatype(Objecttype(sfdecl.source)) in
739    let casted_param = build_pointercast this_param (get_type source) "casted" llbuilder in
740    Hashtbl.replace named_params "this" casted_param;
741    in
742    let _ = codegen_stmt llbuilder (SBlock (sfdecl.sbody)) in
743    if sfdecl.sreturnType = Datatype(Void_t)
744    then ignore(build_ret_void llbuilder);
745    ()
746
```

```
747  let codegen_vtbl scdecls =
748  let rt = pointer_type i64_t in
749  let void_pt = pointer_type i64_t in
750  let void_ppt = pointer_type void_pt in
751
752  let f = func_lookup "lookup" in
753  let llbuilder = builder_at_end context (entry_block f) in
754
755  let len = List.length scdecls in
756  let total_len = ref 0 in
757  let scdecl_llvm_arr = build_array_alloca void_ppt (const_int i32_t len) "tmp" llbuilder
     ↪  in
758
759  let handle_scdecl scdecl =
760  let index = Hashtbl.find Analyzer.struct_indexes scdecl.scname in
761  let len = List.length scdecl.sfuncs in
762  let sfdecl_llvm_arr = build_array_alloca void_pt (const_int i32_t len) "tmp" llbuilder in
763
764  let handle_fdecl i sfdecl =
765  let fptr = func_lookup (Utils.string_of_fname sfdecl.sfname) in
766  let fptr = build_pointercast fptr void_pt "tmp" llbuilder in
767
768  let ep = build_gep sfdecl_llvm_arr [| (const_int i32_t i) |] "tmp" llbuilder in
769  ignore(build_store fptr ep llbuilder);
770  in
771  List.iteri handle_fdecl scdecl.sfuncs;
772  total_len := !total_len + len;
773
774  let ep = build_gep scdecl_llvm_arr [| (const_int i32_t index) |] "tmp" llbuilder in
775  ignore(build_store sfdecl_llvm_arr ep llbuilder);
776  in
777  List.iter handle_scdecl scdecls;
778
779  let c_index = param f 0 in
780  let f_index = param f 1 in
781  set_value_name "c_index" c_index;
782  set_value_name "f_index" f_index;
783
784  if !total_len == 0 then
785  build_ret (const_null rt) llbuilder
786  else
787  let vtbl = build_gep scdecl_llvm_arr [| c_index |] "tmp" llbuilder in
788  let vtbl = build_load vtbl "tmp" llbuilder in
789  let fptr = build_gep vtbl [| f_index |] "tmp" llbuilder in
790  let fptr = build_load fptr "tmp" llbuilder in
791
792  build_ret fptr llbuilder
793
794  let codegen_library_functions () =
```

```ocaml
795    (* C Std lib functions *)
796    let printf_ty = var_arg_function_type i32_t [| pointer_type i8_t |] in
797    let _ = declare_function "printf" printf_ty the_module in
798    let malloc_ty = function_type (str_t) [| i32_t |] in
799    let _ = declare_function "malloc" malloc_ty the_module in
800    let open_ty = function_type i32_t [| (pointer_type i8_t); i32_t |] in
801    let _ = declare_function "open" open_ty the_module in
802    let close_ty = function_type i32_t [| i32_t |] in
803    let _ = declare_function "close" close_ty the_module in
804    let read_ty = function_type i32_t [| i32_t; pointer_type i8_t; i32_t |] in
805    let _ = declare_function "read" read_ty the_module in
806    let write_ty = function_type i32_t [| i32_t; pointer_type i8_t; i32_t |] in
807    let _ = declare_function "write" write_ty the_module in
808    let lseek_ty = function_type i32_t [| i32_t; i32_t; i32_t |] in
809    let _ = declare_function "lseek" lseek_ty the_module in
810    let exit_ty = function_type void_t [| i32_t |] in
811    let _ = declare_function "exit" exit_ty the_module in
812    let realloc_ty = function_type str_t [| str_t; i32_t |] in
813    let _ = declare_function "realloc" realloc_ty the_module in
814    let getchar_ty = function_type (i32_t) [| |] in
815    let _ = declare_function "getchar" getchar_ty the_module in
816
817    (* Dice defined functions *)
818    let fty = function_type (pointer_type i64_t) [| i32_t; i32_t |] in
819    let _ = define_function "lookup" fty the_module in
820    let rec_init_ty = function_type void_t [| (pointer_type i64_t); i32_t; (pointer_type
       ↪ i32_t); (pointer_type i32_t); (pointer_type i32_t); i32_t; i32_t |] in
821    let _ = declare_function "rec_init" rec_init_ty the_module in
822    let init_arr_ty = function_type (pointer_type i64_t) [| (pointer_type i32_t); i32_t |] in
823    let _ = declare_function "init_arr" init_arr_ty the_module in
824    let input_ty = function_type str_t [||] in
825    let _ = declare_function "input" input_ty the_module in
826    ()
827
828    let codegen_struct_stub s =
829    let struct_t = named_struct_type context s.scname in
830    Hashtbl.add struct_types s.scname struct_t
831
832    let codegen_struct s =
833    let struct_t = Hashtbl.find struct_types s.scname in
834    let type_list = List.map (function Field(_, d, _) -> get_type d) s.sfields in
835    let name_list = List.map (function Field(_, _, s) -> s) s.sfields in
836
837    (* Add key field to all structs *)
838    let type_list = i32_t :: type_list in
839    let name_list = ".key" :: name_list in
840
841    let type_array = (Array.of_list type_list) in
842    List.iteri (fun i f ->
```

```
843    let n = s.scname ^ "." ^ f in
844    Hashtbl.add struct_field_indexes n i;
845    ) name_list;
846    struct_set_body struct_t type_array true
847
848    let init_args argv args argc llbuilder =
849    let new_block label =
850    let f = block_parent (insertion_block llbuilder) in
851    append_block (global_context ()) label f
852    in
853    let bbcurr = insertion_block llbuilder in
854    let bbcond = new_block "args.cond" in
855    let bbbody = new_block "args.init" in
856    let bbdone = new_block "args.done" in
857    ignore (build_br bbcond llbuilder);
858    position_at_end bbcond llbuilder;
859
860    (* Counter into the length of the array *)
861    let counter = build_phi [const_int i32_t 0, bbcurr] "counter" llbuilder in
862    add_incoming ((build_add counter (const_int i32_t 1) "tmp" llbuilder), bbbody) counter;
863    let cmp = build_icmp Icmp.Slt counter argc "tmp" llbuilder in
864    ignore (build_cond_br cmp bbbody bbdone llbuilder);
865    position_at_end bbbody llbuilder;
866
867    (* Assign array position to init_val *)
868    let arr_ptr = build_gep args [| counter |] "tmp" llbuilder in
869    let argv_val = build_gep argv [| counter |] "tmp" llbuilder in
870    let argv_val = build_load argv_val "tmp" llbuilder in
871    ignore (build_store argv_val arr_ptr llbuilder);
872    ignore (build_br bbcond llbuilder);
873    position_at_end bbdone llbuilder
874
875    let construct_args argc argv llbuilder =
876    let str_pt = pointer_type str_t in
877    let size_real = build_add argc (const_int i32_t 1) "arr_size" llbuilder in
878
879    let arr = build_array_malloc str_pt size_real "args" llbuilder in
880    let arr = build_pointercast arr str_pt "args" llbuilder in
881    let arr_len_ptr = build_pointercast arr (pointer_type i32_t) "argc_len" llbuilder in
882    let arr_1 = build_gep arr [| const_int i32_t 1 |] "arr_1" llbuilder in
883
884    (* Store length at this position *)
885    ignore(build_store argc arr_len_ptr llbuilder);
886    ignore(init_args argv arr_1 argc llbuilder);
887    arr
888
889    let codegen_main main =
890    Hashtbl.clear named_values;
891    Hashtbl.clear named_params;
```

```
892  let fty = function_type i32_t [| i32_t; pointer_type str_t |] in
893  let f = define_function "main" fty the_module in
894  let llbuilder = builder_at_end context (entry_block f) in
895
896  let argc = param f 0 in
897  let argv = param f 1 in
898  set_value_name "argc" argc;
899  set_value_name "argv" argv;
900  let args = construct_args argc argv llbuilder in
901  Hashtbl.add named_params "args" args;
902
903  let _ = codegen_stmt llbuilder (SBlock (main.sbody)) in
904  build_ret (const_int i32_t 0) llbuilder
905
906  let linker filename =
907  let llctx = Llvm.global_context () in
908  let llmem = Llvm.MemoryBuffer.of_file filename in
909  let llm = Llvm_bitreader.parse_bitcode llctx llmem in
910  ignore(Llvm_linker.link_modules the_module llm)
911
912  let codegen_sprogram sprogram =
913  let _ = codegen_library_functions () in
914  let _ = List.map (fun s -> codegen_struct_stub s) sprogram.classes in
915  let _ = List.map (fun s -> codegen_struct s) sprogram.classes in
916  let _ = List.map (fun f -> codegen_funcstub f) sprogram.functions in
917  let _ = List.map (fun f -> codegen_func f) sprogram.functions in
918  let _ = codegen_main sprogram.main in
919  let _ = codegen_vtbl sprogram.classes in
920  let _ = linker Conf.bindings_path in
921  the_module
922
923  (* Need to handle assignment of two different types *)
924  (* Need to handle private/public access *)
```

conf.ml

```
1  let bindings_path = "_includes/bindings.bc"
2  let stdlib_path = "_includes/stdlib.dice"
```

dice.ml

```
1   open Llvm
2   open Llvm_analysis
3   open Analyzer
4   open Utils
5   open Ast
6   open Yojson
7   open Exceptions
8   open Filepath
9
10  type action = Tokens | TokenEndl | PrettyPrint | Ast | Sast | Compile | CompileToFile |
    ↪ Help
11
12  let get_action = function
13  "-tendl"        -> TokenEndl
14  |       "-t"                -> Tokens
15  |       "-p"                -> PrettyPrint
16  |       "-ast"                -> Ast
17  |       "-sast"         -> Sast
18  |       "-h"                -> Help
19  |       "-c"                -> Compile
20  |       "-f"                -> CompileToFile
21  |        _ as s                -> raise (Exceptions.InvalidCompilerArgument s)
22
23  let check_single_argument = function
24  "-h"        -> Help, ""
25  |       "-tendl"
26  |       "-t"
27  |       "-p"
28  |       "-ast"
29  |       "-sast"
30  |       "-c"
31  |       "-f"        -> raise (Exceptions.NoFileArgument)
32  |        _ as s        -> CompileToFile, s
33
34  let dice_name filename =
35  let basename = Filename.basename filename in
36  let filename = Filename.chop_extension basename in
37  filename ^ ".ll"
38
39  let help_string = (
40  "Usage: dice [optional-option] <source file>\n" ^
41  "optional-option:\n" ^
42  "\t-h: Print help text\n" ^
43  "\t-tendl: Prints tokens with newlines intact\n" ^
44  "\t-t: Prints token stream\n" ^
45  "\t-p: Pretty prints Ast as a program\n" ^
46  "\t-ast: Prints abstract syntax tree as json\n" ^
```

```
47  "\t-sast: Prints semantically checked syntax tree as json\n" ^
48  "\t-c: Compiles source\n" ^
49  "\t-f: Compiles source to file (<filename>.<ext> -> <filename>.ll)\n" ^
50  "Option defaults to \"-f\"\n"
51  )
52
53  let _ =
54  ignore(Printexc.record_backtrace true);
55  try
56  let action, filename =
57  if Array.length Sys.argv = 1 then
58  Help, ""
59  else if Array.length Sys.argv = 2 then
60  check_single_argument (Sys.argv.(1))
61  else if Array.length Sys.argv = 3 then
62  get_action Sys.argv.(1), Sys.argv.(2)
63  else raise (Exceptions.InvalidNumberCompilerArguments (Array.length Sys.argv))
64  in
65  (* Added fun () -> <x> so that each is evaluated only when requested *)
66  let filename         = Filepath.realpath filename in
67  let file_in          = fun () -> open_in filename in
68  let lexbuf               = fun () ->        Lexing.from_channel (file_in ()) in
69  let token_list        = fun () -> Processor.build_token_list (lexbuf ()) in
70  let program          = fun () -> Processor.parser filename (token_list ()) in
71  let sprogram          = fun () -> Analyzer.analyze filename (program ()) in
72  let llm                = fun () -> Codegen.codegen_sprogram (sprogram ()) in
73  (* let _ = Llvm_analysis.assert_valid_module llm in *)
74  match action with
75  Help                          -> print_string help_string
76  |      Tokens                                -> print_string (Utils.token_list_to_string
     ↪ (token_list ()))
77  |      TokenEndl                        -> print_string (Utils.token_list_to_string_endl
     ↪ (token_list ()))
78  |      Ast                              -> print_string (pretty_to_string
     ↪ (Utils.print_tree (program ())))
79  |      Sast                             -> print_string (pretty_to_string
     ↪ (Utils.map_sprogram_to_json (sprogram ())))
80  |      PrettyPrint          -> print_string (Utils.string_of_program (program ()))
81  |      Compile                    -> dump_module (llm ())
82  |      CompileToFile          -> print_module (dice_name filename) (llm ())
83  with
84  Exceptions.IllegalCharacter(filename, c, ln) ->
85  print_string
86  (
87  "In \"" ^ filename ^ "\", Illegal Character, '" ^
88  Char.escaped c ^ "', line " ^ string_of_int ln ^ "\n"
89  )
90  |      Exceptions.UnmatchedQuotation(ln)          -> print_endline("Unmatched
     ↪ Quotation, line " ^ string_of_int ln)
```

```
91  |         Exceptions.IllegalToken(tok)                    -> print_endline("Illegal token "
    ↪   ^ tok)
92  |         Exceptions.MissingEOF                                    -> print_endline("Missing
    ↪   EOF")
93  |         Parsing.Parse_error ->
94  print_string
95  (
96  "File \"" ^ !Processor.filename ^ "\", " ^
97  "line " ^ string_of_int !Processor.line_number ^ ", " ^
98  "character " ^ string_of_int !Processor.char_num ^ ", " ^
99  "Syntax Error, token " ^ Utils.string_of_token !Processor.last_token ^ "\n"
100 )
101
102 |          Exceptions.InvalidNumberCompilerArguments i -> print_endline ("Invalid
    ↪   argument passed " ^ (string_of_int i)); print_string help_string
103 |         Exceptions.InvalidCompilerArgument s            -> print_endline ("Invalid
    ↪   argument passed " ^ s); print_string help_string
104 |         Exceptions.NoFileArgument                               ->
    ↪   print_string ("Must include file argument\n" ^ help_string)
105
106 |         Exceptions.IncorrectNumberOfArgumentsException                    ->
    ↪   print_endline("Incorrect number of arguments passed to function")
107 |         Exceptions.ConstructorNotFound(cname)
    ↪                                          -> print_endline("Constructor" ^ cname ^ "
    ↪   not found")
108 |         Exceptions.DuplicateClassName(cname)                                  ->
    ↪   print_endline("Class " ^ cname ^ " not found")
109 |         Exceptions.DuplicateField
    ↪                                                    ->
    ↪   print_endline("Duplicate field defined")
110 |         Exceptions.DuplicateFunction(fname)                               ->
    ↪   print_endline("Duplicate function defined " ^ fname)
111 |         Exceptions.DuplicateConstructor
    ↪                                          -> print_endline("Duplicate
    ↪   constructor found")
112 |         Exceptions.DuplicateLocal(lname)
    ↪                                          -> print_endline("Duplicate local
    ↪   variable defined " ^ lname)
113 |         Exceptions.UndefinedClass(cname)
    ↪                                          -> print_endline("Undefined class " ^
    ↪   cname)
114 |         Exceptions.UnknownIdentifier(id)
    ↪                                          -> print_endline("Unkown identifier "
    ↪   ^ id)
115 |         Exceptions.InvalidBinopExpression(binop)                          ->
    ↪   print_endline("Invalid binary expression " ^ binop)
116 |         Exceptions.InvalidIfStatementType
    ↪                                          -> print_endline("Invalid type passed
    ↪   to if statement, must be bool")
```

```
117  |          Exceptions.InvalidForStatementType
     ↪                                                  -> print_endline("Invalid type passed
     ↪ to for loop, must be bool")
118  |          Exceptions.ReturnTypeMismatch(t1, t2)                              ->
     ↪ print_endline("Incorrect return type " ^ t1 ^ " expected " ^ t2)
119  |          Exceptions.MainNotDefined
     ↪                                                                  ->
     ↪ print_endline("Main not found in program")
120  |
     ↪          Exceptions.MultipleMainsDefined                                            ->
     ↪ print_endline("Multiple mains defined, can only define 1")
121  |          Exceptions.InvalidWhileStatementType                        ->
     ↪ print_endline("Invalid type passed to while loop, must be bool")
122  |          Exceptions.LocalAssignTypeMismatch(t1, t2)                   ->
     ↪ print_endline("Invalid assignment of " ^ t1 ^ " to " ^ t2)
123  |          Exceptions.InvalidUnaryOperation
     ↪                                                  -> print_endline("Invalid unary
     ↪ operator")
124  |          Exceptions.AssignmentTypeMismatch(t1, t2)                        ->
     ↪ print_endline("Invalid assignment of " ^ t1 ^ " to " ^ t2)
125  |          Exceptions.FunctionNotFound(fname, scope)                        ->
     ↪ print_endline("function " ^ fname ^ " not found in scope " ^ scope)
126  |          Exceptions.UndefinedID(id)
     ↪                                                              ->
     ↪ print_endline("Undefined id " ^ id)
127  |          Exceptions.InvalidAccessLHS(t)
     ↪                                                  -> print_endline("Invalid LHS
     ↪ expression of dot operator with " ^ t)
128  |          Exceptions.LHSofRootAccessMustBeIDorFunc(lhs)                   ->
     ↪ print_endline("Dot operator expects ID, not " ^ lhs)
129  |          Exceptions.ObjAccessMustHaveObjectType(t)                        ->
     ↪ print_endline("Can only dereference objects, not " ^ t)
130  |          Exceptions.UnknownIdentifierForClass(c, id)                   ->
     ↪ print_endline("Unknown id " ^ id ^ " for class " ^ c)
131  |          Exceptions.CannotUseReservedFuncName(f)                        ->
     ↪ print_endline("Cannot use name " ^ f ^ " because it is reserved")
132  |          Exceptions.InvalidArrayPrimitiveConsecutiveTypes(t1,t2)        ->
     ↪ print_endline("Array primitive types must be equal, not " ^ t1 ^ " " ^ t2)
133  |          Exceptions.InvalidArrayPrimitiveType(t)                        ->
     ↪ print_endline("Array primitive type invalid, " ^ t)
134  |          Exceptions.MustPassIntegerTypeToArrayCreate                   ->
     ↪ print_endline("Only integer types can be passed to an array initializer")
135  |          Exceptions.ArrayInitTypeInvalid(t)
     ↪                                                  -> print_endline("Only integer types
     ↪ can be passed to an array initializer, not " ^ t)
136  |          Exceptions.MustPassIntegerTypeToArrayAccess                   ->
     ↪ print_endline("Only integer types can be passed to an array access")
137  |          Exceptions.ArrayAccessInvalidParamLength(o,a)                   ->
     ↪ print_endline("Only arrays can have access to length, not " ^ o ^ " " ^ a)
```

