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
```

copy the tutorial from https://github.com/DavidWatkins/Dice

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
- **-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[].

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

- A string literal with escape characters
- Defining a base class with one or more fields.

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

To compile the sample code above, type:

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

The output will be:

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

Defining methods

Control Flow

Defining custom classes

Using Inheritance

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.

```
BOOL = "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 *intfd*; 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:

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

```
let char = ''' ( ascii | digit ) '''
let float = (digit+) ['.'] digit+
let int = digit+

| '('       { LPAREN }
| ')'       { RPAREN }
| '{'       { LBRACE }
| '}'       { RBRACE }
| ';'       { SEMI }
| ','       { COMMA }
| '+'       { PLUS }
| '-'       { MINUS }
| '*'       { TIMES }
| '/'       { DIVIDE }
| '%'       { MODULO }
| '='       { ASSIGN }
| "=="      { EQ }
| "!="      { NEQ }
| '<'       { LT }
| "<="      { LEQ }
| ">"       { GT }
| ">="      { GEQ }
| "and"     { AND }
| "or"      { OR }
| "not"     { NOT }
| '.'       { DOT }
| '['       { LBRACKET }
| ']'       { RBRACKET }
| '|'            { BAR }
| "if"      { IF }
| "else"    { ELSE }
| "for"     { FOR }
| "while"   { WHILE }
| "return"  { RETURN }
| "int"     { INT }
| "float"   { FLOAT }
| "bool"    { BOOL }
| "char"    { CHAR }
| "void"    { VOID }
| "null"    { NULL }
| "true"    { TRUE }
| "false"   { FALSE }
| "class"        { CLASS }
| "constructor"  { CONSTRUCTOR }
| "public"       { PUBLIC }
| "private"      { PRIVATE }
| "extends"      { EXTENDS }
| "include"      { INCLUDE }
| "this"         { THIS }
```

```
| "break"                { BREAK }
| "continue"        { CONTINUE }
| "new"                 { NEW }
| "delete"               { DELETE }

| int as lxm                  { INT_LITERAL(int_of_string lxm) }
| float as lxm                { FLOAT_LITERAL(float_of_string lxm) }
| char as lxm                 { CHAR_LITERAL( String.get lxm 1 ) }
| escape_char as lxm{ CHAR_LITERAL( String.get (unescape lxm) 1) }
| string                      { STRING_LITERAL(unescape s) }
| id as lxm                   { ID(lxm) }
| eof                         { EOF }

| (* *) {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.

```
program:
        includes cdecls EOF

includes:
                /* nothing */
        |         include_list

include_list:
                include_decl
        |         include_list include_decl

include_decl:
        INCLUDE LPAREN STRING_LITERAL RPAREN SEMI

cdecls:
        cdecl_list

cdecl_list:
                cdecl
        |         cdecl_list cdecl

cdecl:
                CLASS ID LBRACE cbody RBRACE
        |         CLASS ID EXTENDS ID LBRACE cbody RBRACE

cbody:
                /* nothing */
        |         cbody field
        |         cbody constructor
```

```
        |         cbody fdecl

constructor:
        CONSTRUCTOR LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE

scope:
                PRIVATE
        |         PUBLIC

field:
        scope datatype ID SEMI

fname:
        ID

fdecl:
        scope datatype fname LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE

formals_opt:
                /* nothing */
        |         formal_list

formal_list:
                formal
        |         formal_list COMMA formal

formal:
        datatype ID

actuals_opt:
                /* nothing */
        |         actuals_list

actuals_list:
                expr
        |         actuals_list COMMA expr

primitive:
                INT
        |         FLOAT
        |         CHAR
        |         BOOL
        |         VOID

name:
        CLASS ID

type_tag:
                primitive
```

```
        |       name

array_type:
        type_tag LBRACKET brackets RBRACKET

datatype:
                type_tag
        |         array_type

brackets:
                /* nothing */
        |         brackets RBRACKET LBRACKET

stmt_list:
                /* nothing */
        |         stmt_list stmt

stmt:
        expr SEMI
|         RETURN expr SEMI
|        RETURN SEMI
|         LBRACE stmt_list RBRACE
|         IF LPAREN expr RPAREN stmt
|         IF LPAREN expr RPAREN stmt ELSE stmt
|         FOR LPAREN expr_opt SEMI expr_opt SEMI expr_opt RPAREN stmt
|         WHILE LPAREN expr RPAREN stmt
|        BREAK SEMI
|        CONTINUE SEMI
|   datatype ID SEMI
|         datatype ID ASSIGN expr SEMI

expr_opt:
                /* nothing */
        |         expr

expr:
                literals
        |         expr PLUS   expr
        |         expr MINUS  expr
        |         expr TIMES  expr
        |         expr DIVIDE expr
        |         expr EQ     expr
        |         expr NEQ    expr
        |         expr LT     expr
        |         expr LEQ    expr
        |         expr GT     expr
        |         expr GEQ    expr
        |         expr AND    expr
        |         expr MODULO expr
```

```
        |         NOT  expr
        |         expr OR     expr
        |         expr DOT    expr
        |         expr ASSIGN expr
        |         DELETE expr
        |   MINUS expr
        |         ID LPAREN actuals_opt RPAREN
        |         NEW ID LPAREN actuals_opt RPAREN
        |        NEW type_tag bracket_args RBRACKET
        |         expr bracket_args RBRACKET
        |         LPAREN expr RPAREN

bracket_args:
                LBRACKET expr
        |         bracket_args RBRACKET LBRACKET expr

literals:
                INT_LITERAL
        |         FLOAT_LITERAL
        |         TRUE
        |         FALSE
        |         STRING_LITERAL
        |         CHAR_LITERAL
        |         THIS
        |         ID
        |         NULL
        |         BAR array_prim BAR

array_prim:
                expr
        |        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:

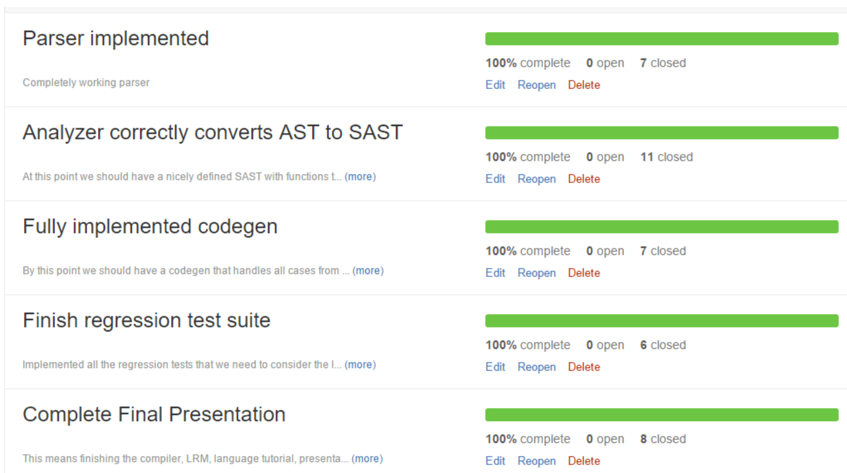

Figure 4.1: Milestoning on Github.

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.

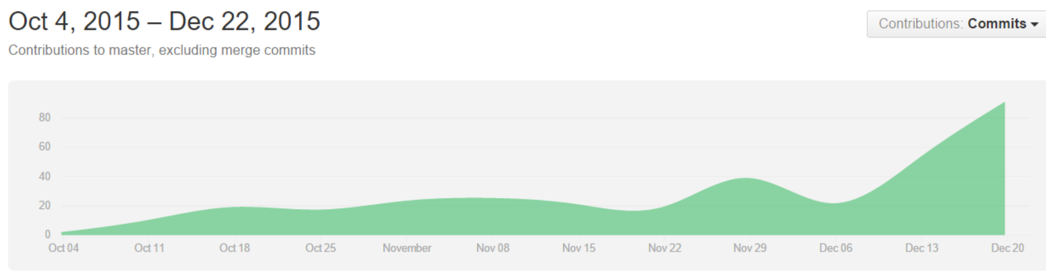

Figure 4.2: Commit timeline on Github.

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 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 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. Along with the code produced from the semantic abstract syntax tree, the code generator also output boiler-plate code for Dice's built-in functions and the virtual table. Definitions for built-in functions are hard-coded in the code generator. The codegen uses those definitions to write the declarations that LLVM needs to call those built in functions from the C standard library. The virtual table is an LLVM function called lookup(). Every time a user function is called in the semantic abstract syntax tree, the code generator instead produces a call to the lookup function, which constructs the virtual table and indexes into it in order to find the desired function. When this is found, the corresponding function pointer is returned.

Once the boiler-plate 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 llvm.moe library, which uses ocaml functions to produce the desired LLVM code.

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

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

The Utilities

Pretty printing, token printing, JSON printing

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

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

test-hello.ll

```
; ModuleID = 'Dice Codegen'
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

%test = type <{ i32 }>

@tmp = private unnamed_addr constant [14 x i8] c"Hello, World!\00"
@tmp.1 = private unnamed_addr constant [3 x i8] c"%s\00"

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

declare noalias i8* @malloc(i32)

declare i32 @open(i8*, i32)

declare i32 @close(i32)

declare i32 @read(i32, i8*, i32)

declare i32 @write(i32, i8*, i32)

declare i32 @lseek(i32, i32, i32)

declare void @exit(i32)

declare i8* @realloc(i8*, i32)

declare i32 @getchar()
```

```

define i64* @lookup(i32 %c_index, i32 %f_index) {
entry:
  %tmp = alloca i64**
  %tmp1 = alloca i64*, i32 0
  %tmp2 = getelementptr i64**, i64*** %tmp, i32 0
  store i64** %tmp1, i64*** %tmp2
  ret i64* null
}

define %test* @test.constructor() {
entry:
  %this = alloca %test
  %tmp = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to
  ↪  i32))
  %tmp1 = bitcast i8* %tmp to %test*
  %tmp2 = load %test, %test* %tmp1
  store %test %tmp2, %test* %this
  %.key = getelementptr inbounds %test, %test* %this, i32 0, i32 0
  store i32 0, i32* %.key
  ret %test* %this
}

define i32 @main(i32 %argc, i8** %argv) {
entry:
  %arr_size = add i32 %argc, 1
  %mallocsize = mul i32 %arr_size, ptrtoint (i1** getelementptr (i1*, i1** null, i32 1)
  ↪  to i32)
  %malloccall = tail call i8* @malloc(i32 %mallocsize)
  %args = bitcast i8* %malloccall to i8***
  %args1 = bitcast i8*** %args to i8**
  %argc_len = bitcast i8** %args1 to i32*
  %arr_1 = getelementptr i8*, i8** %args1, i32 1
  store i32 %argc, i32* %argc_len
  br label %args.cond

args.cond:                                        ; preds = %args.init, %entry
  %counter = phi i32 [ 0, %entry ], [ %tmp, %args.init ]
  %tmp = add i32 %counter, 1
  %tmp2 = icmp slt i32 %counter, %argc
  br i1 %tmp2, label %args.init, label %args.done

args.init:                                        ; preds = %args.cond
  %tmp3 = getelementptr i8*, i8** %arr_1, i32 %counter
  %tmp4 = getelementptr i8*, i8** %argv, i32 %counter
  %tmp5 = load i8*, i8** %tmp4
  store i8* %tmp5, i8** %tmp3
  br label %args.cond