```
138  |          Exceptions.ArrayAccessExpressionNotArray(a)                        ->
↪   print_endline("This expression is not an array " ^ a)
139  |          Exceptions.CanOnlyAccessLengthOfArray
↪                                           -> print_endline("Can only access the length
↪   of an array")
140  |          Exceptions.CanOnlyDeleteObjectsOrArrays                           ->
↪   print_endline("Can only delete objects or arrays")
141  |          Exceptions.CannotAccessLengthOfCharArray                          ->
↪   print_endline("Cannot access the length of a char array")
142  |          Exceptions.AllNonVoidFunctionsMustEndWithReturn(f)            ->
↪   print_endline("Non-void function " ^ f ^ " does not end in return")
143  |          Exceptions.CyclicalDependencyBetween(c1, c2)                      ->
↪   print_endline("Class " ^ c1 ^ " and " ^ c2 ^ " have a cylical dependence")
144  |          Exceptions.CannotAccessPrivateFieldInNonProperScope(f, cp, cc) ->
↪   print_endline("Cannot access private field " ^ f ^ " in scope " ^ cp ^ " from object
↪   " ^ cc)
145  |          Exceptions.CannotCallBreakOutsideOfLoop                           ->
↪   print_endline("Cannot call break outside of loop")
146  |          Exceptions.CannotCallContinueOutsideOfLoop                        ->
↪   print_endline("Cannot call continue outside of loop")
147  |          Exceptions.CannotAccessPrivateFunctionInNonProperScope(f, cp, cc) ->
↪   print_endline("Cannot access private function " ^ f ^ " in scope " ^ cp ^ " from
↪   object " ^ cc)
148  |          Exceptions.CannotPassNonInheritedClassesInPlaceOfOthers(c1, c2)        ->
↪   print_endline("Cannot pass non-inherited classe" ^ c1 ^ " to parameter " ^ c2)
149  |          Exceptions.IncorrectTypePassedToFunction(id, t)
↪                                           -> print_endline("Canot pass type " ^ t ^ "
↪   to " ^ id)
150  |          Exceptions.IncorrectNumberOfArguments(f, a1, a2) -> print_endline("Cannot pass
↪   " ^ string_of_int a1 ^ " args when expecting " ^ string_of_int a2 ^ " in " ^f)
151  |          Exceptions.ClassIsNotExtendedBy(c1, c2)                           ->
↪   print_endline("Class " ^ c1 ^ " not extended by " ^ c2)
152
153  |          Exceptions.InvalidTypePassedToPrintf                              ->
↪   print_endline("Invalid type passed to print")
154  |          Exceptions.InvalidBinaryOperator                                 ->
↪   print_endline("Invalid binary operator")
155  |          Exceptions.UnknownVariable(id)
↪                                           -> print_endline("Unknown variable "
↪   ^ id)
156  |          Exceptions.AssignLHSMustBeAssignable                             ->
↪   print_endline("Assignment lhs must be assignable")
157  |          Exceptions.CannotCastTypeException(t1, t2)                        ->
↪   print_endline("Cannot cast " ^ t1 ^ " to " ^ t2)
158  |          Exceptions.InvalidBinopEvaluationType                            ->
↪   print_endline("Invalid binary expression evaluation type")
159  |          Exceptions.FloatOpNotSupported
↪                                           -> print_endline("Float operation not
↪   supported")
```

```
160    |         Exceptions.IntOpNotSupported                              ->
  ↪  print_endline("Integer operation not supported")
161    |         Exceptions.LLVMFunctionNotFound(f)                   ->
  ↪  print_endline("LLVM function " ^ f ^ " not found")
162    |         Exceptions.InvalidStructType(t)                     ->
  ↪  print_endline("Invalid structure type " ^ t)
163    |         Exceptions.UnableToCallFunctionWithoutParent(f)      ->
  ↪  print_endline("Unable to call function " ^ f ^ " without parent")
164    |         Exceptions.CannotAssignParam(p)                     ->
  ↪  print_endline("Cannot assign to param " ^ p)
165    |         Exceptions.InvalidUnopEvaluationType                 ->
  ↪  print_endline("Invalid unary expression evaluation type")
166    |         Exceptions.UnopNotSupported                          ->
  ↪  print_endline("Unary operator not supported")
167    |         Exceptions.ArrayLargerThan1Unsupported               ->
  ↪  print_endline("Array dimensions greater than 1 not supported")
168    |         Exceptions.CanOnlyCompareObjectsWithNull(e1, e2)      -> print_endline("Can
  ↪  only compare objects with null " ^ e1 ^ " " ^ e2)
169    |         Exceptions.ObjOpNotSupported(op)                     ->
  ↪  print_endline("Object operator not supported " ^ op)
170    |         Exceptions.CanOnlyCompareArraysWithNull(e1, e2)      -> print_endline("Can
  ↪  only compare arrays with null " ^ e1 ^ " " ^ e2)
```

exceptions.ml

```
1   (* Dice Exceptions *)
2   exception InvalidNumberCompilerArguments of int
3   exception InvalidCompilerArgument of string
4   exception NoFileArgument
5
6   (* Processor Exceptions *)
7   exception MissingEOF
8
9   (* Scanner Exceptions *)
10  exception IllegalCharacter of string * char * int
11  exception UnmatchedQuotation of int
12  exception IllegalToken of string
13
14  (* Analyzer Exceptions *)
15  exception IncorrectNumberOfArgumentsException
16  exception ConstructorNotFound of string
17  exception DuplicateClassName of string
18  exception DuplicateField
19  exception DuplicateFunction of string
20  exception DuplicateConstructor
21  exception DuplicateLocal of string
22  exception UndefinedClass of string
23  exception UnknownIdentifier of string
24  exception InvalidBinopExpression of string
25  exception InvalidIfStatementType
26  exception InvalidForStatementType
27  exception ReturnTypeMismatch of string * string
28  exception MainNotDefined
29  exception MultipleMainsDefined
30  exception InvalidWhileStatementType
31  exception LocalAssignTypeMismatch of string * string
32  exception InvalidUnaryOperation
33  exception AssignmentTypeMismatch of string * string
34  exception FunctionNotFound of string * string
35  exception UndefinedID of string
36  exception InvalidAccessLHS of string
37  exception LHSofRootAccessMustBeIDorFunc of string
38  exception ObjAccessMustHaveObjectType of string
39  exception UnknownIdentifierForClass of string * string
40  exception CannotUseReservedFuncName of string
41  exception InvalidArrayPrimitiveConsecutiveTypes of string * string
42  exception InvalidArrayPrimitiveType of string
43  exception MustPassIntegerTypeToArrayCreate
44  exception ArrayInitTypeInvalid of string
45  exception MustPassIntegerTypeToArrayAccess
46  exception ArrayAccessInvalidParamLength of string * string
47  exception ArrayAccessExpressionNotArray of string
```

```
48   exception CanOnlyAccessLengthOfArray
49   exception CanOnlyDeleteObjectsOrArrays
50   exception CannotAccessLengthOfCharArray
51   exception AllNonVoidFunctionsMustEndWithReturn of string
52   exception CyclicalDependencyBetween of string * string
53   exception CannotAccessPrivateFieldInNonProperScope of string * string * string
54   exception CannotCallBreakOutsideOfLoop
55   exception CannotCallContinueOutsideOfLoop
56   exception CannotAccessPrivateFunctionInNonProperScope of string * string * string
57   exception CannotPassNonInheritedClassesInPlaceOfOthers of string * string
58   exception IncorrectTypePassedToFunction of string * string
59   exception IncorrectNumberOfArguments of string * int * int
60   exception ClassIsNotExtendedBy of string * string
61
62   (* Codegen Exceptions *)
63   exception InvalidTypePassedToPrintf
64   exception InvalidBinaryOperator
65   exception UnknownVariable of string
66   exception AssignLHSMustBeAssignable
67   exception CannotCastTypeException of string * string
68   exception InvalidBinopEvaluationType
69   exception FloatOpNotSupported
70   exception IntOpNotSupported
71   exception LLVMFunctionNotFound of string
72   exception InvalidStructType of string
73   exception UnableToCallFunctionWithoutParent of string
74   exception CannotAssignParam of string
75   exception InvalidUnopEvaluationType
76   exception UnopNotSupported
77   exception ArrayLargerThan1Unsupported
78   exception CanOnlyCompareObjectsWithNull of string * string
79   exception ObjOpNotSupported of string
80   exception CanOnlyCompareArraysWithNull of string * string
```

filepath.ml

```
1   open Filename
2   open Unix
3
4   exception Safe_exception of (string * string list ref)
5
6   let raise_safe fmt =
7     let do_raise msg = raise @@ Safe_exception (msg, ref []) in
8     Printf.ksprintf do_raise fmt
9
10  let reraise_with_context ex fmt =
11    let do_raise context =
12      let () = match ex with
13        | Safe_exception (_, old_contexts) -> old_contexts := context :: !old_contexts
14        | _ -> Printf.eprintf "warning: Attempt to add note '%s' to non-Safe_exception!" context
15      in
16      raise ex
17    in Printf.ksprintf do_raise fmt
18
19  module StringMap = struct
20    include Map.Make(String)
21    let find_nf = find
22    let find_safe key map = try find key map with Not_found -> raise_safe "BUG: Key '%s' not
   ↪   found in StringMap!" key
23    let find key map = try Some (find key map) with Not_found -> None
24    let map_bindings fn map = fold (fun key value acc -> fn key value :: acc) map []
25  end
26
27  type path_component =
28    | Filename of string   (* foo/ *)
29    | ParentDir            (* ../ *)
30    | CurrentDir           (* ./ *)
31    | EmptyComponent       (* / *)
32
33  type filepath = string
34
35
36  let on_windows = Filename.dir_sep <> "/"
37
38  let path_is_absolute path = not (Filename.is_relative path)
39
40  let string_tail s i =
41    let len = String.length s in
42    if i > len then failwith ("String '" ^ s ^ "' too short to split at " ^ (string_of_int
   ↪   i))
43    else String.sub s i (len - i)
44
45  let split_path_str path =
```

```
46  let l = String.length path in
47  let is_sep c = (c = '/' || (on_windows && c = '\\')) in
48
49  (* Skip any leading slashes and return the rest *)
50  let rec find_rest i =
51  if i < l then (
52  if is_sep path.[i] then find_rest (i + 1)
53  else string_tail path i
54  ) else (
55  ""
56  ) in
57
58  let rec find_slash i =
59  if i < l then (
60  if is_sep path.[i] then (String.sub path 0 i, find_rest (i + 1))
61  else find_slash (i + 1)
62  ) else (
63  (path, "")
64  )
65  in
66  find_slash 0
67
68  let split_first path =
69  if path = "" then
70  (CurrentDir, "")
71  else (
72  let (first, rest) = split_path_str path in
73  let parsed =
74  if first = Filename.parent_dir_name then ParentDir
75  else if first = Filename.current_dir_name then CurrentDir
76  else if first = "" then EmptyComponent
77  else Filename first in
78  (parsed, rest)
79  )
80
81  let normpath path : filepath =
82  let rec explode path =
83  match split_first path with
84  | CurrentDir, "" -> []
85  | CurrentDir, rest -> explode rest
86  | first, "" -> [first]
87  | first, rest -> first :: explode rest in
88
89  let rec remove_parents = function
90  | checked, [] -> checked
91  | (Filename _name :: checked), (ParentDir :: rest) -> remove_parents (checked, rest)
92  | checked, (first :: rest) -> remove_parents ((first :: checked), rest) in
93
94  let to_string = function
```

```
95   | Filename name -> name
96   | ParentDir -> Filename.parent_dir_name
97   | EmptyComponent -> ""
98   | CurrentDir -> assert false in
99   String.concat Filename.dir_sep @@ List.rev_map to_string @@ remove_parents ([], explode
     ↪  path)
100
101
102  let abspath path =
103  let (+/) = Filename.concat in
104  normpath (
105  if path_is_absolute path then path
106  else (Sys.getcwd ()) +/ path
107  )
108
109  let realpath path =
110  let (+/) = Filename.concat in    (* Faster version, since we know the path is relative *)
111
112  (* Based on Python's version *)
113  let rec join_realpath path rest seen =
114  (* Printf.printf "join_realpath <%s> + <%s>\n" path rest; *)
115  (* [path] is already a realpath (no symlinks). [rest] is the bit to join to it. *)
116  match split_first rest with
117  | Filename name, rest -> (
118  (* path + name/rest *)
119  let newpath = path +/ name in
120  let link = try Some (Unix.readlink newpath) with Unix.Unix_error _ -> None in
121  match link with
122  | Some target ->
123  (* path + symlink/rest *)
124  begin match StringMap.find newpath seen with
125  | Some (Some cached_path) -> join_realpath cached_path rest seen
126  | Some None -> (normpath (newpath +/ rest), false)    (* Loop; give up *)
127  | None ->
128  (* path + symlink/rest -> realpath(path + target) + rest *)
129  match join_realpath path target (StringMap.add newpath None seen) with
130  | path, false ->
131  (normpath (path +/ rest), false)   (* Loop; give up *)
132  | path, true -> join_realpath path rest (StringMap.add newpath (Some path) seen)
133  end
134  | None ->
135  (* path + name/rest -> path/name + rest (name is not a symlink) *)
136  join_realpath newpath rest seen
137  )
138  | CurrentDir, "" ->
139  (path, true)
140  | CurrentDir, rest ->
141  (* path + ./rest *)
142  join_realpath path rest seen
```

```
143  | ParentDir, rest ->
144  (* path + ../rest *)
145  if String.length path > 0 then (
146  let name = Filename.basename path in
147  let path = Filename.dirname path in
148  if name = Filename.parent_dir_name then
149  join_realpath (path +/ name +/ name) rest seen     (* path/.. +  ../rest -> path/../.. +
     ↪  rest *)
150  else
151  join_realpath path rest seen                        (* path/name + ../rest -> path + rest
     ↪  *)
152  ) else (
153  join_realpath Filename.parent_dir_name rest seen    (* "" + ../rest -> .. + rest *)
154  )
155  | EmptyComponent, rest ->
156  (* [rest] is absolute; discard [path] and start again *)
157  join_realpath Filename.dir_sep rest seen
158  in
159
160  try
161  if on_windows then
162  abspath path
163  else (
164  fst @@ join_realpath (Sys.getcwd ()) path StringMap.empty
165  )
166  with Safe_exception _ as ex -> reraise_with_context ex "... in realpath(%s)" path
```

Makefile

```
1   TARGET=src/dice
2   LIBS=-I,/usr/lib/ocaml/
3   FLAGS= -j 0 -r -use-ocamlfind -pkgs
    ↪  yojson,llvm,llvm.analysis,llvm.bitwriter,llvm.bitreader,llvm.linker,llvm.target,batteries
4   OCAMLBUILD=ocamlbuild
5   OPAM=opam config env
6   CLIBEXT=_includes
7
8
9   all: native
10          @clang-3.7 -c -emit-llvm src/bindings.c
11          @mkdir -p $(CLIBEXT)
12          @mv bindings.bc $(CLIBEXT)/bindings.bc
13          @cp src/stdlib.dice $(CLIBEXT)/stdlib.dice
14          @mv dice.native dice
15          @echo Compilation Complete
16
17  clean:
18          @cd src
19          $(OCAMLBUILD) -clean
20          @cd ..
21          @rm -rf $(CLIBEXT)
22          @echo cleaning complete
23
24  native:
25          @cd src
26          @eval `opam config env`
27          $(OCAMLBUILD) $(FLAGS) $(TARGET).native
28          @cd ..
29
30  byte:
31          $(OCAMLBUILD) $(FLAGS) $(TARGET).byte
32
33  depend:
34          echo "Not needed."
```

parser.mly

```
1    %{  open Ast  %}
2
3    %token CLASS EXTENDS CONSTRUCTOR INCLUDE DOT THIS PRIVATE PUBLIC
4    %token INT FLOAT BOOL CHAR VOID NULL TRUE FALSE
5    %token SEMI LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET COMMA
6    %token AND NOT OR PLUS MINUS TIMES DIVIDE ASSIGN MODULO
7    %token EQ NEQ LT LEQ GT GEQ BAR
8    %token RETURN IF ELSE FOR WHILE BREAK CONTINUE NEW DELETE
9    %token <int> INT_LITERAL
10   %token <float> FLOAT_LITERAL
11   %token <string> STRING_LITERAL
12   %token <string> ID
13   %token <char> CHAR_LITERAL
14   %token EOF
15
16   %nonassoc NOELSE
17   %nonassoc ELSE
18   %right ASSIGN
19   %left AND OR
20   %left EQ NEQ
21   %left LT GT LEQ GEQ
22   %left PLUS MINUS
23   %left TIMES DIVIDE MODULO
24   %right NOT
25   %right DELETE
26   %right RBRACKET
27   %left LBRACKET
28   %right DOT
29
30   %start program
31   %type <Ast.program> program
32
33   %%
34
35   program:
36   includes cdecls EOF { Program($1, $2) }
37
38   /*****************
39   INCLUDE
40   *****************/
41
42   includes:
43   /* nothing */ { [] }
44   |         include_list  { List.rev $1 }
45
46   include_list:
47   include_decl            { [$1] }
```

```
48    |          include_list include_decl { $2::$1 }

49

50    include_decl:
51    INCLUDE LPAREN STRING_LITERAL RPAREN SEMI { Include($3) }

52

53

54    /******************
55    CLASSES
56    ******************/
57    cdecls:
58    cdecl_list     { List.rev $1 }

59

60    cdecl_list:
61    cdecl              { [$1] }
62    | cdecl_list cdecl  { $2::$1 }

63

64    cdecl:
65    CLASS ID LBRACE cbody RBRACE { {
66                    cname = $2;
67                    extends = NoParent;
68                    cbody = $4
69          } }
70    |       CLASS ID EXTENDS ID LBRACE cbody RBRACE { {
71                    cname = $2;
72                    extends = Parent($4);
73                    cbody = $6
74    } }

75

76    cbody:
77    /* nothing */ { {
78                    fields = [];
79                    constructors = [];
80                    methods = [];
81    } }
82    |        cbody field { {
83                    fields = $2 :: $1.fields;
84                    constructors = $1.constructors;
85                    methods = $1.methods;
86    } }
87    |        cbody constructor { {
88                    fields = $1.fields;
89                    constructors = $2 :: $1.constructors;
90                    methods = $1.methods;
91    } }
92    |        cbody fdecl { {
93          fields = $1.fields;
94          constructors = $1.constructors;
95          methods = $2 :: $1.methods;
96    } }
```

```
97
98
99    /*****************
100   CONSTRUCTORS
101   *****************/
102
103   constructor:
104   CONSTRUCTOR LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE {
105           {
106                   scope = Public;
107                   fname = Constructor;
108                   returnType = Datatype(ConstructorType);
109                   formals = $3;
110                   body = List.rev $6;
111                   overrides = false;
112                   root_cname = None;
113           }
114   }
115
116   /*****************
117   FIELDS
118   *****************/
119
120   scope:
121   PRIVATE { Private }
122   |       PUBLIC  { Public }
123
124   /* public UserObj name; */
125   field:
126   scope datatype ID SEMI { Field($1, $2, $3) }
127
128   /*****************
129   METHODS
130   *****************/
131
132   fname:
133   ID { $1 }
134
135   fdecl:
136   scope datatype fname LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
137   {
138           {
139                   scope = $1;
140                   fname = FName($3);
141                   returnType = $2;
142                   formals = $5;
143                   body = List.rev $8;
144                   overrides = false;
145                   root_cname = None;
```

```
146                }
147    }
148
149    /******************
150    FORMALS/PARAMETERS & VARIABLES & ACTUALS
151    ******************/
152
153    formals_opt:
154    /* nothing */ { [] }
155    |          formal_list   { List.rev $1 }
156
157    formal_list:
158    formal                   { [$1] }
159    |          formal_list COMMA formal { $3 :: $1 }
160
161    formal:
162    datatype ID { Formal($1, $2) }
163
164    actuals_opt:
165    /* nothing */ { [] }
166    |          actuals_list  { List.rev $1 }
167
168    actuals_list:
169    expr                     { [$1] }
170    |          actuals_list COMMA expr { $3 :: $1 }
171
172
173    /***************
174    DATATYPES
175    ***************/
176    primitive:
177    INT                { Int_t }
178    |         FLOAT              { Float_t }
179    |         CHAR               { Char_t }
180    |         BOOL               { Bool_t }
181    |         VOID           { Void_t }
182
183    name:
184    CLASS ID { Objecttype($2) }
185
186    type_tag:
187    primitive { $1 }
188    |         name           { $1 }
189
190    array_type:
191    type_tag LBRACKET brackets RBRACKET { Arraytype($1, $3) }
192
193    datatype:
194    type_tag   { Datatype($1) }
```

```
195  |          array_type { $1 }

196

197  brackets:
198  /* nothing */                           { 1 }
199  |          brackets RBRACKET LBRACKET { $1 + 1 }

200

201  /*****************
202  EXPRESSIONS
203  *****************/

204

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

208

209  stmt:
210  expr SEMI { Expr($1) }
211  |          RETURN expr SEMI { Return($2) }
212  |        RETURN SEMI                 { Return(Noexpr) }
213  |          LBRACE stmt_list RBRACE { Block(List.rev $2) }
214  |          IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([Expr(Noexpr)])) }
215  |          IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
216  |          FOR LPAREN expr_opt SEMI expr_opt SEMI expr_opt RPAREN stmt
217  { For($3, $5, $7, $9) }
218  |          WHILE LPAREN expr RPAREN stmt        { While($3, $5) }
219  |          BREAK SEMI                                      { Break }
220  |          CONTINUE SEMI                               { Continue }
221  |    datatype ID SEMI                           { Local($1, $2, Noexpr) }
222  |          datatype ID ASSIGN expr SEMI         { Local($1, $2, $4) }

223

224  expr_opt:
225  /* nothing */ { Noexpr }
226  |          expr        { $1 }

227

228  expr:
229  literals                                                  { $1 }
230  |          expr PLUS   expr                                { Binop($1, Add,   $3)
     ↪  }
231  |          expr MINUS  expr                                { Binop($1, Sub,   $3)
     ↪  }
232  |          expr TIMES  expr                                { Binop($1, Mult,  $3)
     ↪  }
233  |          expr DIVIDE expr                                { Binop($1, Div,   $3)
     ↪  }
234  |          expr EQ     expr                                { Binop($1, Equal, $3)
     ↪  }
235  |          expr NEQ    expr                                { Binop($1, Neq,   $3)
     ↪  }
236  |          expr LT     expr                                { Binop($1, Less,  $3)
     ↪  }
```

```
237  |         expr LEQ    expr                                          { Binop($1, Leq,    $3)
     ↪  }
238  |         expr GT     expr                                          { Binop($1, Greater,
     ↪  $3) }
239  |         expr GEQ    expr                                          { Binop($1, Geq,    $3)
     ↪  }
240  |         expr AND    expr                                          { Binop($1, And,    $3)
     ↪  }
241  |         expr MODULO expr                                          { Binop($1, Mod,
     ↪  $3)}
242  |         NOT  expr                                                      { Unop (Not,
     ↪  $2) }
243  |         expr OR     expr                                          { Binop($1, Or,     $3)
     ↪  }
244  |         expr DOT    expr                                          { ObjAccess($1, $3) }
245  |         expr ASSIGN expr                                          { Assign($1, $3) }
246  |         DELETE expr                                                  { Delete($2) }
247  |     MINUS expr                                                        { Unop (Sub, $2) }
248  |         ID LPAREN actuals_opt RPAREN                  { Call($1, $3) }
249  |         NEW ID LPAREN actuals_opt RPAREN          { ObjectCreate($2, $4) }
250  |         NEW type_tag bracket_args RBRACKET        { ArrayCreate(Datatype($2), List.rev
     ↪  $3) }
251  |         expr bracket_args RBRACKET                        { ArrayAccess($1, List.rev
     ↪  $2) }
252  |         LPAREN expr RPAREN                                     { $2 }
253
254  bracket_args:
255  LBRACKET expr                                             { [$2] }
256  |         bracket_args RBRACKET LBRACKET expr { $4 :: $1 }
257
258  literals:
259  INT_LITERAL                      { Int_Lit($1) }
260  | FLOAT_LITERAL                  { Float_Lit($1) }
261  | TRUE                                       { Boolean_Lit(true) }
262  | FALSE                                      { Boolean_Lit(false) }
263  | STRING_LITERAL                 { String_Lit($1) }
264  | CHAR_LITERAL                    { Char_Lit($1) }
265  | THIS                               { This }
266  | ID                                   { Id($1) }
267  | NULL                           { Null }
268  | BAR array_prim BAR        { ArrayPrimitive($2) }
269
270  /* ARRAY LITERALS */
271
272  array_prim:
273  expr                                 { [$1] }
274  |         array_prim COMMA expr        { $3 :: $1 }
```

processor.ml

```ocaml
1  open Parser
2
3  type token_attr = {
4          lineno: int;
5          cnum: int;
6  }
7
8  let line_number = ref 1
9  let last_token = ref EOF
10 let char_num = ref 1
11 let filename = ref ""
12
13 let build_token_list lexbuf =
14 Scanner.filename := !filename;
15 let rec helper prev_cnum prev_lineno lexbuf token_list =
16 let token = Scanner.token lexbuf in
17 let lineno = !Scanner.lineno in
18 let cnum = (Lexing.lexeme_start_p lexbuf).Lexing.pos_cnum in
19 let prev_cnum = if lineno > prev_lineno then cnum else prev_cnum in
20 let cnum = cnum - prev_cnum in
21 match token with
22 EOF as eof -> (eof, { lineno = lineno; cnum = cnum } )::token_list
23 |   t           -> (t, { lineno = lineno; cnum = cnum } )::(helper prev_cnum lineno lexbuf
   ↪  token_list)
24 in helper 0 0 lexbuf []
25
26 let parser filen token_list =
27 let token_list = ref(token_list) in
28 let tokenizer _ =
29 match !token_list with
30 | (head, curr) :: tail ->
31 filename := filen;
32 line_number := curr.lineno;
33 char_num     := curr.cnum;
34 last_token := head;
35 token_list := tail;
36 head
37 | [] -> raise (Exceptions.MissingEOF)
38 in
39 let program = Parser.program tokenizer (Lexing.from_string "") in
40 program
```

sast.ml

```
1   open Ast
2
3   type sexpr =
4   SInt_Lit of int
5   |          SBoolean_Lit of bool
6   |          SFloat_Lit of float
7   |          SString_Lit of string
8   |          SChar_Lit of char
9   |          SId of string * datatype
10  |          SBinop of sexpr * op * sexpr * datatype
11  |          SAssign of sexpr * sexpr * datatype
12  |          SNoexpr
13  |          SArrayCreate of datatype * sexpr list * datatype
14  |          SArrayAccess of sexpr * sexpr list * datatype
15  |          SObjAccess of sexpr * sexpr * datatype
16  |          SCall of string * sexpr list * datatype * int
17  |     SObjectCreate of string * sexpr list * datatype
18  |          SArrayPrimitive of sexpr list * datatype
19  |           SUnop of op * sexpr * datatype
20  |          SNull
21  |          SDelete of sexpr
22
23  type sstmt =
24  SBlock of sstmt list
25  |          SExpr of sexpr * datatype
26  |          SReturn of sexpr  * datatype
27  |          SIf of sexpr * sstmt * sstmt
28  |          SFor of sexpr * sexpr * sexpr * sstmt
29  |          SWhile of sexpr * sstmt
30  |           SBreak
31  |     SContinue
32  |     SLocal of datatype * string * sexpr
33
34  type func_type = User | Reserved
35
36  type sfunc_decl = {
37          sfname : fname;
38          sreturnType : datatype;
39          sformals : formal list;
40          sbody : sstmt list;
41          func_type : func_type;
42          source : string;
43          overrides : bool;
44  }
45
46  type sclass_decl = {
47          scname : string;
```

```
48          sfields : field list;
49          sfuncs: sfunc_decl list;
50  }
51
52  (* Class Declarations | All method declarations | Main entry method *)
53  type sprogram =  {
54          classes : sclass_decl list;
55          functions : sfunc_decl list;
56          main : sfunc_decl;
57          reserved : sfunc_decl list;
58  }
```

scanner.mll

```
1   {
2           open Parser
3           let lineno = ref 1
4           let depth = ref 0
5           let filename = ref ""
6
7           let unescape s =
8           Scanf.sscanf ("\"" ^ s ^ "\"") "%S%!" (fun x -> x)
9   }
10
11  let alpha = ['a'-'z' 'A'-'Z']
12  let escape = '\\' ['\\' ''' '"' 'n' 'r' 't']
13  let escape_char = ''' (escape) '''
14  let ascii = ([' '-'!' '#'-'[' ']'-'~'])
15  let digit = ['0'-'9']
16  let id = alpha (alpha | digit | '_')*
17  let string = '"' ( (ascii | escape)* as s ) '"'
18  let char = ''' ( ascii | digit ) '''
19  let float = (digit+) ['.'] digit+
20  let int = digit+
21  let whitespace = [' ' '\t' '\r']
22  let return = '\n'
23
24  rule token = parse
25  whitespace { token lexbuf }
26  | return          { incr lineno; token lexbuf}
27  | "(*"       { incr depth; comment lexbuf }
28
29  | '('       { LPAREN }
30  | ')'       { RPAREN }
31  | '{'       { LBRACE }
32        | '}'       { RBRACE }
33  | ';'       { SEMI }
34  | ','       { COMMA }
35
36  (* Operators *)
37  | '+'       { PLUS }
38  | '-'       { MINUS }
39  | '*'       { TIMES }
40  | '/'       { DIVIDE }
41  | '%'       { MODULO }
42  | '='       { ASSIGN }
43  | "=="      { EQ }
44  | "!="      { NEQ }
45  | '<'       { LT }
46  | "<="      { LEQ }
47  | ">"       { GT }
```

```
48  | ">="     { GEQ }
49  | "and"    { AND }
50  | "or"     { OR }
51  | "not"    { NOT }
52  | '.'      { DOT }
53  | '['      { LBRACKET }
54  | ']'      { RBRACKET }
55  | '|'         { BAR }
56
57  (* Branch Control *)
58  | "if"     { IF }
59  | "else"   { ELSE }
60  | "for"    { FOR }
61  | "while"  { WHILE }
62  | "return" { RETURN }
63
64  (* Data Types *)
65  | "int"    { INT }
66  | "float"  { FLOAT }
67  | "bool"   { BOOL }
68  | "char"   { CHAR }
69  | "void"   { VOID }
70  | "null"   { NULL }
71  | "true"   { TRUE }
72  | "false"  { FALSE }
73
74  (* Classes *)
75  | "class"       { CLASS }
76  | "constructor" { CONSTRUCTOR }
77  | "public"      { PUBLIC }
78  | "private"     { PRIVATE }
79  | "extends"     { EXTENDS }
80  | "include"     { INCLUDE }
81  | "this"        { THIS }
82  | "break"             { BREAK }
83  | "continue"        { CONTINUE }
84  | "new"              { NEW }
85  | "delete"             { DELETE }
86
87  | int as lxm              { INT_LITERAL(int_of_string lxm) }
88  | float as lxm           { FLOAT_LITERAL(float_of_string lxm) }
89  | char as lxm            { CHAR_LITERAL( String.get lxm 1 ) }
90  | escape_char as lxm{ CHAR_LITERAL( String.get (unescape lxm) 1) }
91  | string                 { STRING_LITERAL(unescape s) }
92  | id as lxm              { ID(lxm) }
93  | eof                    { EOF }
94
95  | '"'                    { raise (Exceptions.UnmatchedQuotation(!lineno)) }
96  | _ as illegal  { raise (Exceptions.IllegalCharacter(!filename, illegal, !lineno)) }
```

```
97
98  and comment = parse
99  return         { incr lineno; comment lexbuf }
100 |        "*)"          { decr depth; if !depth > 0 then comment lexbuf else token lexbuf }
101 |   "(*"          { incr depth; comment lexbuf }
102 |        _             { comment lexbuf }
```

stdlibe.dice

```
1   class Integer {
2
3           private int my_int;
4
5           constructor(int input) {
6                   this.my_int = input;
7           }
8
9           public int num() {
10                  return this.my_int;
11          }
12
13
14          public char toChar(int digit) {
15
16                  if (digit == 0) {
17                          return '0';
18                  } else if (digit == 1) {
19                          return '1';
20                  } else if (digit == 2) {
21                          return '2';
22                  } else if (digit == 3) {
23                  return '3';
24                  } else if (digit == 4) {
25                  return '4';
26                  } else if (digit == 5) {
27                  return '5';
28                  } else if (digit == 6) {
29                  return '6';
30                  } else if (digit == 7) {
31                  return '7';
32                  } else if (digit == 8) {
33                  return '8';
34                  } else if (digit == 9) {
35                  return '9';
36                  }
37
38          return 'z';
39  }
40
41
42
43
44
45  public class String toString() {
46
47          (* integer cannot be greater than 10 digits in 32 bit *)
```

```
48          int temp = this.my_int;
49          int i = 0;
50          char[] str = new char[9];
51
52          int digit = temp % 10;
53          str[i] = this.toChar(digit);
54          i = i + 1;
55          temp = temp / 10;
56          while (temp > 0) {
57
58                  digit = temp % 10;
59                  str[i] = this.toChar(digit);
60                  temp = temp / 10;
61                  i = i + 1;
62          }
63
64          str[i] = 0;
65          class String newString = new String(str);
66          class String a = newString.reverse();
67          return newString.reverse();
68  }
69  }
70
71
72
73  class String {
74
75          private char[] my_string;
76          private int length;
77
78          constructor(char[] input) {
79
80                  this.my_string = this.copy_internal(input);
81
82                  this.length = this.length();
83          }
84
85          (* PRIVATE CLASSES ----------------------------------------   *)
86
87          private int length_internal(char[] input) {
88                  int length = 0;
89
90                  while(input[length] != 0) {
91                          length = length + 1;
92                  }
93
94                  return length;
95          }
96
```

```
97          private char[] copy_internal(char[] input) {

98

99                  char[] newString = new char[this.length_internal(input) + 1];

100

101                 int i = 0;
102                 for (; input[i] != 0; i = i + 1) {
103                         newString[i] = input[i];
104                 }

105

106                 newString[i] = 0;
107                 return newString;
108         }

109

110         (* PUBLIC CLASSES ----------------------------------------   *)

111

112         public char[] string() {
113                 return this.my_string;
114         }

115

116         public char getChar(int index) {

117

118                 return this.my_string[index];
119         }

120

121         public int length() {

122

123                 int length = 0;

124

125                 while(this.my_string[length] != 0){
126                         length = length + 1;
127                 }

128

129                 return length;
130         }

131

132         public int toInteger() {

133

134                 char[] temp = this.string();
135                 int ndigit = 0;
136                 int i;
137                 int j;
138                 for (i = 0; i < this.length; i = i + 1) {

139

140                         int exp = 1;
141                         int xdigit = this.toDigit(temp[i]);
142                         for (j = 0; j < (this.length-i-1); j = j + 1) {
143                                 exp = exp * 10;
144                         }
145                         xdigit = xdigit * exp;
```

```
146                         ndigit = ndigit + xdigit;
147                     }
148
149                     return ndigit;
150             }
151
152         public int toDigit(char digit) {
153
154                     if (digit == '0') {
155                             return 0;
156                     } else if (digit == '1') {
157                     return 1;
158             } else if (digit == '2') {
159             return 2;
160   } else if (digit == '3') {
161   return 3;
162   } else if (digit == '4') {
163   return 4;
164   } else if (digit == '5') {
165   return 5;
166   } else if (digit == '6') {
167   return 6;
168   } else if (digit == '7') {
169   return 7;
170   } else if (digit == '8') {
171   return 8;
172   } else if (digit == '9') {
173   return 9;
174   }
175
176   return -1;
177   }
178
179
180   public class String copy(class String input) {
181
182           char[] newArray = this.copy_internal(input.string());
183           class String newString = new String(newArray);
184           return newString;
185   }
186
187   public int indexOf(char x) {
188
189           int i = 0;
190           for (; this.getChar(i) != x and this.getChar(i) != 0; i = i + 1) {
191           }
192
193           (* If the char was not found, return -1 *)
194           if (i == this.length()) {
```

```
195                    return -1;
196            }
197
198            return i;
199    }
200
201    public class String reverse() {
202
203            class String newString;
204
205            char[] temp = new char[this.length + 1];
206            int i = this.length;
207            for (; i > 0; i = i - 1) {
208
209                    temp[this.length - i] = this.getChar(i-1);
210            }
211            temp[this.length] = 0;
212            newString = new String(temp);
213            return newString;
214    }
215
216    public class String concat(class String temp) {
217
218            char[] temparray = new char[this.length() + temp.length() + 1];
219
220            (* Copy over the current string into a new char array *)
221            int i = 0;
222            for (; this.getChar(i) != 0; i = i + 1) {
223                    temparray[i] = this.getChar(i);
224            }
225
226            (* Append the new string *)
227            int j = 0;
228            for (; temp.getChar(j) != 0; j = j + 1) {
229                    temparray[i+j] = temp.getChar(j);
230            }
231
232            temparray[this.length() + temp.length()] = 0;
233            class String newString = new String(temparray);
234            return newString;
235    }
236
237    public bool compare(class String check) {
238
239            if (check.length != this.length) {
240                    return false;
241            }
242
243            int i = 0;
```