```

```
args.done:                                        ; preds = %args.cond
  %this = alloca %test
  %tmp6 = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32))
  %tmp7 = bitcast i8* %tmp6 to %test*
  %tmp8 = load %test, %test* %tmp7
  store %test %tmp8, %test* %this
  %.key = getelementptr inbounds %test, %test* %this, i32 0, i32 0
  store i32 0, i32* %.key
  %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))
  ret i32 0
}

; Function Attrs: nounwind uwtable
define i8* @input() #0 {
  %initial_size = alloca i32, align 4
  %str = alloca i8*, align 8
  %index = alloca i32, align 4
  %tmp = alloca i8, align 1
  store i32 100, i32* %initial_size, align 4
  %1 = load i32, i32* %initial_size, align 4
  %2 = sext i32 %1 to i64
  %3 = call noalias i8* bitcast (i8* (i32)* @malloc to i8* (i64)*)(i64 %2) #1
  store i8* %3, i8** %str, align 8
  store i32 0, i32* %index, align 4
  store i8 48, i8* %tmp, align 1
  br label %4

; <label>:4                                       ; preds = %20, %0
  %5 = call i32 @getchar()
  %6 = trunc i32 %5 to i8
  store i8 %6, i8* %tmp, align 1
  %7 = sext i8 %6 to i32
  %8 = icmp ne i32 %7, 10
  br i1 %8, label %9, label %27

; <label>:9                                       ; preds = %4
  %10 = load i32, i32* %index, align 4
  %11 = load i32, i32* %initial_size, align 4
  %12 = sub nsw i32 %11, 1
  %13 = icmp sge i32 %10, %12
  br i1 %13, label %14, label %20

; <label>:14                                      ; preds = %9
  %15 = load i8*, i8** %str, align 8
  %16 = load i32, i32* %initial_size, align 4
  %17 = mul nsw i32 %16, 2
```

```
  store i32 %17, i32* %initial_size, align 4
  %18 = sext i32 %17 to i64
  %19 = call i8* bitcast (i8* (i8*, i32)* @realloc to i8* (i8*, i64)*)(i8* %15, i64 %18)
   ↪  #1
  store i8* %19, i8** %str, align 8
  br label %20

; <label>:20                                      ; preds = %14, %9
  %21 = load i8, i8* %tmp, align 1
  %22 = load i32, i32* %index, align 4
  %23 = add nsw i32 %22, 1
  store i32 %23, i32* %index, align 4
  %24 = sext i32 %22 to i64
  %25 = load i8*, i8** %str, align 8
  %26 = getelementptr inbounds i8, i8* %25, i64 %24
  store i8 %21, i8* %26, align 1
  br label %4

; <label>:27                                      ; preds = %4
  %28 = load i32, i32* %index, align 4
  %29 = sext i32 %28 to i64
  %30 = load i8*, i8** %str, align 8
  %31 = getelementptr inbounds i8, i8* %30, i64 %29
  store i8 0, i8* %31, align 1
  %32 = load i8*, i8** %str, align 8
  ret i8* %32
}

; Function Attrs: nounwind uwtable
define void @rec_init(i64* %arr, i32 %curr_offset, i32* %static_offsets, i32* %indexes,
 ↪  i32* %dims, i32 %dimc, i32 %dim_curr) #0 {
  %1 = alloca i64*, align 8
  %2 = alloca i32, align 4
  %3 = alloca i32*, align 8
  %4 = alloca i32*, align 8
  %5 = alloca i32*, align 8
  %6 = alloca i32, align 4
  %7 = alloca i32, align 4
  %static_offset = alloca i32, align 4
  %dynamic_offset = alloca i32, align 4
  %i = alloca i32, align 4
  %tmp = alloca i32, align 4
  %j = alloca i32, align 4
  %i1 = alloca i32, align 4
  %offset = alloca i32, align 4
  %sub = alloca i64*, align 8
  store i64* %arr, i64** %1, align 8
  store i32 %curr_offset, i32* %2, align 4
  store i32* %static_offsets, i32** %3, align 8
```

```
  store i32* %indexes, i32** %4, align 8
  store i32* %dims, i32** %5, align 8
  store i32 %dimc, i32* %6, align 4
  store i32 %dim_curr, i32* %7, align 4
  %8 = load i32, i32* %7, align 4
  %9 = sext i32 %8 to i64
  %10 = load i32*, i32** %5, align 8
  %11 = getelementptr inbounds i32, i32* %10, i64 %9
  %12 = load i32, i32* %11, align 4
  %13 = sext i32 %12 to i64
  %14 = load i32, i32* %2, align 4
  %15 = sext i32 %14 to i64
  %16 = load i64*, i64** %1, align 8
  %17 = getelementptr inbounds i64, i64* %16, i64 %15
  store i64 %13, i64* %17, align 8
  %18 = load i32, i32* %7, align 4
  %19 = add nsw i32 %18, 1
  %20 = load i32, i32* %6, align 4
  %21 = icmp sge i32 %19, %20
  br i1 %21, label %22, label %23

; <label>:22                                      ; preds = %0
  br label %115

; <label>:23                                      ; preds = %0
  %24 = load i32, i32* %7, align 4
  %25 = sext i32 %24 to i64
  %26 = load i32*, i32** %3, align 8
  %27 = getelementptr inbounds i32, i32* %26, i64 %25
  %28 = load i32, i32* %27, align 4
  store i32 %28, i32* %static_offset, align 4
  store i32 0, i32* %dynamic_offset, align 4
  store i32 0, i32* %i, align 4
  br label %29

; <label>:29                                      ; preds = %60, %23
  %30 = load i32, i32* %i, align 4
  %31 = load i32, i32* %7, align 4
  %32 = icmp slt i32 %30, %31
  br i1 %32, label %33, label %63

; <label>:33                                      ; preds = %29
  %34 = load i32, i32* %i, align 4
  %35 = sext i32 %34 to i64
  %36 = load i32*, i32** %4, align 8
  %37 = getelementptr inbounds i32, i32* %36, i64 %35
  %38 = load i32, i32* %37, align 4
  store i32 %38, i32* %tmp, align 4
  %39 = load i32, i32* %i, align 4
```

```
  %40 = add nsw i32 %39, 1
  store i32 %40, i32* %j, align 4
  br label %41

; <label>:41                                      ; preds = %53, %33
  %42 = load i32, i32* %j, align 4
  %43 = load i32, i32* %7, align 4
  %44 = icmp sle i32 %42, %43
  br i1 %44, label %45, label %56

; <label>:45                                      ; preds = %41
  %46 = load i32, i32* %j, align 4
  %47 = sext i32 %46 to i64
  %48 = load i32*, i32** %5, align 8
  %49 = getelementptr inbounds i32, i32* %48, i64 %47
  %50 = load i32, i32* %49, align 4
  %51 = load i32, i32* %tmp, align 4
  %52 = mul nsw i32 %51, %50
  store i32 %52, i32* %tmp, align 4
  br label %53

; <label>:53                                      ; preds = %45
  %54 = load i32, i32* %j, align 4
  %55 = add nsw i32 %54, 1
  store i32 %55, i32* %j, align 4
  br label %41

; <label>:56                                      ; preds = %41
  %57 = load i32, i32* %tmp, align 4
  %58 = load i32, i32* %dynamic_offset, align 4
  %59 = add nsw i32 %58, %57
  store i32 %59, i32* %dynamic_offset, align 4
  br label %60

; <label>:60                                      ; preds = %56
  %61 = load i32, i32* %i, align 4
  %62 = add nsw i32 %61, 1
  store i32 %62, i32* %i, align 4
  br label %29

; <label>:63                                      ; preds = %29
  store i32 0, i32* %i1, align 4
  br label %64

; <label>:64                                      ; preds = %112, %63
  %65 = load i32, i32* %i1, align 4
  %66 = load i32, i32* %7, align 4
  %67 = sext i32 %66 to i64
  %68 = load i32*, i32** %5, align 8
```

```
  %69 = getelementptr inbounds i32, i32* %68, i64 %67
  %70 = load i32, i32* %69, align 4
  %71 = icmp slt i32 %65, %70
  br i1 %71, label %72, label %115

; <label>:72                                      ; preds = %64
  %73 = load i32, i32* %static_offset, align 4
  %74 = load i32, i32* %dynamic_offset, align 4
  %75 = load i32, i32* %i1, align 4
  %76 = add nsw i32 %74, %75
  %77 = load i32, i32* %7, align 4
  %78 = add nsw i32 %77, 1
  %79 = sext i32 %78 to i64
  %80 = load i32*, i32** %5, align 8
  %81 = getelementptr inbounds i32, i32* %80, i64 %79
  %82 = load i32, i32* %81, align 4
  %83 = add nsw i32 %82, 1
  %84 = mul nsw i32 %76, %83
  %85 = add nsw i32 %73, %84
  store i32 %85, i32* %offset, align 4
  %86 = load i64*, i64** %1, align 8
  %87 = load i32, i32* %offset, align 4
  %88 = sext i32 %87 to i64
  %89 = getelementptr inbounds i64, i64* %86, i64 %88
  store i64* %89, i64** %sub, align 8
  %90 = load i64*, i64** %sub, align 8
  %91 = ptrtoint i64* %90 to i64
  %92 = load i32, i32* %2, align 4
  %93 = add nsw i32 %92, 1
  %94 = load i32, i32* %i1, align 4
  %95 = add nsw i32 %93, %94
  %96 = sext i32 %95 to i64
  %97 = load i64*, i64** %1, align 8
  %98 = getelementptr inbounds i64, i64* %97, i64 %96
  store i64 %91, i64* %98, align 8
  %99 = load i32, i32* %i1, align 4
  %100 = load i32, i32* %7, align 4
  %101 = sext i32 %100 to i64
  %102 = load i32*, i32** %4, align 8
  %103 = getelementptr inbounds i32, i32* %102, i64 %101
  store i32 %99, i32* %103, align 4
  %104 = load i64*, i64** %1, align 8
  %105 = load i32, i32* %offset, align 4
  %106 = load i32*, i32** %3, align 8
  %107 = load i32*, i32** %4, align 8
  %108 = load i32*, i32** %5, align 8
  %109 = load i32, i32* %6, align 4
  %110 = load i32, i32* %7, align 4
  %111 = add nsw i32 %110, 1
```

```
  call void @rec_init(i64* %104, i32 %105, i32* %106, i32* %107, i32* %108, i32 %109, i32
   ↪  %111)
  br label %112

; <label>:112                                     ; preds = %72
  %113 = load i32, i32* %i1, align 4
  %114 = add nsw i32 %113, 1
  store i32 %114, i32* %i1, align 4
  br label %64

; <label>:115                                     ; preds = %22, %64
  ret void
}

; Function Attrs: nounwind uwtable
define i64* @init_arr(i32* %dims, i32 %dimc) #0 {
  %1 = alloca i32*, align 8
  %2 = alloca i32, align 4
  %3 = alloca i8*
  %total = alloca i32, align 4
  %i = alloca i32, align 4
  %j = alloca i32, align 4
  %i1 = alloca i32, align 4
  %length = alloca i32, align 4
  %i2 = alloca i32, align 4
  %tmp = alloca i32, align 4
  %j3 = alloca i32, align 4
  %arr = alloca i64*, align 8
  %i4 = alloca i32, align 4
  store i32* %dims, i32** %1, align 8
  store i32 %dimc, i32* %2, align 4
  %4 = load i32, i32* %2, align 4
  %5 = zext i32 %4 to i64
  %6 = call i8* @llvm.stacksave()
  store i8* %6, i8** %3
  %7 = alloca i32, i64 %5, align 16
  store i32 0, i32* %total, align 4
  store i32 0, i32* %i, align 4
  br label %8

; <label>:8                                       ; preds = %56, %0
  %9 = load i32, i32* %i, align 4
  %10 = load i32, i32* %2, align 4
  %11 = icmp slt i32 %9, %10
  br i1 %11, label %12, label %59

; <label>:12                                      ; preds = %8
  %13 = load i32, i32* %i, align 4
  %14 = sext i32 %13 to i64
```

```
  %15 = getelementptr inbounds i32, i32* %7, i64 %14
  store i32 1, i32* %15, align 4
  store i32 0, i32* %j, align 4
  br label %16

; <label>:16                                      ; preds = %31, %12
  %17 = load i32, i32* %j, align 4
  %18 = load i32, i32* %i, align 4
  %19 = icmp slt i32 %17, %18
  br i1 %19, label %20, label %34

; <label>:20                                      ; preds = %16
  %21 = load i32, i32* %j, align 4
  %22 = sext i32 %21 to i64
  %23 = load i32*, i32** %1, align 8
  %24 = getelementptr inbounds i32, i32* %23, i64 %22
  %25 = load i32, i32* %24, align 4
  %26 = load i32, i32* %i, align 4
  %27 = sext i32 %26 to i64
  %28 = getelementptr inbounds i32, i32* %7, i64 %27
  %29 = load i32, i32* %28, align 4
  %30 = mul nsw i32 %29, %25
  store i32 %30, i32* %28, align 4
  br label %31

; <label>:31                                      ; preds = %20
  %32 = load i32, i32* %j, align 4
  %33 = add nsw i32 %32, 1
  store i32 %33, i32* %j, align 4
  br label %16

; <label>:34                                      ; preds = %16
  %35 = load i32, i32* %i, align 4
  %36 = sext i32 %35 to i64
  %37 = load i32*, i32** %1, align 8
  %38 = getelementptr inbounds i32, i32* %37, i64 %36
  %39 = load i32, i32* %38, align 4
  %40 = add nsw i32 %39, 1
  %41 = load i32, i32* %i, align 4
  %42 = sext i32 %41 to i64
  %43 = getelementptr inbounds i32, i32* %7, i64 %42
  %44 = load i32, i32* %43, align 4
  %45 = mul nsw i32 %44, %40
  store i32 %45, i32* %43, align 4
  %46 = load i32, i32* %total, align 4
  %47 = load i32, i32* %i, align 4
  %48 = sext i32 %47 to i64
  %49 = getelementptr inbounds i32, i32* %7, i64 %48
  %50 = load i32, i32* %49, align 4
```

```
  %51 = add nsw i32 %50, %46
  store i32 %51, i32* %49, align 4
  %52 = load i32, i32* %i, align 4
  %53 = sext i32 %52 to i64
  %54 = getelementptr inbounds i32, i32* %7, i64 %53
  %55 = load i32, i32* %54, align 4
  store i32 %55, i32* %total, align 4
  br label %56

; <label>:56                                      ; preds = %34
  %57 = load i32, i32* %i, align 4
  %58 = add nsw i32 %57, 1
  store i32 %58, i32* %i, align 4
  br label %8

; <label>:59                                      ; preds = %8
  %60 = load i32, i32* %2, align 4
  %61 = zext i32 %60 to i64
  %62 = alloca i32, i64 %61, align 16
  store i32 0, i32* %i1, align 4
  br label %63

; <label>:63                                      ; preds = %71, %59
  %64 = load i32, i32* %i1, align 4
  %65 = load i32, i32* %2, align 4
  %66 = icmp slt i32 %64, %65
  br i1 %66, label %67, label %74

; <label>:67                                      ; preds = %63
  %68 = load i32, i32* %i1, align 4
  %69 = sext i32 %68 to i64
  %70 = getelementptr inbounds i32, i32* %62, i64 %69
  store i32 0, i32* %70, align 4
  br label %71

; <label>:71                                      ; preds = %67
  %72 = load i32, i32* %i1, align 4
  %73 = add nsw i32 %72, 1
  store i32 %73, i32* %i1, align 4
  br label %63

; <label>:74                                      ; preds = %63
  store i32 0, i32* %length, align 4
  store i32 0, i32* %i2, align 4
  br label %75

; <label>:75                                      ; preds = %108, %74
  %76 = load i32, i32* %i2, align 4
  %77 = load i32, i32* %2, align 4
```

```
  %78 = icmp slt i32 %76, %77
  br i1 %78, label %79, label %111

; <label>:79                                      ; preds = %75
  store i32 1, i32* %tmp, align 4
  %80 = load i32, i32* %i2, align 4
  %81 = sub nsw i32 %80, 1
  store i32 %81, i32* %j3, align 4
  br label %82

; <label>:82                                      ; preds = %93, %79
  %83 = load i32, i32* %j3, align 4
  %84 = icmp sge i32 %83, 0
  br i1 %84, label %85, label %96

; <label>:85                                      ; preds = %82
  %86 = load i32, i32* %j3, align 4
  %87 = sext i32 %86 to i64
  %88 = load i32*, i32** %1, align 8
  %89 = getelementptr inbounds i32, i32* %88, i64 %87
  %90 = load i32, i32* %89, align 4
  %91 = load i32, i32* %tmp, align 4
  %92 = mul nsw i32 %91, %90
  store i32 %92, i32* %tmp, align 4
  br label %93

; <label>:93                                      ; preds = %85
  %94 = load i32, i32* %j3, align 4
  %95 = add nsw i32 %94, -1
  store i32 %95, i32* %j3, align 4
  br label %82

; <label>:96                                      ; preds = %82
  %97 = load i32, i32* %i2, align 4
  %98 = sext i32 %97 to i64
  %99 = load i32*, i32** %1, align 8
  %100 = getelementptr inbounds i32, i32* %99, i64 %98
  %101 = load i32, i32* %100, align 4
  %102 = add nsw i32 %101, 1
  %103 = load i32, i32* %tmp, align 4
  %104 = mul nsw i32 %103, %102
  store i32 %104, i32* %tmp, align 4
  %105 = load i32, i32* %tmp, align 4
  %106 = load i32, i32* %length, align 4
  %107 = add nsw i32 %106, %105
  store i32 %107, i32* %length, align 4
  br label %108

; <label>:108                                     ; preds = %96
```

```
  %109 = load i32, i32* %i2, align 4
  %110 = add nsw i32 %109, 1
  store i32 %110, i32* %i2, align 4
  br label %75

; <label>:111                                     ; preds = %75
  %112 = load i32, i32* %length, align 4
  %113 = sext i32 %112 to i64
  %114 = call noalias i8* bitcast (i8* (i32)* @malloc to i8* (i64)*)(i64 %113) #1
  %115 = bitcast i8* %114 to i64*
  store i64* %115, i64** %arr, align 8
  store i32 0, i32* %i4, align 4
  br label %116

; <label>:116                                     ; preds = %125, %111
  %117 = load i32, i32* %i4, align 4
  %118 = load i32, i32* %length, align 4
  %119 = icmp slt i32 %117, %118
  br i1 %119, label %120, label %128

; <label>:120                                     ; preds = %116
  %121 = load i32, i32* %i4, align 4
  %122 = sext i32 %121 to i64
  %123 = load i64*, i64** %arr, align 8
  %124 = getelementptr inbounds i64, i64* %123, i64 %122
  store i64 0, i64* %124, align 8
  br label %125

; <label>:125                                     ; preds = %120
  %126 = load i32, i32* %i4, align 4
  %127 = add nsw i32 %126, 1
  store i32 %127, i32* %i4, align 4
  br label %116

; <label>:128                                     ; preds = %116
  %129 = load i64*, i64** %arr, align 8
  %130 = load i32*, i32** %1, align 8
  %131 = load i32, i32* %2, align 4
  call void @rec_init(i64* %129, i32 0, i32* %7, i32* %62, i32* %130, i32 %131, i32 0)
  %132 = load i64*, i64** %arr, align 8
  %133 = load i8*, i8** %3
  call void @llvm.stackrestore(i8* %133)
  ret i64* %132
}

; Function Attrs: nounwind
declare i8* @llvm.stacksave() #1

; Function Attrs: nounwind
```

```
declare void @llvm.stackrestore(i8*) #1

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" }
attributes #1 = { nounwind }

!llvm.ident = !{!0}

!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