```
244
245          for (; i < check.length(); i = i + 1) {
246
247                  if (check.getChar(i) != this.getChar(i)) {
248                          return false;
249                  }
250          }
251
252          return true;
253  }
254
255  public bool contains(class String check) {
256
257
258          if (this.length < check.length) {
259                  return false;
260          } else if (this.compare(check)) {
261          return true;
262  } else {
263
264  int diff = this.length - check.length + 1;
265  int i;
266  int j;
267  for ( i = 0; i < diff; i = i + 1)
268
269  for ( j = 0; j < check.length; j = j + 1) {
270
271          if (this.getChar(i+j) != check.getChar(j)) {
272                  break;
273          }
274
275          if (j == check.length - 1) {
276                  return true;
277          }
278  }
279  }
280  return false;
281  }
282
283  public void free() {
284
285          delete(this.my_string);
286  }
287
288  }
289
290
291
292  class File {
```

```
293
294        private class String filePath;
295        private bool isWriteEnabled;
296        private int fd;
297
298        constructor(char[] path, bool isWriteEnabled) {
299
300                this.filePath = new String(path);
301                this.isWriteEnabled = isWriteEnabled;
302                class String a = this.filePath;
303                this.fd = this.openfile(a, this.isWriteEnabled);
304                if (this.fd < 0) {
305                        print("open failed");
306                        exit(1);
307                }
308        }
309
310        (* PRIVATE CLASSES --------------------------------------   *)
311
312        private int openfile(class String path, bool isWriteEnabled) {
313
314                if (isWriteEnabled) {
315                        (* 2 is the value for O_RDWR *)
316                        return open(path.string(), 2);
317                }
318
319                (* 0 is the value for O_RDONLY *)
320                return open(path.string(), 0);
321        }
322
323        (* PUBLIC CLASSES ---------------------------------------   *)
324
325        public void closefile() {
326
327                if (close(this.fd) < 0) {
328                        print("close failed");
329                }
330        }
331
332        public char[] readfile(int bytes) {
333
334                char[] buf = new char[bytes];
335
336                int ret = read(this.fd, buf, bytes);
337
338                if (ret < 0) {
339                        print("read failed");
340                }
341
```

```
342             return buf;
343         }
344
345         public int writefile(char[] buf, int offset) {
346
347             class String temp = new String(buf);
348             int err;
349             (* seek to desired offset from beginning of file *)
350             if (offset > 0) {
351                     err = lseek(this.fd, offset, 0);
352             } else if (offset == -1) {
353             err = lseek(this.fd, 0, 0);
354         } else {
355         (* Seek to the end of the file by default *)
356         err = lseek(this.fd, 0, 2);
357 }
358
359 if (err < 0) {
360         print("seek failed");
361 }
362
363 err = write(this.fd, temp.string(), temp.length());
364 if (err < 0) {
365         print("write failed");
366 }
367 return err;
368 }
369
370 }
```

utils.ml

```ocaml
1   (* Pretty Printer *)
2   open Ast
3   open Sast
4   open Parser
5   open Processor
6   open Yojson
7
8   let save file string =
9   let channel = open_out file in
10  output_string channel string;
11  close_out channel
12
13  let replace input output =
14  Str.global_replace (Str.regexp_string input) output
15
16  (* Print data types *)
17
18  let string_of_scope = function
19  Public        -> "public"
20  |       Private -> "private"
21
22  let string_of_primitive = function
23  Int_t                                        -> "int"
24  |       Float_t                              -> "float"
25  |       Void_t                               -> "void"
26  |       Bool_t                               -> "bool"
27  |       Char_t                               -> "char"
28  |       Objecttype(s)                        -> "class " ^ s
29  |       ConstructorType                      -> "constructor"
30  |        Null_t                              -> "null"
31
32  let string_of_object = function
33  Datatype(Objecttype(s))      -> s
34  |         _ -> ""
35
36  let rec print_brackets = function
37  1 -> "[]"
38  |       a -> "[]" ^ print_brackets (a - 1)
39
40  let string_of_datatype = function
41  Arraytype(p, i)       -> (string_of_primitive p) ^ (print_brackets i)
42  |       Datatype(p)                  -> (string_of_primitive p)
43  |        Any                         -> "Any"
44
45  (* Print expressions *)
46
47  let string_of_op = function
```

```
48   Add                          -> "+"
49   |        Sub                          -> "-"
50   |        Mult                  -> "*"
51   |        Div                          -> "/"
52   |        Equal               -> "=="
53   |        Neq                          -> "!="
54   |        Less                  -> "<"
55   |        Leq                          -> "<="
56   |        Greater                -> ">"
57   |        Geq                          -> ">="
58   |        And                          -> "and"
59   |        Not                          -> "not"
60   |        Or                           -> "or"
61   |        Mod                   -> "%"
62
63   let rec string_of_bracket_expr = function
64   []                          -> ""
65   |        head :: tail        -> "[" ^ (string_of_expr head) ^ "]" ^
     ↪ (string_of_bracket_expr tail)
66   and string_of_array_primitive = function
67   []                                 -> ""
68   |   [last]                        -> (string_of_expr last)
69   |        head :: tail        -> (string_of_expr head) ^ ", " ^
     ↪ (string_of_array_primitive tail)
70   and string_of_expr = function
71   Int_Lit(i)                            -> string_of_int i
72   |        Boolean_Lit(b)               -> if b then "true" else "false"
73   |        Float_Lit(f)                 -> string_of_float f
74   |        String_Lit(s)                -> "\"" ^ (String.escaped s) ^ "\""
75   |        Char_Lit(c)                    -> Char.escaped c
76   |        This                          -> "this"
77   |        Id(s)                          -> s
78   |        Binop(e1, o, e2)             -> (string_of_expr e1) ^ " " ^ (string_of_op o)
     ↪ ^ " " ^ (string_of_expr e2)
79   |        Assign(e1, e2)               -> (string_of_expr e1) ^ " = " ^
     ↪ (string_of_expr e2)
80   |        Noexpr                        -> ""
81   |        ObjAccess(e1, e2)         -> (string_of_expr e1) ^ "." ^ (string_of_expr
     ↪ e2)
82   |        Call(f, el)                  -> f ^ "(" ^ String.concat ", "
     ↪ (List.map string_of_expr el) ^ ")"
83   |        ArrayPrimitive(el)         -> "|" ^ (string_of_array_primitive el) ^ "|"
84   |         Unop(op, e)                 -> (string_of_op op) ^ "(" ^
     ↪ string_of_expr e ^ ")"
85   |        Null                          -> "null"
86   |   ArrayCreate(d, el)         -> "new " ^ string_of_datatype d ^ string_of_bracket_expr
     ↪ el
87   |   ArrayAccess(e, el)         -> (string_of_expr e) ^ (string_of_bracket_expr el)
```

```
 88    |    ObjectCreate(s, el)          -> "new " ^ s ^ "(" ^ String.concat ", " (List.map
    ↪  string_of_expr el) ^ ")"
 89    |        Delete(e)                              -> "delete (" ^ (string_of_expr e) ^
    ↪  ")"
 90    ;;
 91
 92    let rec string_of_bracket_sexpr = function
 93    []                              -> ""
 94    |        head :: tail        -> "[" ^ (string_of_sexpr head) ^ "]" ^
    ↪  (string_of_bracket_sexpr tail)
 95    and string_of_sarray_primitive = function
 96    []                              -> ""
 97    |    [last]                      -> (string_of_sexpr last)
 98    |        head :: tail        -> (string_of_sexpr head) ^ ", " ^
    ↪  (string_of_sarray_primitive tail)
 99    and string_of_sexpr = function
100    SInt_Lit(i)                            -> string_of_int i
101    |        SBoolean_Lit(b)                  -> if b then "true" else "false"
102    |        SFloat_Lit(f)                    -> string_of_float f
103    |        SString_Lit(s)                   -> "\"" ^ (String.escaped s) ^
    ↪  "\""
104    |        SChar_Lit(c)                      -> Char.escaped c
105    |        SId(s, _)                          -> s
106    |        SBinop(e1, o, e2, _)            -> (string_of_sexpr e1) ^ " " ^
    ↪  (string_of_op o) ^ " " ^ (string_of_sexpr e2)
107    |        SAssign(e1, e2, _)                  -> (string_of_sexpr e1) ^ " = " ^
    ↪  (string_of_sexpr e2)
108    |        SNoexpr                                -> ""
109    |        SObjAccess(e1, e2, _)          -> (string_of_sexpr e1) ^ "." ^
    ↪  (string_of_sexpr e2)
110    |        SCall(f, el, _, _)                      -> f ^ "(" ^ String.concat ", "
    ↪  (List.map string_of_sexpr el) ^ ")"
111    |        SArrayPrimitive(el, _)            -> "|" ^ (string_of_sarray_primitive el) ^
    ↪  "|"
112    |          SUnop(op, e, _)                        -> (string_of_op op) ^ "(" ^
    ↪  string_of_sexpr e ^ ")"
113    |        SNull                                      -> "null"
114    |    SArrayCreate(d, el, _)        -> "new " ^ string_of_datatype d ^
    ↪  string_of_bracket_sexpr el
115    |    SArrayAccess(e, el, _)        -> (string_of_sexpr e) ^ (string_of_bracket_sexpr el)
116    |    SObjectCreate(s, el, _)        -> "new " ^ s ^ "(" ^ String.concat ", " (List.map
    ↪  string_of_sexpr el) ^ ")"
117    |        SDelete(e)                              -> "delete (" ^
    ↪  (string_of_sexpr e) ^ ")"
118    ;;
119
120    let string_of_local_expr = function
121    Noexpr -> ""
122    |        e              -> " = " ^ string_of_expr e
```

```
123
124  (* Print statements *)
125
126  let rec string_of_stmt indent =
127  let indent_string = String.make indent '\t' in
128  let get_stmt_string = function
129
130  Block(stmts)                           ->
131  indent_string ^ "{\n" ^
132          String.concat "" (List.map (string_of_stmt (indent+1)) stmts) ^
133          indent_string ^ "}\n"
134
135  |       Expr(expr)                              ->
136  indent_string ^ string_of_expr expr ^ ";\n";
137
138  |       Return(expr)                        ->
139  indent_string ^ "return " ^ string_of_expr expr ^ ";\n";
140
141  |       If(e, s, Block([Expr(Noexpr)]))        ->
142  indent_string ^ "if (" ^ string_of_expr e ^ ")\n" ^
143  (string_of_stmt (indent+1) s)
144
145  |       If(e, s1, s2)                      ->
146  indent_string ^ "if (" ^ string_of_expr e ^ ")\n" ^
147  string_of_stmt (indent+1) s1 ^
148  indent_string ^ "else\n" ^
149  string_of_stmt (indent+1) s2
150
151  |       For(e1, e2, e3, s)                 ->
152  indent_string ^ "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
     ↪  string_of_expr e3  ^ ")\n" ^
153  string_of_stmt (indent) s
154
155  |       While(e, s)                        ->
156  indent_string ^ "while (" ^ string_of_expr e ^ ")\n" ^
157  string_of_stmt (indent) s
158
159  |       Break                                         -> indent_string ^ "break;\n"
160  |       Continue                                    -> indent_string ^ "continue;\n"
161  |   Local(d, s, e)                        -> indent_string ^ string_of_datatype d ^ " "
     ↪  ^ s ^ string_of_local_expr e ^ ";\n"
162  in get_stmt_string
163
164  let string_of_local_sexpr = function
165  SNoexpr        -> ""
166  |       e                            -> " = " ^ string_of_sexpr e
167
168  let rec string_of_sstmt indent =
169  let indent_string = String.make indent '\t' in
```

```
170    let get_stmt_string = function
171
172    SBlock(stmts)                              ->
173    indent_string ^ "{\n" ^
174            String.concat "" (List.map (string_of_sstmt (indent+1)) stmts) ^
175            indent_string ^ "}\n"
176
177    |       SExpr(expr, _)                              ->
178    indent_string ^ string_of_sexpr expr ^ ";\n";
179
180    |       SReturn(expr, _)                       ->
181    indent_string ^ "return " ^ string_of_sexpr expr ^ ";\n";
182
183    |       SIf(e, s, SBlock([SExpr(SNoexpr, _)]))       ->
184    indent_string ^ "if (" ^ string_of_sexpr e ^ ")\n" ^
185    (string_of_sstmt (indent+1) s)
186
187    |       SIf(e, s1, s2)                     ->
188    indent_string ^ "if (" ^ string_of_sexpr e ^ ")\n" ^
189    string_of_sstmt (indent+1) s1 ^
190    indent_string ^ "else\n" ^
191    string_of_sstmt (indent+1) s2
192
193    |       SFor(e1, e2, e3, s)              ->
194    indent_string ^ "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
       ↪  string_of_sexpr e3  ^ ")\n" ^
195    string_of_sstmt (indent) s
196
197    |       SWhile(e, s)                    ->
198    indent_string ^ "while (" ^ string_of_sexpr e ^ ")\n" ^
199    string_of_sstmt (indent) s
200
201    |       SBreak                                        -> indent_string ^ "break;\n"
202    |       SContinue                                -> indent_string ^ "continue;\n"
203    |   SLocal(d, s, e)                       -> indent_string ^ string_of_datatype d ^ " "
       ↪  ^ s ^ string_of_local_sexpr e ^ ";\n"
204    in get_stmt_string
205
206    (* Print Function *)
207
208    let string_of_fname = function
209    Constructor -> "constructor"
210    |       FName(s)        -> s
211
212    let string_of_formal = function
213    Formal(d, s) -> (string_of_datatype d) ^ " " ^ s
214    |          _                           -> ""
215
216    let string_of_formal_name = function
```

```
217   Formal(_, s) -> s
218   |          _ -> ""
219
220   let string_of_func_decl fdecl =
221   "" ^ (string_of_scope fdecl.scope) ^ " " ^ (string_of_datatype fdecl.returnType) ^ " " ^
      ↪  (string_of_fname fdecl.fname) ^ " " ^
222   (* Formals *)
223   "(" ^ String.concat "," (List.map string_of_formal fdecl.formals) ^ ") {\n" ^
224          (* body *)
225          String.concat "" (List.map (string_of_stmt 2) fdecl.body) ^
226          "\t}\n\n"
227
228   (* Class Printing *)
229
230   let string_of_extends = function
231   NoParent        -> ""
232   |       Parent(s)       -> "extends " ^ s ^ " "
233   let string_of_field = function
234   Field(s, d, id) -> (string_of_scope s) ^ " " ^ (string_of_datatype d) ^ " " ^ id ^ ";\n"
235
236   let string_of_cbody cbody =
237   String.concat "" (List.map (fun s -> "\t" ^ s) (List.map string_of_field cbody.fields)) ^
238   String.concat "" (List.map (fun s -> "\t" ^ s) (List.map string_of_func_decl
      ↪  cbody.constructors)) ^
239   String.concat "" (List.map (fun s -> "\t" ^ s) (List.map string_of_func_decl
      ↪  cbody.methods))
240
241   let string_of_class_decl cdecl =
242   "class " ^ cdecl.cname ^ " " ^ (string_of_extends cdecl.extends) ^ "{\n" ^
243          (string_of_cbody cdecl.cbody) ^
244          "}\n"
245
246   (* Include Printing *)
247
248   let rec string_of_include = function
249   Include(s) -> "include(" ^ s ^ ");\n"
250
251   (* Print whole program *)
252
253   let string_of_program = function
254   Program(includes, cdecls) ->
255   String.concat "" (List.map string_of_include includes) ^ "\n" ^
256   String.concat "\n" (List.map string_of_class_decl cdecls)
257
258   (* Print AST tree representation *)
259
260   let includes_tree includes =
261   `List (List.map (function Include s -> `String s) includes)
262
```

```
263  let map_fields_to_json fields =
264  `List (List.map (function Field(scope, datatype, s) ->
265  `Assoc [
266  ("name", `String s);
267  ("scope", `String (string_of_scope scope));
268  ("datatype", `String (string_of_datatype datatype));
269  ]) fields)
270
271  let map_formals_to_json formals =
272  `List (List.map (function Formal(d, s) -> `Assoc [
273  ("name", `String s);
274  ("datatype", `String (string_of_datatype d));
275  ]
276  | Many d -> `Assoc [("Many", `String (string_of_datatype d));]
277  ) formals)
278
279  let rec map_expr_to_json = function
280  Int_Lit(i)                            -> `Assoc [("int_lit", `Int i)]
281  |      Boolean_Lit(b)                  -> `Assoc [("bool_lit", `Bool b)]
282  |      Float_Lit(f)                    -> `Assoc [("float_lit", `Float f)]
283  |      String_Lit(s)                   -> `Assoc [("string_lit", `String s)]
284  |      Char_Lit(c)                     -> `Assoc [("char_lit", `String
     ↪  (Char.escaped c))]
285  |      This                           -> `String "this"
286  |      Id(s)                          -> `Assoc [("id", `String s)]
287  |      Binop(e1, o, e2)               -> `Assoc [("binop", `Assoc [("lhs",
     ↪  map_expr_to_json e1); ("op", `String (string_of_op o)); ("rhs", map_expr_to_json
     ↪  e2)])]
288  |      Assign(e1, e2)                 -> `Assoc [("assign", `Assoc [("lhs",
     ↪  map_expr_to_json e1); ("op", `String "="); ("rhs", map_expr_to_json e2)])]
289  |      Noexpr                         -> `String "noexpr"
290  |      ObjAccess(e1, e2)              -> `Assoc [("objaccess", `Assoc [("lhs",
     ↪  map_expr_to_json e1); ("op", `String "."); ("rhs", map_expr_to_json e2)])]
291  |      Call(f, el)                    -> `Assoc [("call", `Assoc ([("name",
     ↪  `String f); ("params", `List (List.map map_expr_to_json el)); ]) )]
292  |      ArrayPrimitive(el)             -> `Assoc [("arrayprimitive", `List(List.map
     ↪  map_expr_to_json el))]
293  |        Unop(op, e)                  -> `Assoc [("Unop", `Assoc [("op",
     ↪  `String (string_of_op op)); ("operand", map_expr_to_json e)])]
294  |      Null                           -> `String "null"
295  |  ArrayCreate(d, el)         -> `Assoc [("arraycreate", `Assoc [("datatype", `String
     ↪  (string_of_datatype d)); ("args", `List (List.map map_expr_to_json el))])]
296  |  ArrayAccess(e, el)         -> `Assoc [("arrayaccess", `Assoc [("array",
     ↪  map_expr_to_json e); ("args", `List (List.map map_expr_to_json el))])]
297  |  ObjectCreate(s, el)        -> `Assoc [("objectcreate", `Assoc [("type", `String s);
     ↪  ("args", `List (List.map map_expr_to_json el))])]
298  |      Delete(e)                      -> `Assoc [("delete", `Assoc
     ↪  [("expr", map_expr_to_json e)])]
299
```

```
300  let rec map_stmt_to_json = function
301  Block(stmts)                           -> `Assoc [("block", `List (List.map
     ↪  (map_stmt_to_json) stmts))]
302  |       Expr(expr)                             -> `Assoc [("expr", map_expr_to_json
     ↪  expr)]
303  |       Return(expr)                       -> `Assoc [("return", map_expr_to_json
     ↪  expr)]
304  |       If(e, s1, s2)                    -> `Assoc [("if", `Assoc [("cond",
     ↪  map_expr_to_json e); ("ifbody", map_stmt_to_json s1)]); ("else", map_stmt_to_json
     ↪  s2)]
305  |       For(e1, e2, e3, s)             -> `Assoc [("for", `Assoc [("init",
     ↪  map_expr_to_json e1); ("cond", map_expr_to_json e2); ("inc", map_expr_to_json e3);
     ↪  ("body", map_stmt_to_json s)])]
306  |       While(e, s)                      -> `Assoc [("while", `Assoc [("cond",
     ↪  map_expr_to_json e); ("body", map_stmt_to_json s)])]
307  |       Break                                -> `String "break"
308  |       Continue                             -> `String "continue"
309  |   Local(d, s, e)                     -> `Assoc [("local", `Assoc [("datatype",
     ↪  `String (string_of_datatype d)); ("name", `String s); ("val", map_expr_to_json e)])]
310
311  let map_methods_to_json methods =
312  `List (List.map (fun (fdecl:Ast.func_decl) ->
313  `Assoc [
314  ("name", `String (string_of_fname fdecl.fname));
315  ("scope", `String (string_of_scope fdecl.scope));
316  ("returnType", `String (string_of_datatype fdecl.returnType));
317  ("formals", map_formals_to_json fdecl.formals);
318  ("body", `List (List.map (map_stmt_to_json) fdecl.body));
319  ]) methods)
320
321
322  let cdecls_tree cdecls =
323  let map_cdecl_to_json cdecl =
324  `Assoc [
325  ("cname", `String cdecl.cname);
326  ("extends", `String (string_of_extends cdecl.extends));
327  ("fields", map_fields_to_json cdecl.cbody.fields);
328  ("methods", map_methods_to_json cdecl.cbody.methods);
329  ("constructors", map_methods_to_json cdecl.cbody.constructors)
330  ]
331  in
332  `List (List.map (map_cdecl_to_json) cdecls)
333
334  let print_tree = function
335  Program(includes, cdecls) ->
336  `Assoc [("program",
337  `Assoc([
338  ("includes", includes_tree includes);
339  ("classes", cdecls_tree cdecls)
```

```
340    ])
341    )]
342
343    (* Print SAST tree representation *)
344
345    let rec map_sexpr_to_json =
346    let datatype d = [("datatype", `String (string_of_datatype d))] in
347    function
348    SInt_Lit(i)                 -> `Assoc [("int_lit", `Assoc ([("val", `Int i)] @ (datatype
       ↪ (Datatype(Int_t))))))]
349    |   SBoolean_Lit(b)            -> `Assoc [("bool_lit", `Assoc ([("val", `Bool b)] @
       ↪ (datatype (Datatype(Bool_t))))))]
350    |   SFloat_Lit(f)              -> `Assoc [("float_lit", `Assoc ([("val", `Float f)]  @
       ↪ (datatype (Datatype(Float_t))))))]
351    |   SString_Lit(s)             -> `Assoc [("string_lit", `Assoc ([("val", `String s)] @
       ↪ (datatype (Arraytype(Char_t, 1))))))]
352    |   SChar_Lit(c)               -> `Assoc [("char_lit", `Assoc ([("val", `String
       ↪ (Char.escaped c))] @ (datatype (Datatype(Char_t))))))]
353    |   SId(s, d)                  -> `Assoc [("id", `Assoc ([("name", `String s)] @ (datatype
       ↪ d)))]
354    |   SBinop(e1, o, e2, d)    -> `Assoc [("binop", `Assoc ([("lhs", map_sexpr_to_json e1);
       ↪ ("op", `String (string_of_op o)); ("rhs", map_sexpr_to_json e2)] @ (datatype d)))]
355    |   SAssign(e1, e2, d)      -> `Assoc [("assign", `Assoc ([("lhs", map_sexpr_to_json e1);
       ↪ ("op", `String "="); ("rhs", map_sexpr_to_json e2)] @ (datatype d)))]
356    |   SNoexpr                    -> `Assoc [("noexpr", `Assoc (datatype
       ↪ (Datatype(Void_t))))]
357    |   SArrayCreate(t, el, d)  -> `Assoc [("arraycreate", `Assoc ([("datatype", `String
       ↪ (string_of_datatype d)); ("args", `List (List.map map_sexpr_to_json el))] @ (datatype
       ↪ d)))]
358    |   SArrayAccess(e, el, d)  -> `Assoc [("arrayaccess", `Assoc ([("array",
       ↪ map_sexpr_to_json e); ("args", `List (List.map map_sexpr_to_json el))] @ (datatype
       ↪ d)))]
359    |   SObjAccess(e1, e2, d)   -> `Assoc [("objaccess", `Assoc ([("lhs", map_sexpr_to_json
       ↪ e1); ("op", `String "."); ("rhs", map_sexpr_to_json e2)] @ (datatype d)))]
360    |   SCall(fname, el, d, i)  -> `Assoc [("call", `Assoc ([("name", `String fname);
       ↪ ("params", `List (List.map map_sexpr_to_json el)); ("index", `Int i) ] @ (datatype
       ↪ d)) )]
361    |   SObjectCreate(s, el, d) -> `Assoc [("objectcreate", `Assoc ([("type", `String s);
       ↪ ("args", `List (List.map map_sexpr_to_json el))] @ (datatype d)))]
362    |   SArrayPrimitive(el, d)  -> `Assoc [("arrayprimitive", `Assoc ([("expressions",
       ↪ `List(List.map map_sexpr_to_json el))] @ (datatype d)))]
363    |   SUnop(op, e, d)         -> `Assoc [("Unop", `Assoc ([("op", `String (string_of_op
       ↪ op)); ("operand", map_sexpr_to_json e)] @ (datatype d)))]
364    |   SNull                      -> `Assoc [("null", `Assoc (datatype
       ↪ (Datatype(Void_t))))]
365    |       SDelete(e)                          -> `Assoc [("delete", `Assoc
       ↪ ([("expr", map_sexpr_to_json e)] @ (datatype (Datatype(Void_t))))))]
366
367    let rec map_sstmt_to_json =
```

```
368    let datatype d = [("datatype", `String (string_of_datatype d))] in
369    function
370    SBlock sl                              -> `Assoc [("sblock", `List (List.map
   ↪  (map_sstmt_to_json) sl))]
371    |   SExpr(e, d)                        -> `Assoc [("sexpr", `Assoc ([("expr",
   ↪  map_sexpr_to_json e)] @ (datatype d)))]
372    |   SReturn(e, d)                       -> `Assoc [("sreturn", `Assoc ([("return",
   ↪  map_sexpr_to_json e)] @ (datatype d)))]
373    |   SIf (e, s1, s2)                    -> `Assoc [("sif", `Assoc [("cond",
   ↪  map_sexpr_to_json e); ("ifbody", map_sstmt_to_json s1)]); ("selse", map_sstmt_to_json
   ↪  s2)]
374    |   SFor (e1, e2, e3, s)               -> `Assoc [("sfor", `Assoc [("init",
   ↪  map_sexpr_to_json e1); ("cond", map_sexpr_to_json e2); ("inc", map_sexpr_to_json e3);
   ↪  ("body", map_sstmt_to_json s)])]
375    |   SWhile (e, s)                      -> `Assoc [("swhile", `Assoc [("cond",
   ↪  map_sexpr_to_json e); ("body", map_sstmt_to_json s)])]
376    |   SBreak                             -> `String "sbreak"
377    |   SContinue                          -> `String "scontinue"
378    |   SLocal(d, s, e)                    -> `Assoc [("slocal", `Assoc [("datatype",
   ↪  `String (string_of_datatype d)); ("name", `String s); ("val", map_sexpr_to_json e)])]
379
380    let string_of_func_type = function
381    User -> "user" | Reserved -> "reserved"
382
383    let map_sfdecl_to_json sfdecl =
384    `Assoc[("sfdecl", `Assoc[
385    ("sfname", `String (string_of_fname sfdecl.sfname));
386    ("sreturnType", `String (string_of_datatype sfdecl.sreturnType));
387    ("sformals", map_formals_to_json sfdecl.sformals);
388    ("sbody", `List (List.map (map_sstmt_to_json) sfdecl.sbody));
389    ("func_type", `String(string_of_func_type sfdecl.func_type));
390    ])]
391
392    let map_sfdecls_to_json sfdecls =
393    `List(List.map map_sfdecl_to_json sfdecls)
394
395    let map_scdecls_to_json scdecls =
396    `List(List.map (fun scdecl ->
397    `Assoc [("scdecl",
398    `Assoc[
399    ("scname", `String scdecl.scname);
400    ("sfields", map_fields_to_json scdecl.sfields);
401    ("sfuncs", map_sfdecls_to_json scdecl.sfuncs);
402    ])
403    ])
404    scdecls)
405
406    let map_sprogram_to_json sprogram =
407    `Assoc [("sprogram", `Assoc [
```

```
408    ("classes", map_scdecls_to_json sprogram.classes);
409    ("functions", map_sfdecls_to_json sprogram.functions);
410    ("main", map_sfdecl_to_json sprogram.main);
411    ("reserved", map_sfdecls_to_json sprogram.reserved);
412    ])]
413
414    (* Print tokens *)
415
416    let string_of_token = function
417    LPAREN                                          -> "LPAREN"
418    |        RPAREN                                  -> "RPAREN"
419    |        LBRACE                                  -> "LBRACE"
420    |        RBRACE                                  -> "RBRACE"
421    |        SEMI                                    -> "SEMI"
422    |        COMMA                                   -> "COMMA"
423    |        PLUS                                    -> "PLUS"
424    |        MINUS                                   -> "MINUS"
425    |        TIMES                                   -> "TIMES"
426    |        DIVIDE                                  -> "DIVIDE"
427    |        ASSIGN                                  -> "ASSIGN"
428    |        EQ                                      -> "EQ"
429    |        NEQ                                     -> "NEQ"
430    |        LT                                      -> "LT"
431    |        LEQ                                     -> "LEQ"
432    |        GT                                      -> "GT"
433    |        GEQ                                     -> "GEQ"
434    |        AND                                     -> "AND"
435    |        OR                                      -> "OR"
436    |        NOT                                     -> "NOT"
437    |        DOT                                     -> "DOT"
438    |        LBRACKET                        -> "LBRACKET"
439    |        RBRACKET                        -> "RBRACKET"
440    |        BAR                                     -> "BAR"
441    |        IF                                      -> "IF"
442    |        ELSE                                    -> "ELSE"
443    |        FOR                                     -> "FOR"
444    |        WHILE                                   -> "WHILE"
445    |        RETURN                                  -> "RETURN"
446    |        INT                                     -> "INT"
447    |        FLOAT                                   -> "FLOAT"
448    |        BOOL                                    -> "BOOL"
449    |        CHAR                                    -> "CHAR"
450    |        VOID                                    -> "VOID"
451    |        NULL                                    -> "NULL"
452    |        TRUE                                    -> "TRUE"
453    |        FALSE                                   -> "FALSE"
454    |        CLASS                                   -> "CLASS"
455    |        CONSTRUCTOR                     -> "CONSTRUCTOR"
456    |        PUBLIC                                  -> "PUBLIC"
```

```
457  |        PRIVATE                              -> "PRIVATE"
458  |        EXTENDS                              -> "EXTENDS"
459  |        INCLUDE                              -> "INCLUDE"
460  |        THIS                               -> "THIS"
461  |        BREAK                               -> "BREAK"
462  |        CONTINUE                         -> "CONTINUE"
463  |    NEW                                 -> "NEW"
464  |        INT_LITERAL(i)              -> "INT_LITERAL(" ^ string_of_int i ^ ")"
465  |        FLOAT_LITERAL(f)        -> "FLOAT_LITERAL(" ^ string_of_float f ^ ")"
466  |        CHAR_LITERAL(c)              -> "CHAR_LITERAL(" ^ Char.escaped c ^ ")"
467  |        STRING_LITERAL(s)        -> "STRING_LITERAL(" ^ s ^ ")"
468  |        ID(s)                           -> "ID(" ^ s ^ ")"
469  |        DELETE                          -> "DELETE"
470  |        MODULO                          -> "MODULO"
471  |         EOF                                -> "EOF"
472
473  let string_of_token_no_id = function
474  LPAREN                                  -> "LPAREN"
475  |        RPAREN                              -> "RPAREN"
476  |        LBRACE                              -> "LBRACE"
477  |        RBRACE                              -> "RBRACE"
478  |        SEMI                               -> "SEMI"
479  |        COMMA                             -> "COMMA"
480  |        PLUS                             -> "PLUS"
481  |        MINUS                             -> "MINUS"
482  |        TIMES                             -> "TIMES"
483  |        DIVIDE                              -> "DIVIDE"
484  |        ASSIGN                              -> "ASSIGN"
485  |        EQ                                  -> "EQ"
486  |        NEQ                                  -> "NEQ"
487  |        LT                                  -> "LT"
488  |        LEQ                                  -> "LEQ"
489  |        GT                                  -> "GT"
490  |        GEQ                                  -> "GEQ"
491  |        AND                                  -> "AND"
492  |        OR                                  -> "OR"
493  |        NOT                                  -> "NOT"
494  |        DOT                                  -> "DOT"
495  |        LBRACKET                       -> "LBRACKET"
496  |        RBRACKET                       -> "RBRACKET"
497  |        BAR                                    -> "BAR"
498  |        IF                                  -> "IF"
499  |        ELSE                              -> "ELSE"
500  |        FOR                                   -> "FOR"
501  |        WHILE                             -> "WHILE"
502  |        RETURN                             -> "RETURN"
503  |        INT                                  -> "INT"
504  |        FLOAT                              -> "FLOAT"
505  |        BOOL                               -> "BOOL"
```

```
506  |          CHAR                                        -> "CHAR"
507  |          VOID                                        -> "VOID"
508  |          NULL                                        -> "NULL"
509  |          TRUE                                        -> "TRUE"
510  |          FALSE                                        -> "FALSE"
511  |          CLASS                                        -> "CLASS"
512  |          CONSTRUCTOR                                  -> "CONSTRUCTOR"
513  |          PUBLIC                                          -> "PUBLIC"
514  |          PRIVATE                                        -> "PRIVATE"
515  |          EXTENDS                                        -> "EXTENDS"
516  |          INCLUDE                                        -> "INCLUDE"
517  |          THIS                                        -> "THIS"
518  |          BREAK                                        -> "BREAK"
519  |          CONTINUE                                -> "CONTINUE"
520  |    NEW                                      -> "NEW"
521  |          INT_LITERAL(i)                  -> "INT_LITERAL"
522  |          FLOAT_LITERAL(f)        -> "FLOAT_LITERAL"
523  |          CHAR_LITERAL(c)                 -> "CHAR_LITERAL"
524  |          STRING_LITERAL(s)        -> "STRING_LITERAL"
525  |          ID(s)                                -> "ID"
526  |          DELETE                                 -> "DELETE"
527  |          MODULO                                 -> "MODULO"
528  |           EOF                                      -> "EOF"
529
530  let token_list_to_string_endl token_list =
531  let rec helper last_line_number = function
532  (token, curr)::tail ->
533  let line = curr.lineno in
534  (if line != last_line_number then "\n" ^ string_of_int line ^ ". " else " ") ^
535  string_of_token token ^ helper line tail
536  |          [] -> "\n"
537  in helper 0 token_list
538
539  let token_list_to_string token_list =
540  let rec helper = function
541  (token, line)::tail ->
542  string_of_token_no_id token ^ " " ^ helper tail
543  |          [] -> "\n"
544  in helper token_list
```

Test Suite Code

tester.sh

```bash
#!/bin/bash
# This script must reside in the "Test Suite" directory of the project
# Make sure the "dice" executable is in the "Compiler" directory

diceExecPath=./dice
testOption=$1 #stores the test flag since functions can't see the £1
vFlag=$2 #stores the -v flag since functions can't see it with £2
pass=0
fail=0
RED='\033[0;31m'
GREEN='\033[0;32m'
CYAN='\033[0;36m'
NC='\033[0m'
errorFile=errors.log
excpTestFlag=0

# Set time limit for all operations
ulimit -t 30

usage(){
        echo "Usage: $0 [test flag] [other]";
        echo "";
        echo "[test flag] = -c  Test Compiler (default if test flag not selected)";
        echo "              -d  Test Compiler and display Dice Compiler messages";
        echo "              -s  Test Scanner";
        echo "              -m  Run script without compiling Dice executable";
        echo "[other]     = -v  Verbose (prints log results)";
        exit 1;
}

confirmation(){
        #£? is the exit code for diff, if 0, then test output matched!
        if [ $? -eq 0 ];
                        then
                        echo -e "${GREEN}$filename passed!${NC}" >> session_file
                        echo -e "${GREEN}$filename passed!${NC}"
                        ((pass++))

              else
                        echo -e "${RED}$filename FAILED${NC}" >> session_file
                        echo -e "${RED}$filename FAILED${NC}"

                        #print out expected output and result
                        echo "Expected Output:" >> session_file

```

```
46                       if [ $excpTestFlag -eq 0 ];          then
47                               cat "$testPath"$filename$testExtension >> session_file
48                       else
49                               cat "$testExceptionsPath"$filename$testExtension >>
                          ↪   session_file
50                       fi
51                       echo "" >> session_file
52                       echo "Generated Output:" >> session_file
53                       cat temp_Dice_Tester  >> session_file
54                       echo "" >> session_file
55                       ((fail++))
56               fi
57   }
58
59   header(){
60           echo ""
61           echo "********************************************" >> session_file
62           echo "Dice Test Script Results:" >> session_file
63           date >> session_file
64           echo "" >> session_file
65   }
66
67   test_function(){
68           header #func
69
70           for testFile in "$testPath"*.dice; do
71
72                   filename=$(basename "$testFile")
73
74                   echo "===================================" >> session_file
75                   echo "Testing: $filename" >> session_file
76
77                   if [ "$testOption" == "-s" ]; then
78                           #Create file to be tested (with tokens)
79                           $diceExecPath $diceOption "$testFile" > temp_Dice_Tester
80                           #Test output differences use the diff command and neglect screen
                            ↪   output
81                           diff temp_Dice_Tester "$testPath"$filename$testExtension >
                            ↪   /dev/null
82                           confirmation #function
83                   else #Only other option is -c or -d which perform the same function
                      ↪   except where noted below
84                           #extract filename without extension for exectuable
85                           name=$(echo $filename | cut -f 1 -d '.')
86
87                           if [ "$testOption" == "-d" ]; then
88                                   #run the executable and port output (stderr) to temp test
                                    ↪   file
89                                   #port stdout (compiler msgs) to screen with color
```