```
class shape {
        public float xCoord;
        public float yCoord;
}

class circle extends shape {
        public float radius;
}

class test {
        public void main(char[][] args) {
                class circle a = new circle();
                a.xCoord = 1.5;
                print(a.xCoord);
        }
}
```

test-classExtends.ll

```
        ; ModuleID = 'Dice Codegen'
        target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
        target triple = "x86_64-pc-linux-gnu"

        %test = type <{ i32 }>
        %circle = type <{ i32, double, double, double }>
        %shape = type <{ i32, double, double }>

        @tmp = private unnamed_addr constant [3 x i8] c"%f\00"

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

        declare noalias i8* @malloc(i32)

        declare i32 @open(i8*, i32)

        declare i32 @close(i32)

        declare i32 @read(i32, i8*, i32)

        declare i32 @write(i32, i8*, i32)

        declare i32 @lseek(i32, i32, i32)

        declare void @exit(i32)
```

```

        declare i8* @realloc(i8*, i32)

        declare i32 @getchar()

        define i64* @lookup(i32 %c_index, i32 %f_index) {
        entry:
        %tmp = alloca i64**, i32 3
        %tmp1 = alloca i64*, i32 0
        %tmp2 = getelementptr i64**, i64*** %tmp, i32 2
        store i64** %tmp1, i64*** %tmp2
        %tmp3 = alloca i64*, i32 0
        %tmp4 = getelementptr i64**, i64*** %tmp, i32 1
        store i64** %tmp3, i64*** %tmp4
        %tmp5 = alloca i64*, i32 0
        %tmp6 = getelementptr i64**, i64*** %tmp, i32 0
        store i64** %tmp5, i64*** %tmp6
        ret i64* null
}

define %test* @test.constructor() {
entry:
%this = alloca %test
%tmp = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32))
%tmp1 = bitcast i8* %tmp to %test*
%tmp2 = load %test, %test* %tmp1
store %test %tmp2, %test* %this
%.key = getelementptr inbounds %test, %test* %this, i32 0, i32 0
store i32 2, i32* %.key
ret %test* %this
}

define %circle* @circle.constructor() {
entry:
%this = alloca %circle
%tmp = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32))
%tmp1 = bitcast i8* %tmp to %circle*
%tmp2 = load %circle, %circle* %tmp1
store %circle %tmp2, %circle* %this
%.key = getelementptr inbounds %circle, %circle* %this, i32 0, i32 0
store i32 1, i32* %.key
ret %circle* %this
}

define %shape* @shape.constructor() {
entry:
%this = alloca %shape
%tmp = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32))
%tmp1 = bitcast i8* %tmp to %shape*
```

```
%tmp2 = load %shape, %shape* %tmp1
store %shape %tmp2, %shape* %this
%.key = getelementptr inbounds %shape, %shape* %this, i32 0, i32 0
store i32 0, i32* %.key
ret %shape* %this
}

define i32 @main(i32 %argc, i8** %argv) {
entry:
%arr_size = add i32 %argc, 1
%mallocsize = mul i32 %arr_size, ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32)
%malloccall = tail call i8* @malloc(i32 %mallocsize)
%args = bitcast i8* %malloccall to i8***
%args1 = bitcast i8*** %args to i8**
%argc_len = bitcast i8** %args1 to i32*
%arr_1 = getelementptr i8*, i8** %args1, i32 1
store i32 %argc, i32* %argc_len
br label %args.cond

args.cond:                                        ; preds = %args.init, %entry
%counter = phi i32 [ 0, %entry ], [ %tmp, %args.init ]
%tmp = add i32 %counter, 1
%tmp2 = icmp slt i32 %counter, %argc
br i1 %tmp2, label %args.init, label %args.done

args.init:                                        ; preds = %args.cond
%tmp3 = getelementptr i8*, i8** %arr_1, i32 %counter
%tmp4 = getelementptr i8*, i8** %argv, i32 %counter
%tmp5 = load i8*, i8** %tmp4
store i8* %tmp5, i8** %tmp3
br label %args.cond

args.done:                                        ; preds = %args.cond
%this = alloca %test
%tmp6 = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32))
%tmp7 = bitcast i8* %tmp6 to %test*
%tmp8 = load %test, %test* %tmp7
store %test %tmp8, %test* %this
%.key = getelementptr inbounds %test, %test* %this, i32 0, i32 0
store i32 2, i32* %.key
%a = alloca %circle
%tmp9 = call %circle* @circle.constructor()
%tmp10 = load %circle, %circle* %tmp9
store %circle %tmp10, %circle* %a
%xCoord = getelementptr inbounds %circle, %circle* %a, i32 0, i32 2
store double 1.500000e+00, double* %xCoord
%xCoord11 = getelementptr inbounds %circle, %circle* %a, i32 0, i32 2
```

```
%xCoord12 = load double, double* %xCoord11
%tmp13 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @tmp, i32 0, i32 0), double %xCoord12)
ret i32 0
}

; Function Attrs: nounwind uwtable
define i8* @input() #0 {
        %initial_size = alloca i32, align 4
        %str = alloca i8*, align 8
        %index = alloca i32, align 4
        %tmp = alloca i8, align 1
        store i32 100, i32* %initial_size, align 4
        %1 = load i32, i32* %initial_size, align 4
        %2 = sext i32 %1 to i64
        %3 = call noalias i8* bitcast (i8* (i32)* @malloc to i8* (i64)*)(i64 %2) #1
        store i8* %3, i8** %str, align 8
        store i32 0, i32* %index, align 4
        store i8 48, i8* %tmp, align 1
        br label %4

        ; <label>:4                                         ; preds = %20, %0
        %5 = call i32 @getchar()
        %6 = trunc i32 %5 to i8
        store i8 %6, i8* %tmp, align 1
        %7 = sext i8 %6 to i32
        %8 = icmp ne i32 %7, 10
        br i1 %8, label %9, label %27

        ; <label>:9                                         ; preds = %4
        %10 = load i32, i32* %index, align 4
        %11 = load i32, i32* %initial_size, align 4
        %12 = sub nsw i32 %11, 1
        %13 = icmp sge i32 %10, %12
        br i1 %13, label %14, label %20

        ; <label>:14                                        ; preds = %9
        %15 = load i8*, i8** %str, align 8
        %16 = load i32, i32* %initial_size, align 4
        %17 = mul nsw i32 %16, 2
        store i32 %17, i32* %initial_size, align 4
        %18 = sext i32 %17 to i64
        %19 = call i8* bitcast (i8* (i8*, i32)* @realloc to i8* (i8*, i64)*)(i8* %15, i64 %18) #1
        store i8* %19, i8** %str, align 8
        br label %20

        ; <label>:20                                        ; preds = %14, %9
        %21 = load i8, i8* %tmp, align 1
```

```
        %22 = load i32, i32* %index, align 4
        %23 = add nsw i32 %22, 1
        store i32 %23, i32* %index, align 4
        %24 = sext i32 %22 to i64
        %25 = load i8*, i8** %str, align 8
        %26 = getelementptr inbounds i8, i8* %25, i64 %24
        store i8 %21, i8* %26, align 1
        br label %4

        ; <label>:27                                        ; preds = %4
        %28 = load i32, i32* %index, align 4
        %29 = sext i32 %28 to i64
        %30 = load i8*, i8** %str, align 8
        %31 = getelementptr inbounds i8, i8* %30, i64 %29
        store i8 0, i8* %31, align 1
        %32 = load i8*, i8** %str, align 8
        ret i8* %32
}

; Function Attrs: nounwind uwtable
define void @rec_init(i64* %arr, i32 %curr_offset, i32* %static_offsets, i32* %indexes,
 ↪  i32* %dims, i32 %dimc, i32 %dim_curr) #0 {
%1 = alloca i64*, align 8
%2 = alloca i32, align 4
%3 = alloca i32*, align 8
%4 = alloca i32*, align 8
%5 = alloca i32*, align 8
%6 = alloca i32, align 4
%7 = alloca i32, align 4
%static_offset = alloca i32, align 4
%dynamic_offset = alloca i32, align 4
%i = alloca i32, align 4
%tmp = alloca i32, align 4
%j = alloca i32, align 4
%i1 = alloca i32, align 4
%offset = alloca i32, align 4
%sub = alloca i64*, align 8
store i64* %arr, i64** %1, align 8
store i32 %curr_offset, i32* %2, align 4
store i32* %static_offsets, i32** %3, align 8
store i32* %indexes, i32** %4, align 8
store i32* %dims, i32** %5, align 8
store i32 %dimc, i32* %6, align 4
store i32 %dim_curr, i32* %7, align 4
%8 = load i32, i32* %7, align 4
%9 = sext i32 %8 to i64
%10 = load i32*, i32** %5, align 8
%11 = getelementptr inbounds i32, i32* %10, i64 %9
%12 = load i32, i32* %11, align 4
```

```
%13 = sext i32 %12 to i64
%14 = load i32, i32* %2, align 4
%15 = sext i32 %14 to i64
%16 = load i64*, i64** %1, align 8
%17 = getelementptr inbounds i64, i64* %16, i64 %15
store i64 %13, i64* %17, align 8
%18 = load i32, i32* %7, align 4
%19 = add nsw i32 %18, 1
%20 = load i32, i32* %6, align 4
%21 = icmp sge i32 %19, %20
br i1 %21, label %22, label %23

; <label>:22                                      ; preds = %0
br label %115

; <label>:23                                      ; preds = %0
%24 = load i32, i32* %7, align 4
%25 = sext i32 %24 to i64
%26 = load i32*, i32** %3, align 8
%27 = getelementptr inbounds i32, i32* %26, i64 %25
%28 = load i32, i32* %27, align 4
store i32 %28, i32* %static_offset, align 4
store i32 0, i32* %dynamic_offset, align 4
store i32 0, i32* %i, align 4
br label %29

; <label>:29                                      ; preds = %60, %23
%30 = load i32, i32* %i, align 4
%31 = load i32, i32* %7, align 4
%32 = icmp slt i32 %30, %31
br i1 %32, label %33, label %63

; <label>:33                                      ; preds = %29
%34 = load i32, i32* %i, align 4
%35 = sext i32 %34 to i64
%36 = load i32*, i32** %4, align 8
%37 = getelementptr inbounds i32, i32* %36, i64 %35
%38 = load i32, i32* %37, align 4
store i32 %38, i32* %tmp, align 4
%39 = load i32, i32* %i, align 4
%40 = add nsw i32 %39, 1
store i32 %40, i32* %j, align 4
br label %41

; <label>:41                                      ; preds = %53, %33
%42 = load i32, i32* %j, align 4
%43 = load i32, i32* %7, align 4
%44 = icmp sle i32 %42, %43
br i1 %44, label %45, label %56
```

```
; <label>:45                                        ; preds = %41
%46 = load i32, i32* %j, align 4
%47 = sext i32 %46 to i64
%48 = load i32*, i32** %5, align 8
%49 = getelementptr inbounds i32, i32* %48, i64 %47
%50 = load i32, i32* %49, align 4
%51 = load i32, i32* %tmp, align 4
%52 = mul nsw i32 %51, %50
store i32 %52, i32* %tmp, align 4
br label %53

; <label>:53                                        ; preds = %45
%54 = load i32, i32* %j, align 4
%55 = add nsw i32 %54, 1
store i32 %55, i32* %j, align 4
br label %41

; <label>:56                                        ; preds = %41
%57 = load i32, i32* %tmp, align 4
%58 = load i32, i32* %dynamic_offset, align 4
%59 = add nsw i32 %58, %57
store i32 %59, i32* %dynamic_offset, align 4
br label %60

; <label>:60                                        ; preds = %56
%61 = load i32, i32* %i, align 4
%62 = add nsw i32 %61, 1
store i32 %62, i32* %i, align 4
br label %29

; <label>:63                                        ; preds = %29
store i32 0, i32* %i1, align 4
br label %64