```
 90                            echo -e -n "${CYAN}"
 91                            $diceExecPath $diceOption "$testFile" 2> temp.ll
 92                            echo -e -n "${NC}"
 93                            echo ""
 94
 95                    else
 96                            #Create header for any messages coming from Dice compiler
 97
 98                            echo -e "${CYAN}Dice Compiler Messages (if any):" >>
                            ↪  session_file
 99                            #run the executable and port output (stderr) to temp test
                            ↪  file
100                            #port stdout (compiler msgs) to log file
101                            $diceExecPath $diceOption "$testFile" 2> temp.ll 1>>
                            ↪  session_file
102                            echo -e "${NC}">> session_file
103                            echo "" >> session_file
104                    fi
105
106                    #Run the llvm executable and port output to temp test file
107                    lli temp.ll > temp_Dice_Tester
108
109                    #Send all error messages this script generates (if any) to error
                    ↪  log file
110                    exec 2> $errorFile
111
112                    #Perform comparison of outputs
113                    diff temp_Dice_Tester "$testPath"$filename$testExtension >
                    ↪  /dev/null
114                    confirmation #function
115            fi
116        done
117
118        #The following portion is only to test compiler errors
119        if [ "$testOption" == "-c" ] || [ "$testOption" == "-d" ] || [ "$testOption" ==
        ↪  "-m" ] || [ $# -eq 0 ]; then
120
121            #set flag to prevent
122            excpTestFlag=1
123            for testFile in "$testExceptionsPath"*.dice; do
124
125                    filename=$(basename "$testFile")
126
127                    echo "===================================" >> session_file
128                    echo "Testing: $filename" >> session_file
129
130                    #Only other option is -c or -d which perform the same function
                    ↪  except where noted below
131                    #extract filename without extension for exectuable
```

```bash
132                     name=$(echo $filename | cut -f 1 -d '.')
133
134                         #run the executable and port error  output (stdout) to temp test
          ↪ file
135                         #port stdout (compiler msgs) to log file
136                         $diceExecPath $diceOption "$testFile" 1> temp_Dice_Tester
          ↪ 2>/dev/null
137
138                         #Perform comparison of outputs
139                         diff temp_Dice_Tester
          ↪ "$testExceptionsPath"$filename$testExtension >> /dev/null
140                         confirmation #function
141                 done
142
143                 #Test if our executable can take in command line arguments:
144                 filename=test-args.dice
145                 $diceExecPath $diceOption "$argsPath"test-args.dice 2>temp.ll
146                 lli temp.ll david emily phil > tempArgs
147                 diff tempArgs "$argsPath"test-args.dice.out >/dev/null
148                 confirmation
149                 rm tempArgs
150
151         fi
152         echo "" >> session_file
153
154         #Verbose flag actuated
155         if [ "$vFlag" == "-v" ]; then
156                 cat session_file
157         fi
158
159         #Copy session output to historical log
160         cat session_file >> "$logFile"
161
162         #Test status output
163         echo ""
164         echo -e "${GREEN}Tests Passed: $pass ${NC}"
165         echo -e "${RED}Tests Failed: $fail ${NC}"
166         echo "View $logFile for more information"
167
168         #Clean up temp files
169         rm temp_Dice_Tester;
170         rm session_file;
171 }
172
173 createDice(){
174         echo "Compiling dice executable"
175         cd ..
176         make clean 2>&1 > /dev/null
177         make
```

```
178            #cp dice ../Test\ Suite/Hello_World_Demo/dice
179            # cd Test\ Suite
180            echo "Compilation of dice executable complete"
181    }
182
183    #-----------Script starts flag checking here ------------------
184    if [ "$testOption" == "-s" ]; then
185            echo "Scanner Test Started"
186            createDice
187            logFile=Test\ Suite/scanner_tests.log
188            testPath=Test\ Suite/Scanner\ Test\ Suite/
189            diceOption=-tendl
190            testExtension=.ManualTokens
191            test_function
192
193    elif [ "$testOption" == "-c" ] || [ "$testOption" == "-d" ] || [ "$testOption" == "-m" ]
    ↪   || [ $# -eq 0 ]; then
194            echo "Compiler Test Started"
195
196            if [ "$testOption" == "-m" ]; then
197                    if [ -f ../dice ]; then
198                            echo "Skipping Dice recompilation"
199                            cd ..
200                    else
201                            createDice
202                    fi
203            else
204                    createDice
205
206            fi
207
208            logFile=Test\ Suite/compiler_tests.log
209            testPath=Test\ Suite/Compiler_Test_Suite/
210            testExceptionsPath=Test\ Suite/Compiler_Test_Suite/Exceptions/
211            argsPath=Test\ Suite/Compiler_Test_Suite/Args/
212            diceOption=-c
213            testExtension=.out
214            test_function
215            rm temp.ll;
216
217    else
218            usage
219    fi
220
221    #Print out number of bash script errors and
222    if [ "$testOption" != "-s" ]; then
223            errorLines=$(cat $errorFile | wc -l)
224            mv $errorFile Test\ Suite/$errorFile
225            if [ $errorLines -ne 0 ]; then
```

```
226                echo "$errorLines lines of script errors reported. Please check $errorFile!"
227                else
228                        mv Test\ Suite/$errorFile
229                fi
230        fi
231
232    exit 0
```

test-var1.dice.out

1   42

test-stdlib-stringclass.dice.out

```
1  hi
```

test-stdlib-integerclass1.dice

```
1  include("stdlib");
2
3  class Two {
4        public void main(char[][] args) {
5        class Integer x = new Integer(128);
6        print(x.num(), "\n");
7      }
8  }
```

test-constructorInherited.dice

```
1   class shape {
2     public int xCoord;
3     public int yCoord;
4
5     constructor(){
6     this.xCoord = 0;
7     this.yCoord = 0;
8     }
9
10    constructor(int x, int y){
11    this.xCoord = x;
12    this.yCoord = y;
13    }
14  }
15
16  class circle extends shape {
17    public int radius;
18
19    constructor(){
20            this.radius = 0;
21    }
22    constructor(int r){
23            this.radius = r;
24    }
25    constructor(int x, int y, int r){
26            this.radius = r;
27            this.xCoord = x;
28            this.yCoord = y;
29    }
30  }
31
32  class test {
33    public void main(char[][] args) {
34        class circle a = new circle(0,0,7);
35        print(a.xCoord);
36        print(a.yCoord);
37        print(a.radius);
38    }
39  }
```

test-ifEmptyBlock2.dice.out

1   17

test-global1.dice.out

1   42214322

test-if7.dice

```
1   class test {
2         public void main(char[][] args) {
3
4                 if(false) {
5                         print("if");
6                 }
7
8                 else if(false) {
9                         print("elseif");
10                }
11
12                else if(false) {
13                        print("elseif2");
14                }
15
16                else {
17                        print("else");
18                }
19        }
20  }
```

test-var3.dice

```
1   class test {
2
3         public int a;
4
5         public void print2(int x, int y) {
6           print(x);
7           print(y);
8         }
9
10        public void main(char[][] args) {
11          int b;
12          this.a = 42;
13          b = 57;
14          this.print2(this.a + b * 3, 77);
15        }
16  }
```

test-classFunctionOverload1.dice.out

```
1    10
```

test-applicative.dice

```
1   class test {
2
3           public int p(int i){
4                   print(i);
5                   return i;
6           }
7
8           public void q(int a, int b, int c){
9                   int total = a ;
10                  print(b);
11                  total = total + c ;
12          }
13
14          public void main(char[][] args) {
15                  this.q( this.p(1), 2, this.p(3));
16          }
17  }
```

test-forEmptyBlock2.dice

```
1   class test {
2         public void main(char[][] args) {
3           int i;
4           for (i = 0 ; i < 5 ; i = i + 1) {
5            (*empty block*) null;
6           }
7           print(1);
8         }
9   }
```

test-if1.dice

```
1  class test {
2          public void main(char[][] args) {
3                  if (true) print(42);
4                  print(17);
5          }
6  }
```

test-func5.dice

```
1   class test {
2
3           public void foo(int a, int b){
4             int c;
5             int d;
6             int e;
7             print(a);
8             e = a + b + 10;
9             print(e);
10          }
11
12          public void main(char[][] args) {
13                   this.foo(1,2);
14          }
15
16  }
```

test-arith5.dice

```
1  class test {
2          public void main(char[][] args) {
3                  print(15-5);
4          }
5  }
```

test-bool5.dice

```
1  class test {
2          public void main(char[][] args) {
3                  print(1==2);
4                  print(1==1);
5          }
6  }
```

test-constructor2.dice

```
1   class shape {
2           public int xCoord;
3           public int yCoord;
4
5           constructor(int x, int y){
6                   this.xCoord = x;
7                   this.yCoord = y;
8           }
9
10          constructor(float x, float y){
11                  this.xCoord = 0;
12                  this.yCoord = 0;
13          }
14  }
15
16   class test {
17          public void main(char[][] args) {
18                   class shape a = new shape(5,10);
19                   print (a.xCoord);
20                   print (a.yCoord);
21          }
22   }
```

test-arithSigned2.dice.out

1 -3-3-3.000000-3.000000

test-classExtends2.dice

```
1   class person {
2     public int ssn;
3   }
4
5   class worker extends person {
6     public int workid;
7   }
8
9   class programmer extends worker {
10          public int nerdCred;
11  }
12
13  class test {
14    public void main(char[][] args) {
15        class programmer david = new programmer();
16        david.ssn = 123456789;
17        david.workid = 57;
18        david.nerdCred = 99;
19
20        print(david.ssn);
21        print(david.workid);
22        print(david.nerdCred);
23    }
24  }
```

test-arithSigned1.dice.out

1   -5-5-5.000000-5.000000

test-forEmptyBlock.dice

```
1   class test {
2         public void main(char[][] args) {
3           int i;
4           for (i = 0 ; i < 5 ; i = i + 1) {
5            (*empty block*)
6           }
7           print(1);
8         }
9   }
```

test-func5.dice.out

1   113

test-float.dice.out

```
1   1.500000
```

test-stdlib-integerclass1.dice.out

1   128

test-for2.dice.out

1 5432142

test-if4.dice

```
1  class test {
2        public void main(char[][] args) {
3          if (false)
4                print(42);
5          else
6                print(8);
7            print(17);
8        }
9  }
```

test-arith7.dice

```
1   class test {
2           public void main(char[][] args) {
3                   print(15/5);
4           }
5   }
```

test-if5.dice

```
1   class test {
2           public void main(char[][] args) {
3                   this.foo(3,5,6);
4           }
5
6           public void foo(int a, int b, int c) {
7                   int d;
8                   if (a == 3)
9                     d = b;
10                  else
11                    d = c;
12                   print(d);
13                 }
14   }
```

test-arithSigned3.dice

```
1  class test {
2          public void main(char[][] args) {
3                  print(-1+3);
4                  print(1+-3);
5                  print(-1.0+3.0);
6                  print(1.0+-3.0);
7          }
8  }
```

test-if7.dice.out

```
1  else
```

test-classGetter.dice.out

1   13

test-stdlib-compare.dice

```
1   include("stdlib");
2
3   class Two {
4           public void main(char[][] args) {
5           class String b = new String("phil");
6           class String c = new String("khal");
7           class String d = c.copy(c);
8           print(b.string(), " == ", c.string(), " is ", b.compare(c));
9           print(c.string(), " == ", d.string(), " is ", c.compare(d));
10          }
11  }
```

test-class.dice.out

1   13

test-for1.dice.out

1   0123442

test-classInheritanceArgument.dice

```
1   class shape {
2     public int xCoord;
3     public int yCoord;
4   }
5
6   class circle extends shape {
7     public int radius;
8   }
9
10  class test {
11
12    public void main(char[][] args) {
13        class circle a = new circle();
14        this.inheritanceTest(a);
15    }
16
17    public void inheritanceTest(class shape a){
18      print("pass");
19    }
20
21  }
```

test-whileBreak.dice

```
1   class test {
2         public void main(char[][] args) {
3            int i;
4            i = 5;
5            while (i > 0) {
6              print(i);
7              if(i==3){
8                      break;
9                      }
10             i = i - 1;
11           }
12         }
13   }
```

test-while1.dice

```
1   class test {
2         public void main(char[][] args) {
3            int i;
4            i = 5;
5            while (i > 0) {
6              print(i);
7              i = i - 1;
8            }
9            print(42);
10         }
11   }
```

test-fileio.dice.out

```
1  include("stdlib");
2
3  class Two {
4
5          public void main(char[][] args) {
6          class File a = new File("Test Suite/Compiler_Test_Suite/test-fileio.dice", true);
7          char[] buf = a.readfile(243);
8          a.closefile();
9          print(buf);
10         }
11 }
```

test-classExtends2.dice.out

```
1  1234567895799
```

test-forContinue.dice.out

1   23420

test-fib.dice

```
1   class test {
2
3           public int fib(int x) {
4                   if (x < 2)
5                           return 1;
6                   return this.fib(x-1) + this.fib(x-2);
7           }
8
9           public void main(char[][] args) {
10                  print(this.fib(0));
11                  print(this.fib(1));
12                  print(this.fib(2));
13                  print(this.fib(3));
14                  print(this.fib(4));
15                  print(this.fib(5));
16          }
17  }
```

test-bool1.dice

```
1  class test {
2          public void main(char[][] args) {
3                  print(1<2);
4                  print(1.0<2);
5                  print(1<2.0);
6                  print(1.0<2.0);
7          }
8  }
```

test-forBreak.dice

```
1   class test {
2         public void main(char[][] args) {
3            int i;
4            for (i = 0 ; i < 5 ; i = i + 1) {
5                   if(i==3){
6                          break;
7                   }
8              print(i);
9            }
10           print(100);
11        }
12  }
```

test-bool6.dice

```
1  class test {
2          public void main(char[][] args) {
3                  print(1!=2);
4                  print(1!=1);
5          }
6  }
```

test-bool4.dice.out

1   truetruetruefalse

test-stdlib-stringclassContains2.dice

```
1   include("stdlib");
2
3   class Two {
4        public void main(char[][] args) {
5        class String b = new String("philkhal");
6        class String c = new String("butts");
7        print(b.contains(c));
8        }
9   }
```

test-classGetter.dice

```
1   class shape {
2     public int xCoord;
3     public int yCoord;
4
5     public int getX(){
6             return this.xCoord;
7     }
8     public int getY(){
9             return this.yCoord;
10    }
11
12  }
13
14  class test {
15    public void main(char[][] args) {
16        class shape a = new shape();
17        a.xCoord = 1;
18        a.yCoord = 3;
19        print(a.getX());
20        print(a.getY());
21    }
22  }
```

test-var3.dice.out

1   21377

test-forContinue.dice

```
1   class test {
2          public void main(char[][] args) {
3            int i;
4            for (i = 0 ; i < 5 ; i = i + 1) {
5                    if(i<2){ continue; }
6                    else{
7            print(i);
8                }
9            }
10           print(20);
11         }
12  }
```

test-stdlib-stringclassReverse.dice.out

```
1  olleh
```

test-while1.dice.out

1 5432142

test-float.dice

```
1   class test {
2     public void main(char[][] args) {
3         float a = 1.5;
4         print(a);
5
6       }
7   }
```

test-arith5.dice.out

1   10

test-array4.dice

```
1   class shape {
2           public int x;
3           public int y;
4
5           constructor(int a, int b){
6           this.x = a;
7           this.y = b;
8           }
9
10  }
11
12  class test {
13          public void main(char[][] args) {
14                  class shape[] a = new class shape[5];
15                  class shape b = new shape(2,3);
16                  a[1] = b;
17                  print(a[1].x);
18          }
19  }
```

test-arithSigned1.dice

```
1   class test {
2         public void main(char[][] args) {
3                 print(-15/3);
4                 print(15/-3);
5                 print(-15.0/3.0);
6                 print(15.0/-3.0);
7         }
8   }
```

test-if2.dice.out

1   4217

test-stdlib-concat.dice

```
1   include("stdlib");
2
3   class Two {
4         public void main(char[][] args) {
5         class String b = new String("phil");
6         class String c = new String("khal");
7         class String a = b.concat(c);
8         print(b.string(), "\n");
9         print(c.string(), "\n");
10        print(a.string(), "\n");
11        }
12  }
```

test-classReturnObjects.dice.out

1   12

test-if8.dice

```
1   class test {
2         public void main(char[][] args) {
3
4                 if(false) {
5                         print("if");
6                 }
7
8                 else if(true) {
9                         print("elseif");
10                }
11
12                else if(false) {
13                        print("elseif2");
14                }
15
16                else {
17                        print("else");
18                }
19        }
20  }
```

test-stmts1.dice

```
1   class test {
2           public void main(char[][] args) {
3                   print(this.foo(1,42));
4                    print(this.foo(0,37));
5           }
6
7           public int foo(int a, int b) {
8              int i;
9              int j = b;
10             if ( a == 1)
11                return b + 3;
12             else
13                for (i = 0 ; i < 5 ; i = i + 1)
14                    j = j + 5;
15             return j;
16          }
17   }
```

test-if6.dice.out

```
1    42278
```

test-classExtendsGetter.dice.out

1   13

test-ops1.dice.out

1 3-125099falsetrue99truefalse99truefalse99truetruefalse99falsetrue99falsetruetrue

test-arith4.dice

```
1   (* Test side-effect sequence in a series of statement *)
2
3   class test {
4           public int g;
5
6           public void main(char[][] args) {
7
8             int l;
9             l = 1;
10            print(l);
11
12            this.g = 3;
13            print(this.g);
14
15            l = 5;
16            print(l+100);
17
18            this.g = 7;
19            print(this.g+100);
20          }
21  }
```

test-func3.dice.out

1   42171928

test-class.dice

```
1   class shape {
2     public int xCoord;
3     public int yCoord;
4
5     constructor (){
6     }
7   }
8
9   class test {
10    public void main(char[][] args) {
11        class shape a = new shape();
12        a.xCoord = 1;
13        a.yCoord = 3;
14        print(a.xCoord);
15        print(a.yCoord);
16    }
17  }
```

test-bool9.dice.out

1   truetruetruefalsefalsetruetruefalse

test-whileContinue.dice.out

1   543

test-stdlib-copy.dice.out

```
1 philkhalkhal
```

test-stdlib-integerclass2.dice.out

1   128

test-classExtends.dice

```
1  class shape {
2    public float xCoord;
3    public float yCoord;
4  }
5
6  class circle extends shape {
7    public float radius;
8  }
9
10 class test {
11   public void main(char[][] args) {
12       class circle a = new circle();
13       a.xCoord = 1.5;
14       print(a.xCoord);
15   }
16 }
```

test–if3.dice