; <label>:64                                        ; preds = %112, %63
%65 = load i32, i32* %i1, align 4
%66 = load i32, i32* %7, align 4
%67 = sext i32 %66 to i64
%68 = load i32*, i32** %5, align 8
%69 = getelementptr inbounds i32, i32* %68, i64 %67
%70 = load i32, i32* %69, align 4
%71 = icmp slt i32 %65, %70
br i1 %71, label %72, label %115

; <label>:72                                        ; preds = %64
%73 = load i32, i32* %static_offset, align 4
%74 = load i32, i32* %dynamic_offset, align 4
%75 = load i32, i32* %i1, align 4
```

```
%76 = add nsw i32 %74, %75
%77 = load i32, i32* %7, align 4
%78 = add nsw i32 %77, 1
%79 = sext i32 %78 to i64
%80 = load i32*, i32** %5, align 8
%81 = getelementptr inbounds i32, i32* %80, i64 %79
%82 = load i32, i32* %81, align 4
%83 = add nsw i32 %82, 1
%84 = mul nsw i32 %76, %83
%85 = add nsw i32 %73, %84
store i32 %85, i32* %offset, align 4
%86 = load i64*, i64** %1, align 8
%87 = load i32, i32* %offset, align 4
%88 = sext i32 %87 to i64
%89 = getelementptr inbounds i64, i64* %86, i64 %88
store i64* %89, i64** %sub, align 8
%90 = load i64*, i64** %sub, align 8
%91 = ptrtoint i64* %90 to i64
%92 = load i32, i32* %2, align 4
%93 = add nsw i32 %92, 1
%94 = load i32, i32* %i1, align 4
%95 = add nsw i32 %93, %94
%96 = sext i32 %95 to i64
%97 = load i64*, i64** %1, align 8
%98 = getelementptr inbounds i64, i64* %97, i64 %96
store i64 %91, i64* %98, align 8
%99 = load i32, i32* %i1, align 4
%100 = load i32, i32* %7, align 4
%101 = sext i32 %100 to i64
%102 = load i32*, i32** %4, align 8
%103 = getelementptr inbounds i32, i32* %102, i64 %101
store i32 %99, i32* %103, align 4
%104 = load i64*, i64** %1, align 8
%105 = load i32, i32* %offset, align 4
%106 = load i32*, i32** %3, align 8
%107 = load i32*, i32** %4, align 8
%108 = load i32*, i32** %5, align 8
%109 = load i32, i32* %6, align 4
%110 = load i32, i32* %7, align 4
%111 = add nsw i32 %110, 1
call void @rec_init(i64* %104, i32 %105, i32* %106, i32* %107, i32* %108, i32 %109, i32
 ↪ %111)
br label %112

; <label>:112                                      ; preds = %72
%113 = load i32, i32* %i1, align 4
%114 = add nsw i32 %113, 1
store i32 %114, i32* %i1, align 4
br label %64
```

```

; <label>:115                                      ; preds = %22, %64
ret void
}

; Function Attrs: nounwind uwtable
define i64* @init_arr(i32* %dims, i32 %dimc) #0 {
%1 = alloca i32*, align 8
%2 = alloca i32, align 4
%3 = alloca i8*
%total = alloca i32, align 4
%i = alloca i32, align 4
%j = alloca i32, align 4
%i1 = alloca i32, align 4
%length = alloca i32, align 4
%i2 = alloca i32, align 4
%tmp = alloca i32, align 4
%j3 = alloca i32, align 4
%arr = alloca i64*, align 8
%i4 = alloca i32, align 4
store i32* %dims, i32** %1, align 8
store i32 %dimc, i32* %2, align 4
%4 = load i32, i32* %2, align 4
%5 = zext i32 %4 to i64
%6 = call i8* @llvm.stacksave()
store i8* %6, i8** %3
%7 = alloca i32, i64 %5, align 16
store i32 0, i32* %total, align 4
store i32 0, i32* %i, align 4
br label %8

; <label>:8                                        ; preds = %56, %0
%9 = load i32, i32* %i, align 4
%10 = load i32, i32* %2, align 4
%11 = icmp slt i32 %9, %10
br i1 %11, label %12, label %59

; <label>:12                                       ; preds = %8
%13 = load i32, i32* %i, align 4
%14 = sext i32 %13 to i64
%15 = getelementptr inbounds i32, i32* %7, i64 %14
store i32 1, i32* %15, align 4
store i32 0, i32* %j, align 4
br label %16

; <label>:16                                       ; preds = %31, %12
%17 = load i32, i32* %j, align 4
%18 = load i32, i32* %i, align 4
%19 = icmp slt i32 %17, %18
```

```
br i1 %19, label %20, label %34

; <label>:20                                      ; preds = %16
%21 = load i32, i32* %j, align 4
%22 = sext i32 %21 to i64
%23 = load i32*, i32** %1, align 8
%24 = getelementptr inbounds i32, i32* %23, i64 %22
%25 = load i32, i32* %24, align 4
%26 = load i32, i32* %i, align 4
%27 = sext i32 %26 to i64
%28 = getelementptr inbounds i32, i32* %7, i64 %27
%29 = load i32, i32* %28, align 4
%30 = mul nsw i32 %29, %25
store i32 %30, i32* %28, align 4
br label %31

; <label>:31                                      ; preds = %20
%32 = load i32, i32* %j, align 4
%33 = add nsw i32 %32, 1
store i32 %33, i32* %j, align 4
br label %16

; <label>:34                                      ; preds = %16
%35 = load i32, i32* %i, align 4
%36 = sext i32 %35 to i64
%37 = load i32*, i32** %1, align 8
%38 = getelementptr inbounds i32, i32* %37, i64 %36
%39 = load i32, i32* %38, align 4
%40 = add nsw i32 %39, 1
%41 = load i32, i32* %i, align 4
%42 = sext i32 %41 to i64
%43 = getelementptr inbounds i32, i32* %7, i64 %42
%44 = load i32, i32* %43, align 4
%45 = mul nsw i32 %44, %40
store i32 %45, i32* %43, align 4
%46 = load i32, i32* %total, align 4
%47 = load i32, i32* %i, align 4
%48 = sext i32 %47 to i64
%49 = getelementptr inbounds i32, i32* %7, i64 %48
%50 = load i32, i32* %49, align 4
%51 = add nsw i32 %50, %46
store i32 %51, i32* %49, align 4
%52 = load i32, i32* %i, align 4
%53 = sext i32 %52 to i64
%54 = getelementptr inbounds i32, i32* %7, i64 %53
%55 = load i32, i32* %54, align 4
store i32 %55, i32* %total, align 4
br label %56
```

```
; <label>:56                                        ; preds = %34
%57 = load i32, i32* %i, align 4
%58 = add nsw i32 %57, 1
store i32 %58, i32* %i, align 4
br label %8

; <label>:59                                        ; preds = %8
%60 = load i32, i32* %2, align 4
%61 = zext i32 %60 to i64
%62 = alloca i32, i64 %61, align 16
store i32 0, i32* %i1, align 4
br label %63

; <label>:63                                        ; preds = %71, %59
%64 = load i32, i32* %i1, align 4
%65 = load i32, i32* %2, align 4
%66 = icmp slt i32 %64, %65
br i1 %66, label %67, label %74

; <label>:67                                        ; preds = %63
%68 = load i32, i32* %i1, align 4
%69 = sext i32 %68 to i64
%70 = getelementptr inbounds i32, i32* %62, i64 %69
store i32 0, i32* %70, align 4
br label %71

; <label>:71                                        ; preds = %67
%72 = load i32, i32* %i1, align 4
%73 = add nsw i32 %72, 1
store i32 %73, i32* %i1, align 4
br label %63

; <label>:74                                        ; preds = %63
store i32 0, i32* %length, align 4
store i32 0, i32* %i2, align 4
br label %75

; <label>:75                                        ; preds = %108, %74
%76 = load i32, i32* %i2, align 4
%77 = load i32, i32* %2, align 4
%78 = icmp slt i32 %76, %77
br i1 %78, label %79, label %111

; <label>:79                                        ; preds = %75
store i32 1, i32* %tmp, align 4
%80 = load i32, i32* %i2, align 4
%81 = sub nsw i32 %80, 1
store i32 %81, i32* %j3, align 4
br label %82
```

```
; <label>:82                                      ; preds = %93, %79
%83 = load i32, i32* %j3, align 4
%84 = icmp sge i32 %83, 0
br i1 %84, label %85, label %96

; <label>:85                                      ; preds = %82
%86 = load i32, i32* %j3, align 4
%87 = sext i32 %86 to i64
%88 = load i32*, i32** %1, align 8
%89 = getelementptr inbounds i32, i32* %88, i64 %87
%90 = load i32, i32* %89, align 4
%91 = load i32, i32* %tmp, align 4
%92 = mul nsw i32 %91, %90
store i32 %92, i32* %tmp, align 4
br label %93

; <label>:93                                      ; preds = %85
%94 = load i32, i32* %j3, align 4
%95 = add nsw i32 %94, -1
store i32 %95, i32* %j3, align 4
br label %82

; <label>:96                                      ; preds = %82
%97 = load i32, i32* %i2, align 4
%98 = sext i32 %97 to i64
%99 = load i32*, i32** %1, align 8
%100 = getelementptr inbounds i32, i32* %99, i64 %98
%101 = load i32, i32* %100, align 4
%102 = add nsw i32 %101, 1
%103 = load i32, i32* %tmp, align 4
%104 = mul nsw i32 %103, %102
store i32 %104, i32* %tmp, align 4
%105 = load i32, i32* %tmp, align 4
%106 = load i32, i32* %length, align 4
%107 = add nsw i32 %106, %105
store i32 %107, i32* %length, align 4
br label %108

; <label>:108                                     ; preds = %96
%109 = load i32, i32* %i2, align 4
%110 = add nsw i32 %109, 1
store i32 %110, i32* %i2, align 4
br label %75

; <label>:111                                     ; preds = %75
%112 = load i32, i32* %length, align 4
%113 = sext i32 %112 to i64
%114 = call noalias i8* bitcast (i8* (i32)* @malloc to i8* (i64)*)(i64 %113) #1
```

```
%115 = bitcast i8* %114 to i64*
store i64* %115, i64** %arr, align 8
store i32 0, i32* %i4, align 4
br label %116

; <label>:116                                       ; preds = %125, %111
%117 = load i32, i32* %i4, align 4
%118 = load i32, i32* %length, align 4
%119 = icmp slt i32 %117, %118
br i1 %119, label %120, label %128

; <label>:120                                       ; preds = %116
%121 = load i32, i32* %i4, align 4
%122 = sext i32 %121 to i64
%123 = load i64*, i64** %arr, align 8
%124 = getelementptr inbounds i64, i64* %123, i64 %122
store i64 0, i64* %124, align 8
br label %125

; <label>:125                                       ; preds = %120
%126 = load i32, i32* %i4, align 4
%127 = add nsw i32 %126, 1
store i32 %127, i32* %i4, align 4
br label %116

; <label>:128                                       ; preds = %116
%129 = load i64*, i64** %arr, align 8
%130 = load i32*, i32** %1, align 8
%131 = load i32, i32* %2, align 4
call void @rec_init(i64* %129, i32 0, i32* %7, i32* %62, i32* %130, i32 %131, i32 0)
%132 = load i64*, i64** %arr, align 8
%133 = load i8*, i8** %3
call void @llvm.stackrestore(i8* %133)
ret i64* %132
}

; Function Attrs: nounwind
declare i8* @llvm.stacksave() #1

; Function Attrs: nounwind
declare void @llvm.stackrestore(i8*) #1

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" }
attributes #1 = { nounwind }
```

```
!llvm.ident = !{!0}

!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

```
class test {
	public void main(char[][] args) {
		int i;
		for (i = 0 ; i < 5 ; i = i + 1) {
			print(i);
		}
		print(42);
	}
}
```

test-for1.ll

```
; ModuleID = 'Dice Codegen'
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

%test = type <{ i32 }>

@tmp = private unnamed_addr constant [3 x i8] c"%d\00"
@tmp.1 = private unnamed_addr constant [3 x i8] c"%d\00"

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

declare noalias i8* @malloc(i32)

declare i32 @open(i8*, i32)

declare i32 @close(i32)

declare i32 @read(i32, i8*, i32)

declare i32 @write(i32, i8*, i32)

declare i32 @lseek(i32, i32, i32)

declare void @exit(i32)

declare i8* @realloc(i8*, i32)

declare i32 @getchar()

define i64* @lookup(i32 %c_index, i32 %f_index) {
entry:
```

```
        %tmp = alloca i64**
        %tmp1 = alloca i64*, i32 0
        %tmp2 = getelementptr i64**, i64*** %tmp, i32 0
        store i64** %tmp1, i64*** %tmp2
        ret i64* null
}

define %test* @test.constructor() {
entry:
%this = alloca %test
%tmp = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32))
%tmp1 = bitcast i8* %tmp to %test*
%tmp2 = load %test, %test* %tmp1
store %test %tmp2, %test* %this
%.key = getelementptr inbounds %test, %test* %this, i32 0, i32 0
store i32 0, i32* %.key
ret %test* %this
}

define i32 @main(i32 %argc, i8** %argv) {
entry:
%arr_size = add i32 %argc, 1
%mallocsize = mul i32 %arr_size, ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32)
%malloccall = tail call i8* @malloc(i32 %mallocsize)
%args = bitcast i8* %malloccall to i8***
%args1 = bitcast i8*** %args to i8**
%argc_len = bitcast i8** %args1 to i32*
%arr_1 = getelementptr i8*, i8** %args1, i32 1
store i32 %argc, i32* %argc_len
br label %args.cond

args.cond:                                        ; preds = %args.init, %entry
%counter = phi i32 [ 0, %entry ], [ %tmp, %args.init ]
%tmp = add i32 %counter, 1
%tmp2 = icmp slt i32 %counter, %argc
br i1 %tmp2, label %args.init, label %args.done

args.init:                                        ; preds = %args.cond
%tmp3 = getelementptr i8*, i8** %arr_1, i32 %counter
%tmp4 = getelementptr i8*, i8** %argv, i32 %counter
%tmp5 = load i8*, i8** %tmp4
store i8* %tmp5, i8** %tmp3
br label %args.cond

args.done:                                        ; preds = %args.cond
%this = alloca %test
%tmp6 = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32))
```

```
%tmp7 = bitcast i8* %tmp6 to %test*
%tmp8 = load %test, %test* %tmp7
store %test %tmp8, %test* %this
%.key = getelementptr inbounds %test, %test* %this, i32 0, i32 0
store i32 0, i32* %.key
%i = alloca i32
store i32 0, i32* %i
br label %cond

loop:                                             ; preds = %cond
%i9 = load i32, i32* %i
%tmp10 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @tmp, i32 0, i32 0), i32 %i9)
br label %inc

inc:                                              ; preds = %loop
%i11 = load i32, i32* %i
%addtmp = add i32 %i11, 1
store i32 %addtmp, i32* %i
br label %cond

cond:                                             ; preds = %inc, %args.done
%i12 = load i32, i32* %i
%lesstmp = icmp slt i32 %i12, 5
br i1 %lesstmp, label %loop, label %afterloop

afterloop:                                        ; preds = %cond
%tmp13 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @tmp.1, i32 0, i32 0), i32 42)
ret i32 0
}

; Function Attrs: nounwind uwtable
define i8* @input() #0 {
        %initial_size = alloca i32, align 4
        %str = alloca i8*, align 8
        %index = alloca i32, align 4
        %tmp = alloca i8, align 1
        store i32 100, i32* %initial_size, align 4
        %1 = load i32, i32* %initial_size, align 4
        %2 = sext i32 %1 to i64
        %3 = call noalias i8* bitcast (i8* (i32)* @malloc to i8* (i64)*)(i64 %2) #1
        store i8* %3, i8** %str, align 8
        store i32 0, i32* %index, align 4
        store i8 48, i8* %tmp, align 1
        br label %4

        ; <label>:4                                       ; preds = %20, %0
        %5 = call i32 @getchar()
```

```
        %6 = trunc i32 %5 to i8
        store i8 %6, i8* %tmp, align 1
        %7 = sext i8 %6 to i32
        %8 = icmp ne i32 %7, 10
        br i1 %8, label %9, label %27

        ; <label>:9                                          ; preds = %4
        %10 = load i32, i32* %index, align 4
        %11 = load i32, i32* %initial_size, align 4
        %12 = sub nsw i32 %11, 1
        %13 = icmp sge i32 %10, %12
        br i1 %13, label %14, label %20

        ; <label>:14                                         ; preds = %9
        %15 = load i8*, i8** %str, align 8
        %16 = load i32, i32* %initial_size, align 4
        %17 = mul nsw i32 %16, 2
        store i32 %17, i32* %initial_size, align 4
        %18 = sext i32 %17 to i64
        %19 = call i8* bitcast (i8* (i8*, i32)* @realloc to i8* (i8*, i64)*)(i8* %15, i64 %18) #1
        store i8* %19, i8** %str, align 8
        br label %20

        ; <label>:20                                         ; preds = %14, %9
        %21 = load i8, i8* %tmp, align 1
        %22 = load i32, i32* %index, align 4
        %23 = add nsw i32 %22, 1
        store i32 %23, i32* %index, align 4
        %24 = sext i32 %22 to i64
        %25 = load i8*, i8** %str, align 8
        %26 = getelementptr inbounds i8, i8* %25, i64 %24
        store i8 %21, i8* %26, align 1
        br label %4

        ; <label>:27                                         ; preds = %4
        %28 = load i32, i32* %index, align 4
        %29 = sext i32 %28 to i64
        %30 = load i8*, i8** %str, align 8
        %31 = getelementptr inbounds i8, i8* %30, i64 %29
        store i8 0, i8* %31, align 1
        %32 = load i8*, i8** %str, align 8
        ret i8* %32
}

; Function Attrs: nounwind uwtable
define void @rec_init(i64* %arr, i32 %curr_offset, i32* %static_offsets, i32* %indexes, i32* %dims, i32 %dimc, i32 %dim_curr) #0 {
%1 = alloca i64*, align 8
```

```
%2 = alloca i32, align 4
%3 = alloca i32*, align 8
%4 = alloca i32*, align 8
%5 = alloca i32*, align 8
%6 = alloca i32, align 4
%7 = alloca i32, align 4
%static_offset = alloca i32, align 4
%dynamic_offset = alloca i32, align 4
%i = alloca i32, align 4
%tmp = alloca i32, align 4
%j = alloca i32, align 4
%i1 = alloca i32, align 4
%offset = alloca i32, align 4
%sub = alloca i64*, align 8
store i64* %arr, i64** %1, align 8
store i32 %curr_offset, i32* %2, align 4
store i32* %static_offsets, i32** %3, align 8
store i32* %indexes, i32** %4, align 8
store i32* %dims, i32** %5, align 8
store i32 %dimc, i32* %6, align 4
store i32 %dim_curr, i32* %7, align 4
%8 = load i32, i32* %7, align 4
%9 = sext i32 %8 to i64
%10 = load i32*, i32** %5, align 8
%11 = getelementptr inbounds i32, i32* %10, i64 %9
%12 = load i32, i32* %11, align 4
%13 = sext i32 %12 to i64
%14 = load i32, i32* %2, align 4
%15 = sext i32 %14 to i64
%16 = load i64*, i64** %1, align 8
%17 = getelementptr inbounds i64, i64* %16, i64 %15
store i64 %13, i64* %17, align 8
%18 = load i32, i32* %7, align 4
%19 = add nsw i32 %18, 1
%20 = load i32, i32* %6, align 4
%21 = icmp sge i32 %19, %20
br i1 %21, label %22, label %23

; <label>:22                                      ; preds = %0
br label %115

; <label>:23                                      ; preds = %0
%24 = load i32, i32* %7, align 4
%25 = sext i32 %24 to i64
%26 = load i32*, i32** %3, align 8
%27 = getelementptr inbounds i32, i32* %26, i64 %25
%28 = load i32, i32* %27, align 4
store i32 %28, i32* %static_offset, align 4
store i32 0, i32* %dynamic_offset, align 4
```

```
store i32 0, i32* %i, align 4
br label %29

; <label>:29                                      ; preds = %60, %23
%30 = load i32, i32* %i, align 4
%31 = load i32, i32* %7, align 4
%32 = icmp slt i32 %30, %31
br i1 %32, label %33, label %63

; <label>:33                                      ; preds = %29
%34 = load i32, i32* %i, align 4
%35 = sext i32 %34 to i64
%36 = load i32*, i32** %4, align 8
%37 = getelementptr inbounds i32, i32* %36, i64 %35
%38 = load i32, i32* %37, align 4
store i32 %38, i32* %tmp, align 4
%39 = load i32, i32* %i, align 4
%40 = add nsw i32 %39, 1
store i32 %40, i32* %j, align 4
br label %41

; <label>:41                                      ; preds = %53, %33
%42 = load i32, i32* %j, align 4
%43 = load i32, i32* %7, align 4
%44 = icmp sle i32 %42, %43
br i1 %44, label %45, label %56

; <label>:45                                      ; preds = %41
%46 = load i32, i32* %j, align 4
%47 = sext i32 %46 to i64
%48 = load i32*, i32** %5, align 8
%49 = getelementptr inbounds i32, i32* %48, i64 %47
%50 = load i32, i32* %49, align 4
%51 = load i32, i32* %tmp, align 4
%52 = mul nsw i32 %51, %50
store i32 %52, i32* %tmp, align 4
br label %53

; <label>:53                                      ; preds = %45
%54 = load i32, i32* %j, align 4
%55 = add nsw i32 %54, 1
store i32 %55, i32* %j, align 4
br label %41

; <label>:56                                      ; preds = %41
%57 = load i32, i32* %tmp, align 4
%58 = load i32, i32* %dynamic_offset, align 4
%59 = add nsw i32 %58, %57
store i32 %59, i32* %dynamic_offset, align 4
```

```
br label %60

; <label>:60                                      ; preds = %56
%61 = load i32, i32* %i, align 4
%62 = add nsw i32 %61, 1
store i32 %62, i32* %i, align 4
br label %29

; <label>:63                                      ; preds = %29
store i32 0, i32* %i1, align 4
br label %64

; <label>:64                                      ; preds = %112, %63
%65 = load i32, i32* %i1, align 4
%66 = load i32, i32* %7, align 4
%67 = sext i32 %66 to i64
%68 = load i32*, i32** %5, align 8
%69 = getelementptr inbounds i32, i32* %68, i64 %67
%70 = load i32, i32* %69, align 4
%71 = icmp slt i32 %65, %70
br i1 %71, label %72, label %115

; <label>:72                                      ; preds = %64
%73 = load i32, i32* %static_offset, align 4
%74 = load i32, i32* %dynamic_offset, align 4
%75 = load i32, i32* %i1, align 4
%76 = add nsw i32 %74, %75
%77 = load i32, i32* %7, align 4
%78 = add nsw i32 %77, 1
%79 = sext i32 %78 to i64
%80 = load i32*, i32** %5, align 8
%81 = getelementptr inbounds i32, i32* %80, i64 %79
%82 = load i32, i32* %81, align 4
%83 = add nsw i32 %82, 1
%84 = mul nsw i32 %76, %83
%85 = add nsw i32 %73, %84
store i32 %85, i32* %offset, align 4
%86 = load i64*, i64** %1, align 8
%87 = load i32, i32* %offset, align 4
%88 = sext i32 %87 to i64
%89 = getelementptr inbounds i64, i64* %86, i64 %88
store i64* %89, i64** %sub, align 8
%90 = load i64*, i64** %sub, align 8
%91 = ptrtoint i64* %90 to i64
%92 = load i32, i32* %2, align 4
%93 = add nsw i32 %92, 1
%94 = load i32, i32* %i1, align 4
%95 = add nsw i32 %93, %94
%96 = sext i32 %95 to i64
```

```
%97 = load i64*, i64** %1, align 8
%98 = getelementptr inbounds i64, i64* %97, i64 %96
store i64 %91, i64* %98, align 8
%99 = load i32, i32* %i1, align 4
%100 = load i32, i32* %7, align 4
%101 = sext i32 %100 to i64
%102 = load i32*, i32** %4, align 8
%103 = getelementptr inbounds i32, i32* %102, i64 %101
store i32 %99, i32* %103, align 4
%104 = load i64*, i64** %1, align 8
%105 = load i32, i32* %offset, align 4
%106 = load i32*, i32** %3, align 8
%107 = load i32*, i32** %4, align 8
%108 = load i32*, i32** %5, align 8
%109 = load i32, i32* %6, align 4
%110 = load i32, i32* %7, align 4
%111 = add nsw i32 %110, 1
call void @rec_init(i64* %104, i32 %105, i32* %106, i32* %107, i32* %108, i32 %109, i32
↪  %111)
br label %112

; <label>:112                                     ; preds = %72
%113 = load i32, i32* %i1, align 4
%114 = add nsw i32 %113, 1
store i32 %114, i32* %i1, align 4
br label %64

; <label>:115                                     ; preds = %22, %64
ret void
}

; Function Attrs: nounwind uwtable
define i64* @init_arr(i32* %dims, i32 %dimc) #0 {
%1 = alloca i32*, align 8
%2 = alloca i32, align 4
%3 = alloca i8*
%total = alloca i32, align 4
%i = alloca i32, align 4
%j = alloca i32, align 4
%i1 = alloca i32, align 4
%length = alloca i32, align 4
%i2 = alloca i32, align 4
%tmp = alloca i32, align 4
%j3 = alloca i32, align 4
%arr = alloca i64*, align 8
%i4 = alloca i32, align 4
store i32* %dims, i32** %1, align 8
store i32 %dimc, i32* %2, align 4
%4 = load i32, i32* %2, align 4
```

```
%5 = zext i32 %4 to i64
%6 = call i8* @llvm.stacksave()
store i8* %6, i8** %3
%7 = alloca i32, i64 %5, align 16
store i32 0, i32* %total, align 4
store i32 0, i32* %i, align 4
br label %8

; <label>:8                                       ; preds = %56, %0
%9 = load i32, i32* %i, align 4
%10 = load i32, i32* %2, align 4
%11 = icmp slt i32 %9, %10
br i1 %11, label %12, label %59

; <label>:12                                      ; preds = %8
%13 = load i32, i32* %i, align 4
%14 = sext i32 %13 to i64
%15 = getelementptr inbounds i32, i32* %7, i64 %14
store i32 1, i32* %15, align 4
store i32 0, i32* %j, align 4
br label %16

; <label>:16                                      ; preds = %31, %12
%17 = load i32, i32* %j, align 4
%18 = load i32, i32* %i, align 4
%19 = icmp slt i32 %17, %18
br i1 %19, label %20, label %34

; <label>:20                                      ; preds = %16
%21 = load i32, i32* %j, align 4
%22 = sext i32 %21 to i64
%23 = load i32*, i32** %1, align 8
%24 = getelementptr inbounds i32, i32* %23, i64 %22
%25 = load i32, i32* %24, align 4
%26 = load i32, i32* %i, align 4
%27 = sext i32 %26 to i64
%28 = getelementptr inbounds i32, i32* %7, i64 %27
%29 = load i32, i32* %28, align 4
%30 = mul nsw i32 %29, %25
store i32 %30, i32* %28, align 4
br label %31

; <label>:31                                      ; preds = %20
%32 = load i32, i32* %j, align 4
%33 = add nsw i32 %32, 1
store i32 %33, i32* %j, align 4
br label %16

; <label>:34                                      ; preds = %16
```

```
%35 = load i32, i32* %i, align 4
%36 = sext i32 %35 to i64
%37 = load i32*, i32** %1, align 8
%38 = getelementptr inbounds i32, i32* %37, i64 %36
%39 = load i32, i32* %38, align 4
%40 = add nsw i32 %39, 1
%41 = load i32, i32* %i, align 4
%42 = sext i32 %41 to i64
%43 = getelementptr inbounds i32, i32* %7, i64 %42
%44 = load i32, i32* %43, align 4
%45 = mul nsw i32 %44, %40
store i32 %45, i32* %43, align 4
%46 = load i32, i32* %total, align 4
%47 = load i32, i32* %i, align 4
%48 = sext i32 %47 to i64
%49 = getelementptr inbounds i32, i32* %7, i64 %48
%50 = load i32, i32* %49, align 4
%51 = add nsw i32 %50, %46
store i32 %51, i32* %49, align 4
%52 = load i32, i32* %i, align 4
%53 = sext i32 %52 to i64
%54 = getelementptr inbounds i32, i32* %7, i64 %53
%55 = load i32, i32* %54, align 4
store i32 %55, i32* %total, align 4
br label %56

; <label>:56                                      ; preds = %34
%57 = load i32, i32* %i, align 4
%58 = add nsw i32 %57, 1
store i32 %58, i32* %i, align 4
br label %8

; <label>:59                                      ; preds = %8
%60 = load i32, i32* %2, align 4
%61 = zext i32 %60 to i64
%62 = alloca i32, i64 %61, align 16
store i32 0, i32* %i1, align 4
br label %63

; <label>:63                                      ; preds = %71, %59
%64 = load i32, i32* %i1, align 4
%65 = load i32, i32* %2, align 4
%66 = icmp slt i32 %64, %65
br i1 %66, label %67, label %74

; <label>:67                                      ; preds = %63
%68 = load i32, i32* %i1, align 4
%69 = sext i32 %68 to i64
%70 = getelementptr inbounds i32, i32* %62, i64 %69
```

```
store i32 0, i32* %70, align 4
br label %71

; <label>:71                                      ; preds = %67
%72 = load i32, i32* %i1, align 4
%73 = add nsw i32 %72, 1
store i32 %73, i32* %i1, align 4
br label %63

; <label>:74                                      ; preds = %63
store i32 0, i32* %length, align 4
store i32 0, i32* %i2, align 4
br label %75

; <label>:75                                      ; preds = %108, %74
%76 = load i32, i32* %i2, align 4
%77 = load i32, i32* %2, align 4
%78 = icmp slt i32 %76, %77
br i1 %78, label %79, label %111

; <label>:79                                      ; preds = %75
store i32 1, i32* %tmp, align 4
%80 = load i32, i32* %i2, align 4
%81 = sub nsw i32 %80, 1
store i32 %81, i32* %j3, align 4
br label %82

; <label>:82                                      ; preds = %93, %79
%83 = load i32, i32* %j3, align 4
%84 = icmp sge i32 %83, 0
br i1 %84, label %85, label %96

; <label>:85                                      ; preds = %82
%86 = load i32, i32* %j3, align 4
%87 = sext i32 %86 to i64
%88 = load i32*, i32** %1, align 8
%89 = getelementptr inbounds i32, i32* %88, i64 %87
%90 = load i32, i32* %89, align 4
%91 = load i32, i32* %tmp, align 4
%92 = mul nsw i32 %91, %90
store i32 %92, i32* %tmp, align 4
br label %93

; <label>:93                                      ; preds = %85
%94 = load i32, i32* %j3, align 4
%95 = add nsw i32 %94, -1
store i32 %95, i32* %j3, align 4
br label %82
```

```
; <label>:96                                        ; preds = %82
%97 = load i32, i32* %i2, align 4
%98 = sext i32 %97 to i64
%99 = load i32*, i32** %1, align 8
%100 = getelementptr inbounds i32, i32* %99, i64 %98
%101 = load i32, i32* %100, align 4
%102 = add nsw i32 %101, 1
%103 = load i32, i32* %tmp, align 4
%104 = mul nsw i32 %103, %102
store i32 %104, i32* %tmp, align 4
%105 = load i32, i32* %tmp, align 4
%106 = load i32, i32* %length, align 4
%107 = add nsw i32 %106, %105
store i32 %107, i32* %length, align 4
br label %108

; <label>:108                                       ; preds = %96
%109 = load i32, i32* %i2, align 4
%110 = add nsw i32 %109, 1
store i32 %110, i32* %i2, align 4
br label %75

; <label>:111                                       ; preds = %75
%112 = load i32, i32* %length, align 4
%113 = sext i32 %112 to i64
%114 = call noalias i8* bitcast (i8* (i32)* @malloc to i8* (i64)*)(i64 %113) #1
%115 = bitcast i8* %114 to i64*
store i64* %115, i64** %arr, align 8
store i32 0, i32* %i4, align 4
br label %116

; <label>:116                                       ; preds = %125, %111
%117 = load i32, i32* %i4, align 4
%118 = load i32, i32* %length, align 4
%119 = icmp slt i32 %117, %118
br i1 %119, label %120, label %128

; <label>:120                                       ; preds = %116
%121 = load i32, i32* %i4, align 4
%122 = sext i32 %121 to i64
%123 = load i64*, i64** %arr, align 8
%124 = getelementptr inbounds i64, i64* %123, i64 %122
store i64 0, i64* %124, align 8
br label %125

; <label>:125                                       ; preds = %120
%126 = load i32, i32* %i4, align 4
%127 = add nsw i32 %126, 1
store i32 %127, i32* %i4, align 4
```

```
br label %116

; <label>:128                                     ; preds = %116
%129 = load i64*, i64** %arr, align 8
%130 = load i32*, i32** %1, align 8
%131 = load i32, i32* %2, align 4
call void @rec_init(i64* %129, i32 0, i32* %7, i32* %62, i32* %130, i32 %131, i32 0)
%132 = load i64*, i64** %arr, align 8
%133 = load i8*, i8** %3
call void @llvm.stackrestore(i8* %133)
ret i64* %132
}

; Function Attrs: nounwind
declare i8* @llvm.stacksave() #1

; Function Attrs: nounwind
declare void @llvm.stackrestore(i8*) #1

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" }
attributes #1 = { nounwind }

!llvm.ident = !{!0}

!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

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

analyzer.ml