```
1  class test {
2          public void main(char[][] args) {
3                      if (false)
4                              print(42);
5                      print(17);
6          }
7  }
```

test-bool8.dice.out

```
1  falsetrue
2  falsefalse
```

test-scope.dice.out

1 12321

test-constructor1.dice

```
1    class shape {
2            public int xCoord;
3            public int yCoord;
4
5            constructor(){
6                    this.xCoord = 0;
7                    this.yCoord = 0;
8            }
9
10           constructor(int x, int y){
11                    this.xCoord = x;
12                    this.yCoord = y;
13           }
14   }
15
16    class test {
17            public void main(char[][] args) {
18                    class shape a = new shape();
19                    class shape b = new shape(5,10);
20                    print (a.xCoord);
21                    print (a.yCoord);
22                    print (b.xCoord);
23                    print (b.yCoord);
24            }
25    }
```

test-stdlib-concat.dice.out

```
1  phil
2  khal
3  philkhal
```

test-forEmptyBlock2.dice.out

```
1    1
```

test-if4.dice.out

1   817

test-array.dice.out

```
1   04
```

test-array2.dice.out

1   1.5000004.500000

test-objectDeclarationInheritance.dice.out

```
1  pass
```

test-if5.dice.out

```
1    5
```

test-forEmptyBlock.dice.out

```
1    1
```

test-var4.dice.out

1   1242

test-whileContinue.dice

```
1   class test {
2         public void main(char[][] args) {
3            int i;
4            i = 6;
5            while (i > 0) {
6                    i = i - 1;
7
8              if(i<3){
9                    continue;
10             }
11
12             print(i);
13           }
14         }
15  }
```

test-array3.dice

```
1   class test {
2         public void main(char[][] args) {
3               int[] a = new int[10];
4               a[0] = 1;
5               print(a[0]);
6               a[0] = 10;
7               print(a[0]);
8               a[9] = 2;
9               print(a[9]);
10        }
11  }
```

test-if3.dice.out

1   17

test-arith6.dice

```
1   class test {
2           public void main(char[][] args) {
3                   print(10*5);
4           }
5   }
```

test-helloTwice.dice.out

```
1  Hello, World!
2  Professor Edwards favorite number is: 42!
```

test-stdlib-stringclassLength.dice.out

```
1    9
```

test-bool3.dice.out

1   falsetruefalsetrue

test-hello.dice

```
1  class test {
2          public void main(char[][] args) {
3                  print("Hello, World!");
4          }
5  }
```

test-array.dice

```
1  class test {
2         public void main(char[][] args) {
3                 int[] a = |0,1,2,3,4|;
4                 print(a[0]);
5                 print(a[4]);
6         }
7  }
```

test-exit.dice

```
1  class test {
2          public void main(char[][] args) {
3
4                  print(1);
5                  exit(1);
6                  print(2);
7
8          }
9  }
```

test-helloTwice.dice

```
1   class test {
2         public void main(char[][] args) {
3                 print("Hello, World!\n");
4                 print("Professor Edwards favorite number is: 42!\n");
5         }
6   }
```

test-arithSigned2.dice

```
1  class test {
2          public void main(char[][] args) {
3                  print(-1*3);
4                  print(1*-3);
5                  print(-1.0*3.0);
6                  print(1.0*-3.0);
7          }
8  }
```

test-cyclicalIncludes2.dice

```
1   include("Test Suite/Compiler_Test_Suite/test-cyclicalIncludes.dice");
2
3   class test2 {
4           constructor(){
5                   this.output2();
6           }
7           public void output2(){
8                   print("b");
9           }
10  }
```

test–if2.dice

```
1  class test {
2          public void main(char[][] args) {
3                  if (true) print(42);
4                  else print(8);
5                    print(17);
6          }
7  }
```

test-constructorDefault.dice

```
1    class shape {
2            public int xCoord;
3            public int yCoord;
4
5    }
6
7     class test {
8            public void main(char[][] args) {
9                    class shape a = new shape();
10                   a.xCoord = 5;
11                   print (a.xCoord);
12           }
13    }
```

test-var1.dice

```
1  class test {
2          public void main(char[][] args) {
3
4            int a;
5            a = 42;
6            print(a);
7          }
8  }
```

test-arithSigned4.dice.out

```
1   -44-4.0000004.000000
```

test-ifEmptyBlock.dice.out

1   17

test-stdlib-compare.dice.out

```
1  phil == khal is falsekhal == khal is true
```

test-cyclicalIncludes.dice.out

1   ba

test-bool7.dice.out

```
1  truefalsefalsefalse
2  truetruetruefalse
```

test-classSetter.dice.out

1   13

test-stdlib-stringclassReverse.dice

```
1   include("stdlib");
2
3   class Test {
4       public void main(char[][] args) {
5           class String a = new String("hello");
6           class String reverse = a.reverse();
7
8           print(reverse.string());
9       }
10  }
```

test-factorialRecursive.dice

```
1   class Factorial {
2
3       public void main(char[][] args) {
4           print(this.factorial(5));
5       }
6
7       public int factorial(int n) {
8        int temp;
9        if(n <= 1) return 1;
10       temp = n * this.factorial(n - 1);
11       return temp;
12      }
13  }
```

test-classInheritanceArgument.dice.out

```
1  pass
```

test-constructorInherited.dice.out

1   007

test-bool8.dice

```
1  class test {
2      public void main(char[][] args) {
3          print(not true);
4          print(not false);
5          print("\n");
6          print(not true and true);
7          print(not (true and true));
8      }
9  }
```

test-classFunctionOverload.dice

```
1   class shape {
2     public int xCoord;
3     public int yCoord;
4
5     constructor(){
6     this.xCoord = 0;
7     this.yCoord = 0;
8     }
9
10    constructor(int x, int y){
11    this.xCoord = x;
12    this.yCoord = y;
13    }
14
15    public int getArea(){
16      return 10;
17    }
18  }
19
20  class circle extends shape {
21    public int radius;
22
23    constructor(){
24          this.radius = 0;
25    }
26    constructor(int r){
27          this.radius = r;
28    }
29    constructor(int x, int y, int r){
30          this.radius = r;
31          this.xCoord = x;
32          this.yCoord = y;
33    }
34
35    public int getArea(){
36      return 3*this.radius*this.radius;
37    }
38  }
39
40  class test {
41    public void main(char[][] args) {
42        class circle a = new circle(0,0,2);
43        print(a.getArea());
44    }
45  }
```

test-stdlib-stringclassContains.dice.out

```
1  true
```

test-arith8.dice

```
1  class test {
2          public void main(char[][] args) {
3                  print(15+5.0);
4                  print("\n");
5                  print(1.5+1);
6          }
7  }
```

test-array4.dice.out

```
1    2
```

test-stdlib-copy.dice

```
1   include("stdlib");
2
3   class Two {
4           public void main(char[][] args) {
5           class String b = new String("phil");
6           class String c = new String("khal");
7           class String d = c.copy(c);
8           print(b.string());
9           print(c.string());
10          print(d.string());
11          }
12  }
```

test-arith7.dice.out

```
1    3
```

test-classFunctionOverload1.dice

```
1   class shape {
2     public int xCoord;
3     public int yCoord;
4
5     constructor(){
6     this.xCoord = 0;
7     this.yCoord = 0;
8     }
9
10    constructor(int x, int y){
11    this.xCoord = x;
12    this.yCoord = y;
13    }
14
15    public int getArea(){
16      return 10;
17    }
18  }
19
20  class circle extends shape {
21    public int radius;
22
23    constructor(){
24            this.radius = 0;
25    }
26    constructor(int r){
27            this.radius = r;
28    }
29    constructor(int x, int y){
30            this.radius = 0;
31            this.xCoord = x;
32            this.yCoord = y;
33    }
34
35    public int getArea(){
36      return 3*this.radius*this.radius;
37    }
38  }
39
40  class test {
41    public void main(char[][] args) {
42        class shape a = new shape(0,0);
43        print(a.getArea());
44    }
45  }
```

test-stdlib-stringclassContains.dice

```
1  include("stdlib");
2
3  class Two {
4        public void main(char[][] args) {
5        class String b = new String("philkhal");
6        class String c = new String("khal");
7        print(b.contains(c));
8        }
9  }
```

test-factorialRecursive.dice.out

1   120

test-stdlib-integerclass2.dice

```
1   include("stdlib");
2
3   class Two {
4           public void main(char[][] args) {
5           class Integer x = new Integer(128);
6           class String str = x.toString();
7           print(str.string(), "\n");
8           }
9   }
```

test-bool6.dice.out

1   truefalse

test-cyclicalIncludes.dice

```
1   include("Test Suite/Compiler_Test_Suite/test-cyclicalIncludes2.dice");
2
3   class test {
4           public void main(char[][] args) {
5           class test2 a = new test2();
6           this.output();
7           }
8
9           public void output(){
10                  print("a");
11          }
12  }
```

test-bool1.dice.out

1   truetruetruetrue

test-stdlib-stringclass3.dice

```
1   include("stdlib");
2
3   class test{
4
5           private class String x;
6
7           public void main(char[][] args) {
8
9           class String a = new String("goodBye");
10          this.x = a;
11          print(this.x.string());
12
13          }
14  }
```

test-arith3.dice

```
1   (* Test left-to-right evaluation of expressions *)
2
3   class test {
4
5           public int a; (* Global variable *)
6
7           public int inca() {
8                   this.a = this.a + 1;  (* Increment a; return its new value *)
9                   return this.a;
10          }
11
12          public void main(char[][] args) {
13                  this.a = 42;    (* Initialize a *)
14                  print(this.inca() + this.a);
15          }
16  }
```

test-emptyBlock.dice

```
1   class test {
2           public void main(char[][] args) {
3                   {
4                   (* Nothing in the following blocks*) {} {}
5                   }
6                   { null; }
7                   print(1);
8           }
9   }
```

test-intOverflow.dice.out

1   passpass

test-stdlib.dice.out

1 hi

test-classFunctionOverload.dice.out

```
1    12
```

test-exit.dice.out

```
1    1
```

test-if1.dice.out

1   4217

test-stdlib-stringclass2.dice

```
1  include("stdlib");
2
3  class test {
4        public void main(char[][] args) {
5        class String s = new String("StringDoesn'tStartWithH");
6        print(s.string());
7        }
8  }
```

test-arith6.dice.out

1   50

test-stdlib-stringclassLength.dice

```
1   include("stdlib");
2
3   class Two {
4           public void main(char[][] args) {
5           class String s = new String("123456789");
6           print(s.length());
7           }
8   }
```

test-stdlib-stringclassContains2.dice.out

1   false

test-ops1.dice

```
1   class test {
2     public void main(char[][] args) {
3         print(1 + 2);
4         print(1 - 2);
5         print(1 * 2);
6         print(100 / 2);
7         print(99);
8         print(1 == 2);
9         print(1 == 1);
10        print(99);
11        print(1 != 2);
12        print(1 != 1);
13        print(99);
14        print(1 < 2);
15        print(2 < 1);
16        print(99);
17        print(1 <= 2);
18        print(1 <= 1);
19        print(2 <= 1);
20        print(99);
21        print(1 > 2);
22        print(2 > 1);
23        print(99);
24        print(1 >= 2);
25        print(1 >= 1);
26        print(2 >= 1);
27    }
28  }
```

test-stdlib-stringclass.dice

```
1   include("stdlib");
2
3   class Two {
4           public void main(char[][] args) {
5           class String s = new String("hi");
6           print(s.string());
7           }
8   }
```

test-arith2.dice

```
1  class test {
2          public void main(char[][] args) {
3                          print(1 + 2 * 3 + 4);
4          }
5  }
```

test-float-max.dice

```
1   class test {
2     public void main(char[][] args) {
3         float a = 0.01175494;
4         float b = 1010123.45;
5         print(a);
6         print("\n");
7         print(b);
8
9       }
10  }
```

test-arith1.dice

```
1   class test {
2           public void main(char[][] args) {
3                   print(5+15);
4           }
5   }
```

test-stdlib-stringclass3.dice.out

```
1   goodBye
```

test-ifEmptyBlock2.dice

```
1   class test {
2         public void main(char[][] args) {
3                   if (false){}
4                   else {}
5                   print(17);
6         }
7   }
```

test-array3.dice.out

1   1102

test-arithSigned4.dice

```
1   class test {
2          public void main(char[][] args) {
3                  print(-1-3);
4                  print(1--3);
5                  print(-1.0-3.0);
6                  print(1.0--3.0);
7          }
8   }
```

test-classSetter.dice

```
1   class shape {
2     public int xCoord;
3     public int yCoord;
4
5     public void setX(int x){
6             this.xCoord = x;
7     }
8     public void setY(int y){
9             this.yCoord = y;
10    }
11
12  }
13
14  class test {
15    public void main(char[][] args) {
16        class shape a = new shape();
17        a.setX(1);
18        a.setY(3);
19        print(a.xCoord);
20        print(a.yCoord);
21    }
22  }
```

test-classExtendsSetter.dice

```
1   class shape {
2     public int xCoord;
3     public int yCoord;
4
5     public void setX(int x){
6             this.xCoord = x;
7     }
8     public void setY(int y){
9             this.yCoord = y;
10    }
11
12  }
13
14  class circle extends shape {
15    public int radius;
16
17  }
18
19  class test {
20    public void main(char[][] args) {
21        class circle a = new circle();
22        a.setX(1);
23        a.setY(3);
24        print(a.xCoord);
25        print(a.yCoord);
26    }
27  }
```

test-gcd.dice

```
1   class test {
2
3           public void main(char[][] args) {
4                   print(this.gcd(2,14));
5                   print(this.gcd(3,15));
6                   print(this.gcd(99,121));
7           }
8
9           public int gcd(int x, int y){
10                  int a = x;
11                  int b = y;
12                    while (a != b) {
13                      if (a > b)
14                              a = a - b;
15                      else
16                              b = b - a;
17                          }
18                    return a;
19          }
20  }
```

test-bool7.dice

```
1   class test {
2           public void main(char[][] args) {
3                   print(true and true);
4                   print(false and true);
5                   print(true and false);
6                   print(false and false);
7                   print("\n");
8                   print(true or true);
9                   print(false or true);
10                  print(true or false);
11                  print(false or false);
12          }
13  }
```

test-classExtendsGetter.dice

```
1   class shape {
2     public int xCoord;
3     public int yCoord;
4
5     public int getX(){
6             return this.xCoord;
7     }
8     public int getY(){
9             return this.yCoord;
10    }
11
12  }
13
14  class circle extends shape {
15    public int radius;
16
17  }
18
19  class test {
20    public void main(char[][] args) {
21        class circle a = new circle();
22        a.xCoord = 1;
23        a.yCoord = 3;
24        print(a.getX());
25        print(a.getY());
26    }
27  }
```

test-func4.dice.out

1   371

test-constructor1.dice.out

1   00510

test-fib.dice.out

1   112358

test-forBreak.dice.out

1   012100

test-func3.dice

```
1   class test {
2          public void main(char[][] args) {
3                   this.printem(42,17,192,8);
4          }
5
6          public void printem(int a, int b, int c, int d) {
7                   print(a);
8                   print(b);
9                   print(c);
10                  print(d);
11         }
12  }
```

test-scope.dice

```
1   class test {
2         public void main(char[][] args) {
3            int a;
4           a = 1;
5           {
6                   int b = 2;
7                   {
8                           int c = 3;
9                           print(a);
10                          print(b);
11                          print(c);
12                  }
13                  print(b);
14          }
15          print(a);
16      }
17  }
```

test-objectDeclarationInheritance.dice

```
1   class A {}
2   class B extends A {}
3   class C extends B {}
4
5   class test {
6
7           public void main(char[][] args) {
8                   class A myCObj = new C();
9                   print("pass");
10          }
11  }
```

test-bool9.dice

```
1  class test {
2      public void main(char[][] args) {
3          print(true, true, true, false, false, true, true, false, "\n");
4      }
5  }
```

test-if8.dice.out

```
1  elseif
```

test-hello.dice.out

```
1   Hello, World!
```

test-fileio.dice

```
1   include("stdlib");
2
3   class Two {
4
5           public void main(char[][] args) {
6           class File a = new File("Test Suite/Compiler_Test_Suite/test-fileio.dice", true);
7           char[] buf = a.readfile(243);
8           a.closefile();
9           print(buf);
10          }
11  }
```

test-arith3.dice.out

1   86

test-float-max.dice.out

```
1  0.011755
2  1010123.450000
```

test-var4.dice

```
1   class test {
2         public int a;
3
4         public void foo(int b) {
5           int c;
6           c = this.a;
7           print(c);
8           this.a = b;
9           print(this.a);
10        }
11
12        public void main(char[][] args) {
13          this.a = 12;
14          this.foo(42);
15        }
16   }
```

test-cyclicalIncludes2.dice.out

```
1  ba
```

test-classExtendsSetter.dice.out

1   13

test-bool4.dice

```
1   class test {
2           public void main(char[][] args) {
3                   print(1<=2);
4                   print(1<=1);
5                   print(1<=2.0);
6                   print(2.1<=2.0);
7           }
8   }
```

test-bool2.dice

```
1   class test {
2           public void main(char[][] args) {
3                   print(1>2);
4                   print(1.0>2);
5                   print(1>2.0);
6                   print(1.0>2.0);
7           }
8   }
```

test-classExtends.dice.out

```
1   1.500000
```

test-gcd.dice.out

1   2311

test-bool2.dice.out

1 falsefalsefalsefalse

test-func4.dice

```
1   class test {
2         public int a;
3
4         constructor() {}
5
6         public int inca() {
7                 this.a = 124;
8                 return this.a + 124;
9         }
10
11        public int add2(int x, int y) {
12                return x + y;
13        }
14
15        public void main(char[][] args) {
16                class test b = new test();
17                  print(b.add2(b.inca(), 123));
18        }
19  }
```

test-emptyBlock.dice.out

```
1    1
```

test-constructor2.dice.out

1   510

test-for2.dice

```
1    class test {
2            public void main(char[][] args) {
3              int i;
4              for ( i = 5 ; i > 0 ; i = i - 1 )
5                print(i);
6              print(42);
7            }
8    }
```

test-array2.dice

```
1   class test {
2           public void main(char[][] args) {
3                   float[] a = |1.0,1.5,2.5,3.5,4.5|;
4                   print(a[1]);
5                   print(a[4]);
6           }
7   }
```

test-constructorDefault.dice.out

1   5

test-applicative.dice.out

1 132

test-stmts1.dice.out

1   4562

test-global1.dice

```
1   class test {
2     public int a;
3     public int b;
4
5     public void printa(){
6       print(this.a);
7     }
8
9     public void printb(){
10      print(this.b);
11    }
12
13    public void incab(){
14      this.a = this.a + 1;
15      this.b = this.b + 1;
16    }
17
18    public void main(char[][] args) {
19        this.a = 42;
20        this.b = 21;
21        this.printa();
22        this.printb();
23        this.incab();
24        this.printa();
25        this.printb();
26    }
27  }
```

test-intOverflow.dice