```
open Sast
open Ast
open Processor
open Utils
open Filepath
open Conf

module StringMap = Map.Make (String)

module StringSet = Set.Make (String)

let struct_indexes:(string, int) Hashtbl.t = Hashtbl.create 10
let predecessors:(string, string list) Hashtbl.t = Hashtbl.create 10

module SS = Set.Make(
struct
let compare = Pervasives.compare
type t = datatype
end )

type class_map = {
        field_map       : Ast.field StringMap.t;
        func_map        : Ast.func_decl StringMap.t;
        constructor_map : Ast.func_decl StringMap.t;
        reserved_map         : sfunc_decl StringMap.t;
        cdecl                          : Ast.class_decl;
}

type env = {
        env_class_maps: class_map StringMap.t;
        env_name      : string;
        env_cmap           : class_map;
        env_locals    : datatype StringMap.t;
        env_parameters: Ast.formal StringMap.t;
        env_returnType: datatype;
        env_in_for    : bool;
        env_in_while  : bool;
        env_reserved  : sfunc_decl list;
}

let update_env_name env env_name =
{
        env_class_maps = env.env_class_maps;
        env_name       = env_name;
        env_cmap            = env.env_cmap;
        env_locals     = env.env_locals;
        env_parameters = env.env_parameters;
```

```
        env_returnType = env.env_returnType;
        env_in_for     = env.env_in_for;
        env_in_while   = env.env_in_while;
        env_reserved   = env.env_reserved;
}

let update_call_stack env in_for in_while =
{
        env_class_maps = env.env_class_maps;
        env_name       = env.env_name;
        env_cmap             = env.env_cmap;
        env_locals     = env.env_locals;
        env_parameters = env.env_parameters;
        env_returnType = env.env_returnType;
        env_in_for     = in_for;
        env_in_while   = in_while;
        env_reserved   = env.env_reserved;
}

let append_code_to_constructor fbody cname ret_type =
let key = Hashtbl.find struct_indexes cname in
let init_this = [SLocal(
ret_type,
"this",
SCall(        "cast",
[SCall("malloc",
[
SCall("sizeof", [SId("ignore", ret_type)], Datatype(Int_t), 0)
],
Arraytype(Char_t, 1), 0)
],
ret_type,
0
)
);
SExpr(
SAssign(
SObjAccess(
SId("this", ret_type),
SId(".key", Datatype(Int_t)),
Datatype(Int_t)
),
SInt_Lit(key),
Datatype(Int_t)
),
Datatype(Int_t)
)
]
in
```

```
let ret_this =
[
SReturn(
SId("this", ret_type),
ret_type
)
]
in
(* Need to check for duplicate default constructs *)
(* Also need to add malloc around other constructors *)
init_this @ fbody @ ret_this

let default_constructor_body cname =
let ret_type = Datatype(Objecttype(cname)) in
let fbody = [] in
append_code_to_constructor fbody cname ret_type

let default_sc cname =
{
        sfname                          = Ast.FName (cname ^ "." ^ "constructor");
        sreturnType         = Datatype(Objecttype(cname));
        sformals                  = [];
        sbody                           = default_constructor_body cname;
        func_type                 = Sast.User;
        overrides       = false;
        source                          = "NA";
}

let default_c cname =
{
        scope                         = Ast.Public;
        fname                          = Ast.Constructor;
        returnType                  = Datatype(ConstructorType);
        formals                  = [];
        body                          = [];
        overrides                  = false;
        root_cname                  = None;
}

let process_includes filename includes classes =
(* Bring in each include  *)
let processInclude include_statement =
let file_in = open_in include_statement in
let lexbuf = Lexing.from_channel file_in in
let token_list = Processor.build_token_list lexbuf in
let program = Processor.parser include_statement token_list in
ignore(close_in file_in);
program
in
```

```
let rec iterate_includes classes m = function
[] -> classes
| (Include h) :: t ->
let h = if h = "stdlib" then Conf.stdlib_path else h in
(* Check each include against the map *)
let realpath = Filepath.realpath h in
if StringMap.mem realpath m then
iterate_includes (classes) (m) (t)
else
let result = processInclude realpath in
match result with Program(i,c) ->
iterate_includes (classes @ c) (StringMap.add realpath 1 m) (i @ t)
in
iterate_includes classes (StringMap.add (Filepath.realpath filename) 1 StringMap.empty)
↪  includes

let get_name cname fdecl =
(* We use '.' to separate types so llvm will recognize the function name and it won't
↪  conflict *)
(* let params = List.fold_left (fun s -> (function Formal(t, _) -> s ^ "." ^
↪  Utils.string_of_datatype t | _ -> "" )) "" fdecl.formals in *)
let name = Utils.string_of_fname fdecl.fname in
if name = "main"
then "main"
else cname ^ "." ^ name(*  ^ params *)

let get_constructor_name cname fdecl =
let params = List.fold_left (fun s -> (function Formal(t, _) -> s ^ "." ^
↪  Utils.string_of_datatype t | _ -> "" )) "" fdecl.formals in
let name = Utils.string_of_fname fdecl.fname in
cname ^ "." ^ name ^ params

let get_name_without_class fdecl =
(* We use '.' to separate types so llvm will recognize the function name and it won't
↪  conflict *)
let params = List.fold_left (fun s -> (function Formal(t, _) -> s ^ "." ^
↪  Utils.string_of_datatype t | _ -> "" )) "" fdecl.formals in
let name = Utils.string_of_fname fdecl.fname in
let ret_type = Utils.string_of_datatype fdecl.returnType in
ret_type ^ "." ^ name ^ "." ^ params

(* Generate list of all classes to be used for semantic checking *)
let build_class_maps reserved cdecls =
let reserved_map = List.fold_left (fun m f -> StringMap.add (Utils.string_of_fname
↪  f.sfname) f m) StringMap.empty reserved in
let helper m (cdecl:Ast.class_decl) =
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
let funcname = get_name cdecl.cname in
```

```
let funcfun m fdecl =
if (StringMap.mem (funcname fdecl) m)
then raise(Exceptions.DuplicateFunction(funcname fdecl))
else if (StringMap.mem (Utils.string_of_fname fdecl.fname) reserved_map)
then raise(Exceptions.CannotUseReservedFuncName(Utils.string_of_fname fdecl.fname))
else (StringMap.add (funcname fdecl) fdecl m)
in
let constructor_name = get_constructor_name cdecl.cname in
let constructorfun m fdecl =
if fdecl.formals = [] then m
else if StringMap.mem (constructor_name fdecl) m
then raise(Exceptions.DuplicateConstructor)
else (StringMap.add (constructor_name fdecl) fdecl m)
in
let default_c = default_c cdecl.cname in
let constructor_map = StringMap.add (get_constructor_name cdecl.cname default_c)
 default_c StringMap.empty in
(if (StringMap.mem cdecl.cname m) then raise (Exceptions.DuplicateClassName(cdecl.cname))
 else
StringMap.add cdecl.cname
{         field_map = List.fold_left fieldfun StringMap.empty cdecl.cbody.fields;
        func_map = List.fold_left funcfun StringMap.empty cdecl.cbody.methods;
        constructor_map = List.fold_left constructorfun constructor_map
         cdecl.cbody.constructors;
        reserved_map = reserved_map;
        cdecl = cdecl }
m) in
List.fold_left helper StringMap.empty cdecls

let rec get_all_descendants cname accum =
if Hashtbl.mem predecessors cname then
let direct_descendants = Hashtbl.find predecessors cname in
let add_childs_descendants desc_set direct_descendant =
get_all_descendants direct_descendant (StringSet.add direct_descendant desc_set)
in
List.fold_left add_childs_descendants accum direct_descendants
else accum

let inherited potential_predec potential_child =
match potential_predec, potential_child with
Datatype(Objecttype(predec_cname)), Datatype(Objecttype(child_cname)) ->
let descendants = get_all_descendants predec_cname StringSet.empty in
if (predec_cname = child_cname) || (StringSet.mem child_cname descendants) then true
else raise (Exceptions.LocalAssignTypeMismatch(predec_cname, child_cname))
| _ , _ -> false

let get_equality_binop_type type1 type2 se1 se2 op =
(* Equality op not supported for float operands. The correct way to test floats
for equality is to check the difference between the operands in question *)
```

```
if (type1 = Datatype(Float_t) || type2 = Datatype(Float_t)) then raise (Exceptions.InvalidBinopExpression "Equality operation is not supported for Float types")
else
match type1, type2 with
Datatype(Char_t), Datatype(Int_t)
|         Datatype(Int_t), Datatype(Char_t)
|         Datatype(Objecttype(_)), Datatype(Null_t)
|         Datatype(Null_t), Datatype(Objecttype(_))
|         Datatype(Null_t), Arraytype(_, _)
|         Arraytype(_, _), Datatype(Null_t) -> SBinop(se1, op, se2, Datatype(Bool_t))
| _ ->
if type1 = type2 then SBinop(se1, op, se2, Datatype(Bool_t))
else raise (Exceptions.InvalidBinopExpression "Equality operator can't operate on different types, with the exception of Int_t and Char_t")

let get_logical_binop_type se1 se2 op = function
(Datatype(Bool_t), Datatype(Bool_t)) -> SBinop(se1, op, se2, Datatype(Bool_t))
| _ -> raise (Exceptions.InvalidBinopExpression "Logical operators only operate on Bool_t types")

let get_comparison_binop_type type1 type2 se1 se2 op =
let numerics = SS.of_list [Datatype(Int_t); Datatype(Char_t); Datatype(Float_t)]
in
if SS.mem type1 numerics && SS.mem type2 numerics
then SBinop(se1, op, se2, Datatype(Bool_t))
else raise (Exceptions.InvalidBinopExpression "Comparison operators operate on numeric types only")


let get_arithmetic_binop_type se1 se2 op = function
(Datatype(Int_t), Datatype(Float_t))
|         (Datatype(Float_t), Datatype(Int_t))
|         (Datatype(Float_t), Datatype(Float_t))         -> SBinop(se1, op, se2, Datatype(Float_t))

|         (Datatype(Int_t), Datatype(Char_t))
|         (Datatype(Char_t), Datatype(Int_t))
|         (Datatype(Char_t), Datatype(Char_t))         -> SBinop(se1, op, se2, Datatype(Char_t))

|         (Datatype(Int_t), Datatype(Int_t))                  -> SBinop(se1, op, se2, Datatype(Int_t))

| _ -> raise (Exceptions.InvalidBinopExpression "Arithmetic operators don't support these types")

let rec get_ID_type env s =
try StringMap.find s env.env_locals
```

```
with | Not_found ->
try let formal = StringMap.find s env.env_parameters in
(function Formal(t, _) -> t | Many t -> t ) formal
with | Not_found -> raise (Exceptions.UndefinedID s)

and check_array_primitive env el =
let rec iter t sel = function
[] -> sel, t
|         e :: el ->
let se, _ = expr_to_sexpr env e in
let se_t = get_type_from_sexpr se in
if t = se_t
then iter t (se :: sel) el
else
let t1 = Utils.string_of_datatype t in
let t2 = Utils.string_of_datatype se_t in
raise(Exceptions.InvalidArrayPrimitiveConsecutiveTypes(t1, t2))
in
let se, _ = expr_to_sexpr env (List.hd el) in
let el = List.tl el in
let se_t = get_type_from_sexpr se in
let sel, t = iter se_t ([se]) el in
let se_t = match t with
Datatype(x) -> Arraytype(x, 1)
|         Arraytype(x, n) -> Arraytype(x, n+1)
|         _ as t -> raise(Exceptions.InvalidArrayPrimitiveType(Utils.string_of_datatype
↪  t))
in
SArrayPrimitive(sel, se_t)

and check_array_init env d el =
(* Get dimension size for the array being created *)
let array_complexity = List.length el in
let check_elem_type e =
let sexpr, _ = expr_to_sexpr env e in
let sexpr_type = get_type_from_sexpr sexpr in
if sexpr_type = Datatype(Int_t)
then sexpr
else raise(Exceptions.MustPassIntegerTypeToArrayCreate)
in
let convert_d_to_arraytype = function
Datatype(x) -> Arraytype(x, array_complexity)
|         _ as t ->
let error_msg = Utils.string_of_datatype t in
raise (Exceptions.ArrayInitTypeInvalid(error_msg))
in
let sexpr_type = convert_d_to_arraytype d in
let sel = List.map check_elem_type el in
SArrayCreate(d, sel, sexpr_type)
```

```

and check_array_access env e el =
(* Get dimensions of array, ex: foo[10][4][2] is dimen=3 *)
let array_dimensions = List.length el in
(* Check every e in el is of type Datatype(Int_t). Ensure all indices are ints *)
let check_elem_type arg =
let sexpr, _ = expr_to_sexpr env arg in
let sexpr_type = get_type_from_sexpr sexpr in
if sexpr_type = Datatype(Int_t)
then sexpr
else raise(Exceptions.MustPassIntegerTypeToArrayAccess)
in
(* converting e to se also checks if the array id has been declared  *)
let se, _ = expr_to_sexpr env e in
let se_type = get_type_from_sexpr se in

(* Check that e has enough dimens as e's in el. Return overall datatype of access*)
let check_array_dim_vs_params num_params = function
Arraytype(t, n) ->
if num_params < n then
Arraytype(t, (n-num_params))
else if num_params = n then
Datatype(t)
else
raise (Exceptions.ArrayAccessInvalidParamLength(string_of_int num_params, string_of_int
 n))
|          _ as t ->
let error_msg = Utils.string_of_datatype t in
raise (Exceptions.ArrayAccessExpressionNotArray(error_msg))
in
let sexpr_type = check_array_dim_vs_params array_dimensions se_type in
let sel = List.map check_elem_type el in

SArrayAccess(se, sel, sexpr_type)

and check_obj_access env lhs rhs =
let check_lhs = function
This                          -> SId("this", Datatype(Objecttype(env.env_name)))
|         Id s                          -> SId(s, get_ID_type env s)
|          ArrayAccess(e, el)        -> check_array_access env e el
|          _ as e          -> raise (Exceptions.LHSofRootAccessMustBeIDorFunc
 (Utils.string_of_expr e))
in
let ptype_name parent_type = match parent_type with
Datatype(Objecttype(name))          -> name
|          _ as d                                                   -> raise
 (Exceptions.ObjAccessMustHaveObjectType (Utils.string_of_datatype d))
in
let rec check_rhs (env) parent_type (top_level_env) =
```

```
let pt_name = ptype_name parent_type in
let get_id_type_from_object env (id) cname tlenv =
let cmap = StringMap.find cname env.env_class_maps in
let match_field f = match f with
Field(scope, d, n) ->
(* Have to update this with all parent classes checks *)
if scope = Ast.Private && tlenv.env_name <> env.env_name then
raise(Exceptions.CannotAccessPrivateFieldInNonProperScope(n, env.env_name, tlenv.env_name))
else d
in
try match_field (StringMap.find id cmap.field_map)
with | Not_found -> raise (Exceptions.UnknownIdentifierForClass(id, cname))
in
function
(* Check fields in parent *)
Id s                                -> SId(s, (get_id_type_from_object env s pt_name top_level_env)), env
(* Check functions in parent *)
|         Call(fname, el)         ->
let env = update_env_name env pt_name in
check_call_type top_level_env true env fname el, env
(* Set parent, check if base is field *)
|         ObjAccess(e1, e2)         ->
let old_env = env in
let lhs, env = check_rhs env parent_type top_level_env e1 in
let lhs_type = get_type_from_sexpr lhs in

let pt_name = ptype_name lhs_type in
let lhs_env = update_env_name env pt_name in

let rhs, env = check_rhs lhs_env lhs_type top_level_env e2 in
let rhs_type = get_type_from_sexpr rhs in
SObjAccess(lhs, rhs, rhs_type), old_env
|         _ as e                                -> raise (Exceptions.InvalidAccessLHS (Utils.string_of_expr e))
in
let arr_lhs, _ = expr_to_sexpr env lhs in
let arr_lhs_type = get_type_from_sexpr arr_lhs in
match arr_lhs_type with
Arraytype(Char_t, 1) -> raise(Exceptions.CannotAccessLengthOfCharArray)
|        Arraytype(_, _) ->
let rhs = match rhs with
Id("length") -> SId("length", Datatype(Int_t))
|         _ -> raise(Exceptions.CanOnlyAccessLengthOfArray)
in
SObjAccess(arr_lhs, rhs, Datatype(Int_t))
| _ ->
let lhs = check_lhs lhs in
```

```
let lhs_type = get_type_from_sexpr lhs in

let ptype_name = ptype_name lhs_type in
let lhs_env = update_env_name env ptype_name in

let rhs, _ = check_rhs lhs_env lhs_type env rhs in
let rhs_type = get_type_from_sexpr rhs in
SObjAccess(lhs, rhs, rhs_type)

and check_call_type top_level_env isObjAccess env fname el =
let sel, env = exprl_to_sexprl env el in
(* check that 'env.env_name' is in the list of defined classes *)
let cmap =
try StringMap.find env.env_name env.env_class_maps
with | Not_found -> raise (Exceptions.UndefinedClass env.env_name)
in

let handle_param formal param =
let fty = match formal with Formal(d, _) -> d | _ -> Datatype(Void_t) in
let pty = get_type_from_sexpr param in
match fty, pty with
Datatype(Objecttype(f)), Datatype(Objecttype(p)) ->
if f <> p then
try let descendants = Hashtbl.find predecessors f in
let _ = try List.find (fun d -> p = d) descendants
with | Not_found -> raise(Exceptions.CannotPassNonInheritedClassesInPlaceOfOthers(f, p))
in
let rt = Datatype(Objecttype(f)) in
SCall("cast", [param; SId("ignore", rt)], rt, 0)
with | Not_found -> raise(Exceptions.ClassIsNotExtendedBy(f, p))
else param
|        _ -> if fty = pty then param else
↪  raise(Exceptions.IncorrectTypePassedToFunction(fname, Utils.string_of_datatype pty))
in

let index fdecl fname =
let cdecl = cmap.cdecl in
(* Have to update this with all parent classes checks *)
let _ =
if fdecl.scope = Ast.Private && top_level_env.env_name <> env.env_name then
raise(Exceptions.CannotAccessPrivateFunctionInNonProperScope(get_name env.env_name fdecl,
↪  env.env_name, top_level_env.env_name))
in
(* Not exactly sure why there needs to be a list.rev *)
let fns = List.rev cdecl.cbody.methods in
let rec find x lst =
match lst with
| [] -> raise (Failure ("Could not find " ^ fname))
| fdecl :: t ->
```

```
let search_name = (get_name env.env_name fdecl) in
if x = search_name then 0
else if search_name = "main" then find x t
else 1 + find x t
in
find fname fns
in

let handle_params (formals) params =
match formals, params with
[Many(Any)], _ -> params
|         [], [] -> []
|         [], _
|         _, [] -> raise(Exceptions.IncorrectTypePassedToFunction(fname,
↪  Utils.string_of_datatype (Datatype(Void_t))))
|         _ ->
let len1 = List.length formals in
let len2 = List.length params in
if len1 <> len2 then raise(Exceptions.IncorrectNumberOfArguments(fname, len1, len2))
else
List.map2 handle_param formals sel
in

let sfname = env.env_name ^ "." ^ fname in
try let func = StringMap.find fname cmap.reserved_map in
let actuals = handle_params func.sformals sel in
SCall(fname, actuals, func.sreturnType, 0)
with | Not_found ->
try let f = StringMap.find sfname cmap.func_map in
let actuals = handle_params f.formals sel in
let index = index f sfname in
SCall(sfname, actuals, f.returnType, index)
with | Not_found -> raise(Exceptions.FunctionNotFound(env.env_name, sfname)) | _ as ex ->
↪  raise ex

and check_object_constructor env s el =
let sel, env = exprl_to_sexprl env el in
(* check that 'env.env_name' is in the list of defined classes *)
let cmap =
try StringMap.find s env.env_class_maps
with | Not_found -> raise (Exceptions.UndefinedClass s)
in
(* get a list of the types of the actuals to match against defined function formals *)
let params = List.fold_left (fun s e -> s ^ "." ^ (Utils.string_of_datatype
↪  (get_type_from_sexpr e))) "" sel in
let constructor_name = s ^ "." ^ "constructor" ^ params in
let _ =
try StringMap.find constructor_name cmap.constructor_map
with | Not_found -> raise (Exceptions.ConstructorNotFound constructor_name)
```

```
in
let ftype = Datatype(Objecttype(s)) in
(* Add a reference to the class in front of the function call *)
(* Must properly handle the case where this is a reserved function *)
SObjectCreate(constructor_name, sel, ftype)

and check_assign env e1 e2 =
let se1, env = expr_to_sexpr env e1 in
let se2, env = expr_to_sexpr env e2 in
let type1 = get_type_from_sexpr se1 in
let type2 = get_type_from_sexpr se2 in
match (type1, se2) with
Datatype(Objecttype(_)), SNull
|         Arraytype(_, _), SNull -> SAssign(se1, se2, type1)
|    _ ->
match type1, type2 with
Datatype(Char_t), Datatype(Int_t)
|         Datatype(Int_t), Datatype(Char_t) -> SAssign(se1, se2, type1)
|         Datatype(Objecttype(d)), Datatype(Objecttype(t)) ->
if d = t then SAssign(se1, se2, type1)
else if inherited type1 type2 then
SAssign(se1, SCall("cast", [se2; SId("ignore", type1)], type1, 0), type1)
else raise (Exceptions.AssignmentTypeMismatch(Utils.string_of_datatype type1,
↪  Utils.string_of_datatype type2))
| _ ->
if type1 = type2
then SAssign(se1, se2, type1)
else raise (Exceptions.AssignmentTypeMismatch(Utils.string_of_datatype type1,
↪  Utils.string_of_datatype type2))

and check_unop env op e =
let check_num_unop t = function
Sub         -> t
|         _                 -> raise(Exceptions.InvalidUnaryOperation)
in
let check_bool_unop = function
Not         -> Datatype(Bool_t)
|         _                 -> raise(Exceptions.InvalidUnaryOperation)
in
let se, env = expr_to_sexpr env e in
let t = get_type_from_sexpr se in
match t with
Datatype(Int_t)
|        Datatype(Float_t)          -> SUnop(op, se, check_num_unop t op)
|          Datatype(Bool_t)          -> SUnop(op, se, check_bool_unop op)
|         _ -> raise(Exceptions.InvalidUnaryOperation)

and check_binop env e1 op e2 =
let se1, env = expr_to_sexpr env e1 in
```

```
let se2, env = expr_to_sexpr env e2 in
let type1 = get_type_from_sexpr se1 in
let type2 = get_type_from_sexpr se2 in
match op with
Equal | Neq -> get_equality_binop_type type1 type2 se1 se2 op
| And | Or -> get_logical_binop_type se1 se2 op (type1, type2)
| Less | Leq | Greater | Geq -> get_comparison_binop_type type1 type2 se1 se2 op
| Add | Mult | Sub | Div | Mod -> get_arithmetic_binop_type se1 se2 op (type1, type2)
| _ -> raise (Exceptions.InvalidBinopExpression ((Utils.string_of_op op) ^ " is not a
↪  supported binary op"))

and check_delete env e =
let se, _ = expr_to_sexpr env e in
let t = get_type_from_sexpr se in
match t with
Arraytype(_, _) | Datatype(Objecttype(_)) -> SDelete(se)
|          _ -> raise(Exceptions.CanOnlyDeleteObjectsOrArrays)

and expr_to_sexpr env = function
Int_Lit i            -> SInt_Lit(i), env
|   Boolean_Lit b       -> SBoolean_Lit(b), env
|   Float_Lit f         -> SFloat_Lit(f), env
|   String_Lit s        -> SString_Lit(s), env
|   Char_Lit c          -> SChar_Lit(c), env
|   This                -> SId("this", Datatype(Objecttype(env.env_name))), env
|   Id s                -> SId(s, get_ID_type env s), env
|   Null                -> SNull, env
|   Noexpr              -> SNoexpr, env

|   ObjAccess(e1, e2)   -> check_obj_access env e1 e2, env
|   ObjectCreate(s, el) -> check_object_constructor env s el, env
|   Call(s, el)         -> check_call_type env false env s el, env

|   ArrayCreate(d, el)  -> check_array_init env d el, env
|   ArrayAccess(e, el)  -> check_array_access env e el, env
|   ArrayPrimitive el   -> check_array_primitive env el, env

|   Assign(e1, e2)      -> check_assign env e1 e2, env
|   Unop(op, e)         -> check_unop env op e, env
|   Binop(e1, op, e2)   -> check_binop env e1 op e2, env
|         Delete(e)                         -> check_delete env e, env


and get_type_from_sexpr = function
SInt_Lit(_)                                 -> Datatype(Int_t)
|         SBoolean_Lit(_)                         -> Datatype(Bool_t)
|         SFloat_Lit(_)                         -> Datatype(Float_t)
|         SString_Lit(_)                          -> Arraytype(Char_t, 1)
|         SChar_Lit(_)                          -> Datatype(Char_t)
```