```
1   class test {
2     public void main(char[][] args) {
3         int a = 2147483648; (*More than an int can hold should overflow*)
4         if(a<2147483647){
5                 print("pass");
6         }
7         else{
8         print(a);
9         }
10
11        int b = -2147483649; (*More than an int can hold should overflow*)
12        if(b>-2147483648){
13                print("pass");
14        }
15        else{
16        print(b);
17        }
18      }
19  }
```

test-if6.dice

```
1   class test {
2         public void main(char[][] args) {
3           if (true){
4                   if(true)
5                           print(42);
6                   print(27);
7           }
8           else
9                   print(8);
10
11          if (false){
12                  if(true)
13                          print(42);
14                  print(27);
15          }
16          else
17                  print(8);
18
19          }
20  }
```

test-ifEmptyBlock.dice

```
1   class test {
2         public void main(char[][] args) {
3                 if (true){}
4                 print(17);
5         }
6   }
```

test-arith1.dice.out

1   20

test-arith4.dice.out

1   13105107

test-whileBreak.dice.out

1 543

test-classReturnObjects.dice

```
1   class shape {
2     public int xCoord;
3     public int yCoord;
4
5     constructor (){
6     this.xCoord = 1;
7     this.yCoord = 2;
8     }
9
10  }
11
12  class test {
13    public void main(char[][] args) {
14        class shape a = this.returnMe();
15        print(a.xCoord);
16        print(a.yCoord);
17    }
18
19    public class shape returnMe(){
20      class shape b = new shape();
21      return b;
22    }
23  }
```

test-stdlib-stringclass2.dice.out

1 StringDoesn'tStartWithH

test-intMax.dice.out

1 2147483647
2 -2147483648

test-arith2.dice.out

1   11

test-for1.dice

```
1   class test {
2           public void main(char[][] args) {
3               int i;
4               for (i = 0 ; i < 5 ; i = i + 1) {
5                   print(i);
6               }
7               print(42);
8           }
9   }
```

test-bool3.dice

```
1  class test {
2          public void main(char[][] args) {
3                  print(1>=2);
4                  print(1>=1);
5                  print(1>=2.0);
6                  print(2.0>=2.0);
7          }
8  }
```

test-arithSigned3.dice.out

1   2-22.000000-2.000000

test-arith8.dice.out

```
1   20.000000
2   2.500000
```

test-intMax.dice

```
1  class test {
2    public void main(char[][] args) {
3        int a = 2147483647;
4        int b = -2147483648;
5        print(a);
6        print("\n");
7        print(b);
8      }
9  }
```

test-bool5.dice.out

1   falsetrue

test-args.dice.out

1   davidemilyphil4

test-args.dice

```
1   class test {
2     public void main(char[][] args) {
3         print(args[1]);
4         print(args[2]);
5         print(args[3]);
6         print(args.length);
7     }
8   }
```

E-test-cyclicalIncludesDuplicate.dice.out

1   Exceptions.DuplicateClassName(test)

E-test-objectCreation2.dice.out

1

E-test-scope3.dice

```
1   class test {
2
3           public void main(char[][] args) {
4                   int x;
5                   for(x = 0; x < 3; x = x+1){
6                           int y = 10;
7                           print(y);
8                   }
9                   print(y);
10          }
11  }
```

E-test-objectCreation2.dice

```
1   class Bar {
2   constructor(char c, float f) {}
3   }
4
5   class Foo {
6   constructor(bool b, char c, float f) {}
7   constructor(int a, bool b, char c, float f) {}
8   }
9
10  class test {
11  public void main(char[][] args) {
12  char myc = 'z';
13  float myf = 4.5;
14  class Bar myb = new Bar(myc, myf);
15  class Foo myFooObj = new Foo(5, true, myc, myf);
16  }
17  }
```

E-test-objectAssignMistmatch.dice.out

```
1  LocalAssignTypeMismatch(B,C)
```

E-test-cyclicalIncludes.dice.out

```
1  Exceptions.DuplicateClassName(test)
```

E-test-scope1.dice.out

1   `UndefinedID(x)`

E-test-objectCreation1.dice.out

1

E-test-scope2.dice.out

```
1  UndefinedID(x)
```

E-test-assignMismatch.dice.out

1 AssignmentTypeMismatch (float,int)

E-test-duplicate.dice

```
1  class test {
2  public void main(char[][] args) {
3  char myc = 'z';
4  int myc = 2;
5  float myf = 4.5;
6  }
7  }
```

E-test-scope3.dice.out

```
1  UndefinedID(y)
```

E-test-objectCreation4.dice

```
1   class Bar {
2   constructor(char c, float f) {}
3   constructor(bool b, char c, float f) {}
4   }
5   class Foo {
6   constructor(int a, bool b, char c, float f) {}
7   }
8    class test {
9    public void main(char[][] args) {
10   char myc = 'z';
11   float myf = 4.5;
12   class Bar myb = new Bar(myc, myf);
13   class Foo myFooObj = new Foo(5, true, myc, myf);
14   }
15    }
```

E-test-constructor.dice

```
1  class Foo {
2  constructor(char c, float f) {}
3  constructor(bool b, char c, float f) {}
4  }
5
6  class test {
7  public void main(char[][] args) {
8  int mya = 2;
9  bool myb = false;
10  char myc = 'z';
11  float myf = 3.5;
12  class Foo myFooObj = new Foo(mya, myb, myc, myf);
13  }
14  }
```

E-test-scope2.dice

```
1   class test {
2
3        public void main(char[][] args) {
4                if(true){
5                        int x = 10;
6                        print(x);
7                }
8                print(x);
9        }
10  }
```

E-test-constructor.dice.out

1   ConstructorNotFound: Foo.constructor.int.bool.char.float

E-test-noReturn.dice

```
1    class test {
2
3            public int increment(int x){
4                    x = x+1;
5            }
6            public void main(char[][] args) {
7                    int x = this.increment(5);
8            }
9    }
```

E-test-cyclicalIncludesDuplicate2.dice.out

1 Exceptions.DuplicateClassName(test)

E-test-objectCreation1.dice

```
1   class Bar {
2   constructor(char c, float f) {}
3   constructor(bool b, char c, float f) {}
4   }
5
6   class Foo {
7   constructor(bool a, int b) {}
8   constructor(int a, bool b, char c, float f) {}
9   }
10
11    class test {
12    public void main(char[][] args) {
13    int mya = 2;
14    bool myb = false;
15    char myc = 'z';
16    float myf = 3.5;
17    class Foo myFooObj = new Foo(mya, myb, myc, myf);
18    }
19    }
```

E-test-cyclicalIncludes.dice

```
1   include("Test Suite/Compiler_Test_Suite/test-cyclicalIncludes.dice");
2
3   class test {
4           public void main(char[][] args) {
5           this.output();
6           }
7
8           public void output(){
9                   print("a");
10          }
11  }
```

E-test-undefinedClass2.dice

```
1  class Foo {}
2
3  class Bar {}
4
5  class test {
6  public void main(char[][] args) {
7  class Baz b;
8  }
9  }
```

E-test-mainClassNotDefined.dice

```
1   class test{
2
3   }
```

E-test-privateFieldsAccess.dice

```
1   class shape {
2          private int area;
3
4          constructor(){
5          this.area = 100;
6          }
7
8          public void setArea(int x){
9                  this.area = x;
10         }
11
12         public int getArea(){
13                 return this.area;
14         }
15
16  }
17
18   class test {
19          public void main(char[][] args) {
20                  class shape a = new shape();
21                  a.area = 50;
22
23                  }
24   }
```

E-test-duplicate.dice.out

```
1  DuplicateLocal: myc
```

E-test-stdlib-overload.dice.out

```
1   CannotUseReservedFuncName(print)
```

E-test-noReturn.dice.out

1 `Exceptions.AllNonVoidFunctionsMustEndWithReturn(test.increment)`

E-test-undefinedClass.dice

```
1  class D {
2    public void main(char[][] args) {}
3  }
4  class A extends B {}
5  class B extends C {}
6  class C extends D {}
7  class G extends H {}
8  class I extends H {}
```

E-test-objectAssignMistmatch.dice

```
1  class A {}
2  class B extends A {}
3  class C {}
4  class test {
5  public void main(char[][] args) {
6  class A myBObj = new B();
7   class B mySecondBObj = new C();
8   }
9   }
```

E-test-privateFunctionAccess.dice.out

1 CannotAccessPrivateFunctionInNonProperScope(something.hi,something,test)

E-test-objectCreation3.dice.out

1

E-test-objectCreation3.dice

```
1  class Foo {}
2
3  class Baz {}
4
5  class test {
6  public void main(char[][] args) {
7  class Baz b;
8  }
9  }
```

E-test-privateFieldsAccess.dice.out

```
1  CannotAccessPrivateFieldInNonProperScope(area,shape,test)
```

E-test-assignMismatch2.dice.out

1   AssignmentTypeMismatch (int,float)

E-test-scope1.dice

```
1   class test {
2
3           public void main(char[][] args) {
4                   {
5                           int x = 10;
6                           print(x);
7                   }
8                   print(x);
9           }
10  }
```

E-test-stdlib-overload.dice

```
1   class test {
2
3           public void print(){
4
5           }
6
7     public void main(char[][] args) {
8
9     }
10  }
```

E-test-objectCreation4.dice.out

1

E-test-cyclicalIncludesDuplicate.dice

```
1   include("Test
    ↪   Suite/Compiler_Test_Suite/Exceptions/E-test-cyclicalIncludesDuplicate2.dice");
2
3   class test {
4           public void main(char[][] args) {
5
6           }
7   }
```

E-test-undefinedClass2.dice.out

```
1  UndefinedClass: Baz
```

E-test-cyclicalIncludesDuplicate2.dice

```
1   include("Test
    ↪   Suite/Compiler_Test_Suite/Exceptions/E-test-cyclicalIncludesDuplicate.dice");
2
3   class test {
4
5   }
```

E-test-assignMismatch2.dice

```
1  class test {
2    public void main(char[][] args) {
3        int a;
4        a = 1.0;
5        print(a);
6    }
7  }
```

E-test-privateFunctionAccess.dice

```
1   class shape {
2   }
3
4   class something {
5
6           private void hi(){
7           }
8   }
9
10   class test {
11          public void main(char[][] args) {
12                  class something a = new something();
13                  a.hi();
14                  }
15      }
```

E-test-constructor1.dice

```
1   class shape {
2           public int xCoord;
3           public int yCoord;
4
5           constructor(int x, int y){
6                   xCoord = 0;
7                   yCoord = 0;
8           }
9
10          constructor(int x, int y){
11                  xCoord = x;
12                  yCoord = y;
13          }
14  }
15
16   class test {
17           public void main(char[][] args) {
18                   (* Constructor clash *)
19           }
20   }
```

E-test-assignMismatch.dice

```
1  class test {
2    public void main(char[][] args) {
3        float a;
4        a = 1;
5        print(a);
6    }
7  }
```

E-test-mainClassNotDefined.dice.out

1 MainNotDefined

E-test-undefinedClass.dice.out

```
1  UndefinedClass: H
```

E-test-constructor1.dice.out

1   DuplicateConstructor

test_pretty.dice

```
1  class test  {
2          public void main (char[][] args) {
3                  print("Hello World");
4          }
5  }
```

test.dice

```
1   class test {
2           public void main(char[][] args) {
3                   print("Hello World");
4           }
5   }
```

primitives.dice

```
1   class testPrims {
2           public int a;
3           public float b;
4           private char c;
5           private bool d;
6           public void main(char[][] args) {
7           int e;
8           float f;
9           char g;
10          bool h;
11          a = -2147483648;
12          e =  2147483647;
13          b = 1.0;
14          f = 2.222222;
15          c = '0';
16          g = '\t';
17          d = true;
18          h = false;
19          }
20  }
```

test_pretty.dice.ManualTokens

1 1. CLASS ID(test) LBRACE
2 2. PUBLIC VOID ID(main) LPAREN CHAR LBRACKET RBRACKET LBRACKET RBRACKET ID(args) RPAREN
   ↪ LBRACE
3 3. ID(print) LPAREN STRING_LITERAL(Hello World) RPAREN SEMI
4 4. RBRACE
5 5. RBRACE
6 6. EOF

primitives.dice.ManualTokens

```
1. CLASS ID(testPrims) LBRACE
2. PUBLIC INT ID(a) SEMI
3. PUBLIC FLOAT ID(b) SEMI
4. PRIVATE CHAR ID(c) SEMI
5. PRIVATE BOOL ID(d) SEMI
6. PUBLIC VOID ID(main) LPAREN CHAR LBRACKET RBRACKET LBRACKET RBRACKET ID(args) RPAREN
   LBRACE
7. INT ID(e) SEMI
8. FLOAT ID(f) SEMI
9. CHAR ID(g) SEMI
10. BOOL ID(h) SEMI
11. ID(a) ASSIGN MINUS INT_LITERAL(2147483648) SEMI
12. ID(e) ASSIGN INT_LITERAL(2147483647) SEMI
13. ID(b) ASSIGN FLOAT_LITERAL(1.) SEMI
14. ID(f) ASSIGN FLOAT_LITERAL(2.222222) SEMI
15. ID(c) ASSIGN CHAR_LITERAL(0) SEMI
16. ID(g) ASSIGN CHAR_LITERAL(\t) SEMI
17. ID(d) ASSIGN TRUE SEMI
18. ID(h) ASSIGN FALSE SEMI
19. RBRACE
20. RBRACE EOF
```

test.dice.ManualTokens

1 1. CLASS ID(test) LBRACE
2 2. PUBLIC VOID ID(main) LPAREN CHAR LBRACKET RBRACKET LBRACKET RBRACKET ID(args) RPAREN
   ↪ LBRACE
3 3. ID(print) LPAREN STRING_LITERAL(Hello World) RPAREN SEMI
4 4. RBRACE
5 5. RBRACE EOF

Demo_Animals.dice

```
1   include("stdlib");
2
3   class Animal{
4         public int weight;
5         constructor(){
6               this.weight = 0;
7         }
8
9         constructor(int w){
10              this.weight = w;
11        }
12
13        public void move(){
14              print("Animals move in many ways");
15        }
16  }
17
18  class Bird extends Animal {
19        public int maxFlyingHeight;
20
21        constructor(){
22              this.weight = 0;
23              this.maxFlyingHeight = 0;
24        }
25
26        constructor(int w, int h){
27              this.weight = w;
28              this.maxFlyingHeight = h;
29        }
30
31        public void move(){
32              print("Birds fly!");
33        }
34
35  }
36
37  class Dog extends Animal {
38        public int speed;
39
40        constructor(){
41              this.weight = 0;
42              this.speed = 0;
43        }
44
45        constructor(int w, int s){
46              this.weight = w;
47              this.speed = s;
```

```
48                }
49
50            public void move(){
51                    print("Dogs run!");
52            }
53    }
54
55    class Stephen extends Animal {
56            private bool isDone;
57
58            constructor() {
59                    this.isDone = true;
60            }
61
62            constructor(bool isDone) {
63                    this.isDone = isDone;
64            }
65
66            public void move() {
67                    if(not this.isDone) {
68                            print("I am a techer!");
69                    } else {
70                            print("Also my favorite number is 42");
71                    }
72                    this.isDone = true;
73            }
74
75    }
76
77    class Snake extends Animal {
78            public int slitherSpeed;
79
80            constructor(){
81                    this.weight = 0;
82                    this.slitherSpeed = 0;
83            }
84
85            constructor(int w, int s){
86                    this.weight = w;
87                    this.slitherSpeed = s;
88            }
89
90            public void move(){
91                    print("Snakes slither!");
92            }
93    }
94
95    class Marnie extends Dog {
96            public int cuteness;
```

```
97
98          constructor(){
99                  this.weight = 0;
100                 this.speed = 0;
101         }
102
103         constructor(int w, int s){
104                 this.weight = w;
105                 this.speed = s;
106         }
107
108         constructor(int w, int s, int c){
109                 this.weight = w;
110                 this.speed = s;
111                 this.cuteness = c;
112         }
113
114         public void move(){
115                 class File a = new File("Demo/marnie1.txt", true);
116         char[] buf = a.readfile(4500);
117         a.closefile();
118         print(buf);
119         print("\n");
120         }
121 }
122
123 class test {
124         private bool isDone;
125         public void main(char[][] args) {
126                 this.logo();
127                 this.isDone = false;
128
129                 bool keepGoing = true;
130                 while(keepGoing){
131                         this.animalsToChoose();
132                         char[] buf = input();
133                         print("\n");
134
135                         int choice = this.getInt(buf[0]);
136
137                         if(choice==5)
138                                 break;
139                         else
140                                 this.printMovement(choice);
141
142                         print("\n");
143                 }
144
145         class Marnie a = new Marnie();
```

```
146            a.move();
147            }
148
149        public int getInt(char num){
150                if(num=='1')
151                        return 1;
152                else if(num=='2')
153                        return 2;
154                else if(num=='3')
155                        return 3;
156                else if(num=='4')
157                        return 4;
158                else if(num=='5')
159                        return 5;
160
161            return 0;
162
163        }
164
165        public void printMovement(int choice){
166
167                class Animal b = new Bird();
168                class Animal d = new Dog();
169                class Animal s = new Snake();
170                class Animal stephen = new Stephen(this.isDone);
171
172                if(choice == 1)
173                        b.move();
174                else if(choice == 2)
175                        d.move();
176                else if(choice == 3)
177                        s.move();
178                else if(choice == 4) {
179                        stephen.move();
180                        this.isDone = true;
181                }
182                else
183                        print("Animal not selected!\n");
184
185                print("\n");
186        }
187
188        public void animalsToChoose(){
189                print("1-Bird\n2-Dog\n3-Snake\n4-Stephen\n5-Exit\nPlease choose an animal
                 ↪  or exit(by selecting a number):");
190
191        }
192
193        public void logo(){
```

```
194                 class File a = new File("Demo/logo.txt", true);
195          char[] buf = a.readfile(4500);
196          a.closefile();
197          print(buf);
198
199          int i;
200                 for(i=0;i<3;i=i+1){
201                 print("\n");
202                 }
203
204                 print("Welcome to the animal farm!\n\n");
205          }
206   }
```

References

[1] http://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html *The GNU C Reference Manual.*. N.p., n.d. Web. 26 Oct. 2015.

[2] https://docs.oracle.com/javase/specs/jls/se8/html/index.html *The Java Language Specification.* . N.p., n.d. Web. 26 Oct. 2015.

[3] Edwards, Stephen. *"Programming Language and Translators."* Lecture.

[4] "Control Flow Statements." *The Java Tutorials Learning the Java Language Language Basics* N.p., n.d. Web. 26 Oct. 2015.