```
|         SId(_, d)                                 -> d
|         SBinop(_, _, _, d)                  -> d
|         SAssign(_, _, d)                  -> d
|         SNoexpr                                -> Datatype(Void_t)
|         SArrayCreate(_, _, d)         -> d
|         SArrayAccess(_, _, d)          -> d
|         SObjAccess(_, _, d)                 -> d
|         SCall(_, _, d, _)                 -> d
|   SObjectCreate(_, _, d)          -> d
|         SArrayPrimitive(_, d)         -> d
|          SUnop(_, _, d)                        -> d
|         SNull                                       -> Datatype(Null_t)
|         SDelete _                                -> Datatype(Void_t)

and exprl_to_sexprl env el =
let env_ref = ref(env) in
let rec helper = function
head::tail ->
let a_head, env = expr_to_sexpr !env_ref head in
env_ref := env;
a_head::(helper tail)
| [] -> []
in (helper el), !env_ref

let rec local_handler d s e env =
if StringMap.mem s env.env_locals
then raise (Exceptions.DuplicateLocal s)
else
let se, env = expr_to_sexpr env e in
let t = get_type_from_sexpr se in
if t = Datatype(Void_t) || t = Datatype(Null_t) || t = d || (inherited d t)
then
let new_env = {
        env_class_maps = env.env_class_maps;
        env_name = env.env_name;
        env_cmap = env.env_cmap;
        env_locals = StringMap.add s d env.env_locals;
        env_parameters = env.env_parameters;
        env_returnType = env.env_returnType;
        env_in_for = env.env_in_for;
        env_in_while = env.env_in_while;
        env_reserved = env.env_reserved;
} in
(* if the user-defined type being declared is not in global classes map, it is an
↪ undefined class *)
(match d with
Datatype(Objecttype(x)) ->
(if not (StringMap.mem (Utils.string_of_object d) env.env_class_maps)
then raise (Exceptions.UndefinedClass (Utils.string_of_object d))
```

```
else
let local = if inherited d t then SLocal(t, s, se) else SLocal(d, s, se)
in local, new_env)
| _ -> SLocal(d, s, se), new_env)
else
(let type1 = (Utils.string_of_datatype t) in
let type2 = (Utils.string_of_datatype d) in
let ex = Exceptions.LocalAssignTypeMismatch(type1, type2) in
raise ex)

let rec check_sblock sl env = match sl with
[] -> SBlock([SExpr(SNoexpr, Datatype(Void_t))]), env
|          _   ->
let sl, _ = convert_stmt_list_to_sstmt_list env sl in
SBlock(sl), env

and check_expr_stmt e env =
let se, env = expr_to_sexpr env e in
let t = get_type_from_sexpr se in
SExpr(se, t), env

and check_return e env =
let se, _ = expr_to_sexpr env e in
let t = get_type_from_sexpr se in
match t, env.env_returnType with
Datatype(Null_t), Datatype(Objecttype(_))
|         Datatype(Null_t), Arraytype(_, _) -> SReturn(se, t), env
|          _ ->
if t = env.env_returnType
then SReturn(se, t), env
else raise (Exceptions.ReturnTypeMismatch(Utils.string_of_datatype t,
↪  Utils.string_of_datatype env.env_returnType))

and check_if e s1 s2 env =
let se, _ = expr_to_sexpr env e in
let t = get_type_from_sexpr se in
let ifbody, _ = parse_stmt env s1 in
let elsebody, _ = parse_stmt env s2 in
if t = Datatype(Bool_t)
then SIf(se, ifbody, elsebody), env
else raise Exceptions.InvalidIfStatementType

and check_for e1 e2 e3 s env =
let old_val = env.env_in_for in
let env = update_call_stack env true env.env_in_while in

let se1, _ = expr_to_sexpr env e1 in
let se2, _ = expr_to_sexpr env e2 in
let se3, _ = expr_to_sexpr env e3 in
```

```
let forbody, _ = parse_stmt env s in
let conditional = get_type_from_sexpr se2 in
let sfor =
if (conditional = Datatype(Bool_t) || conditional = Datatype(Void_t))
then SFor(se1, se2, se3, forbody)
else raise Exceptions.InvalidForStatementType
in

let env = update_call_stack env old_val env.env_in_while in
sfor, env

and check_while e s env =
let old_val = env.env_in_while in
let env = update_call_stack env env.env_in_for true in

let se, _ = expr_to_sexpr env e in
let t = get_type_from_sexpr se in
let sstmt, _ = parse_stmt env s in
let swhile =
if (t = Datatype(Bool_t) || t = Datatype(Void_t))
then SWhile(se, sstmt)
else raise Exceptions.InvalidWhileStatementType
in

let env = update_call_stack env env.env_in_for old_val in
swhile, env

and check_break env =
if env.env_in_for || env.env_in_while then
SBreak, env
else
raise Exceptions.CannotCallBreakOutsideOfLoop

and check_continue env =
if env.env_in_for || env.env_in_while then
SContinue, env
else
raise Exceptions.CannotCallContinueOutsideOfLoop

and parse_stmt env = function
Block sl                                  -> check_sblock sl env
|         Expr e                                          -> check_expr_stmt e env
|         Return e                                  -> check_return e env
|         If(e, s1, s2)                         -> check_if e s1 s2         env
|         For(e1, e2, e3, e4)         -> check_for e1 e2 e3 e4 env
|         While(e, s)                                 -> check_while e s env
|          Break                                          -> check_break env (* Need to
↪   check if in right context *)
```

```
|   Continue                               -> check_continue env (* Need to check if in
↪  right context *)
|   Local(d, s, e)                        -> local_handler d s e env

(* Update this function to return an env object *)
and convert_stmt_list_to_sstmt_list env stmt_list =
let env_ref = ref(env) in
let rec iter = function
head::tail ->
let a_head, env = parse_stmt !env_ref head in
env_ref := env;
a_head::(iter tail)
| [] -> []
in
let sstmt_list = (iter stmt_list), !env_ref in
sstmt_list

let append_code_to_main fbody cname ret_type =
let key = Hashtbl.find struct_indexes cname in
let init_this = [SLocal(
ret_type,
"this",
SCall(        "cast",
[SCall("malloc",
[
SCall("sizeof", [SId("ignore", ret_type)], Datatype(Int_t), 0)
],
Arraytype(Char_t, 1), 0)
],
ret_type, 0
)
);
SExpr(
SAssign(
SObjAccess(
SId("this", ret_type),
SId(".key", Datatype(Int_t)),
Datatype(Int_t)
),
SInt_Lit(key),
Datatype(Int_t)
),
Datatype(Int_t)
)
]
in
init_this @ fbody

let convert_constructor_to_sfdecl class_maps reserved class_map cname constructor =
```

```
let env = {
        env_class_maps          = class_maps;
        env_name              = cname;
        env_cmap                  = class_map;
        env_locals             = StringMap.empty;
        env_parameters         = List.fold_left (fun m f -> match f with Formal(d, s) ->
         ↪  (StringMap.add s f m) | _ -> m) StringMap.empty constructor.formals;
        env_returnType         = Datatype(Objecttype(cname));
        env_in_for                  = false;
        env_in_while          = false;
        env_reserved          = reserved;
} in
let fbody = fst (convert_stmt_list_to_sstmt_list env constructor.body) in
{
        sfname                  = Ast.FName (get_constructor_name cname constructor);
        sreturnType = Datatype(Objecttype(cname));
        sformals          = constructor.formals;
        sbody                  = append_code_to_constructor fbody cname
         ↪  (Datatype(Objecttype(cname)));
        func_type          = Sast.User;
        overrides          = false;
        source                  = "NA";
}

let check_fbody fname fbody returnType =
let len = List.length fbody in
if len = 0 then () else
let final_stmt = List.hd (List.rev fbody) in
match returnType, final_stmt with
Datatype(Void_t), _ -> ()
|          _, SReturn(_, _) -> ()
|          _ -> raise(Exceptions.AllNonVoidFunctionsMustEndWithReturn(fname))

let convert_fdecl_to_sfdecl class_maps reserved class_map cname fdecl =
let root_cname = match fdecl.root_cname with
Some(x) -> x
| None -> cname
in
let class_formal =
if fdecl.overrides then
Ast.Formal(Datatype(Objecttype(root_cname)), "this")
else
Ast.Formal(Datatype(Objecttype(cname)), "this")
in
let env_param_helper m fname = match fname with
Formal(d, s) -> (StringMap.add s fname m)
|          _ -> m
in
```

```
let env_params = List.fold_left env_param_helper StringMap.empty (class_formal :: fdecl.formals) in
let env = {
        env_class_maps          = class_maps;
        env_name                = cname;
        env_cmap                    = class_map;
        env_locals              = StringMap.empty;
        env_parameters          = env_params;
        env_returnType          = fdecl.returnType;
        env_in_for                   = false;
        env_in_while            = false;
        env_reserved            = reserved;
}
in
let fbody = fst (convert_stmt_list_to_sstmt_list env fdecl.body) in
let fname = (get_name cname fdecl) in
ignore(check_fbody fname fbody fdecl.returnType);
let fbody = if fname = "main"
then (append_code_to_main fbody cname (Datatype(Objecttype(cname))))
else fbody
in
(* We add the class as the first parameter to the function for codegen *)
{
        sfname                          = Ast.FName (get_name cname fdecl);
        sreturnType             = fdecl.returnType;
        sformals                    = class_formal :: fdecl.formals;
        sbody                           = fbody;
        func_type                   = Sast.User;
        overrides           = fdecl.overrides;
        source                          = cname;
}

let convert_cdecl_to_sast sfdecls (cdecl:Ast.class_decl) =
{
        scname = cdecl.cname;
        sfields = cdecl.cbody.fields;
        sfuncs = sfdecls;
}

(*
* Given a list of func_decls for the base class and a single func_decl
* for the child class, replaces func_decls for the base class if any of them
* have the same method signature
*)
let replace_fdecl_in_base_methods base_cname base_methods child_fdecl =
let replace base_fdecl accum =
let get_root_cname = function
None -> Some(base_cname)
| Some(x) -> Some(x)
```

```
in
let modify_child_fdecl =
{
        scope = child_fdecl.scope;
        fname = child_fdecl.fname;
        returnType = child_fdecl.returnType;
        formals = child_fdecl.formals;
        body = child_fdecl.body;
        overrides = true;
        root_cname = get_root_cname base_fdecl.root_cname;
}
in
if (get_name_without_class base_fdecl) = (get_name_without_class child_fdecl)
then modify_child_fdecl::accum
else base_fdecl::accum
in
List.fold_right replace base_methods []

let merge_methods base_cname base_methods child_methods =
let check_overrides child_fdecl accum =
let base_checked_for_overrides =
replace_fdecl_in_base_methods base_cname (fst accum) child_fdecl
in
if (fst accum) = base_checked_for_overrides
then ((fst accum), child_fdecl::(snd accum))
else (base_checked_for_overrides, (snd accum))
in
let updated_base_and_child_fdecls =
List.fold_right check_overrides child_methods (base_methods, [])
in
(fst updated_base_and_child_fdecls) @ (snd updated_base_and_child_fdecls)

let merge_cdecls base_cdecl child_cdecl =
(* return a cdecl in which cdecl.cbody.fields contains the fields of
the extended class, concatenated by the fields of the child class *)
let child_cbody =
{
        fields = base_cdecl.cbody.fields @ child_cdecl.cbody.fields;
        constructors = child_cdecl.cbody.constructors;
        methods = merge_methods base_cdecl.cname base_cdecl.cbody.methods
         ↪  child_cdecl.cbody.methods
}
in
{
        cname = child_cdecl.cname;
        extends = child_cdecl.extends;
        cbody = child_cbody
}

```

```
(* returns a list of cdecls that contains inherited fields *)
let inherit_fields_cdecls cdecls inheritance_forest =
(* iterate through cdecls to make a map for lookup *)
let cdecl_lookup = List.fold_left (fun a litem -> StringMap.add litem.cname litem a)
  ↪  StringMap.empty cdecls in
let add_key key pred maps =
let elem1 = StringSet.add key (fst maps) in
let accum acc child = StringSet.add child acc in
let elem2 = List.fold_left (accum) (snd maps) pred in
(elem1, elem2)
in
let empty_s = StringSet.empty in
let res = StringMap.fold add_key inheritance_forest (empty_s, empty_s) in
let roots = StringSet.diff (fst res) (snd res) in
let rec add_inherited_fields predec desc map_to_update =
let merge_fields accum descendant =
let updated_predec_cdecl = StringMap.find predec accum in
let descendant_cdecl_to_update = StringMap.find descendant cdecl_lookup in
let merged = merge_cdecls updated_predec_cdecl descendant_cdecl_to_update in
let updated = (StringMap.add descendant merged accum) in
if (StringMap.mem descendant inheritance_forest) then
let descendants_of_descendant = StringMap.find descendant inheritance_forest in
add_inherited_fields descendant descendants_of_descendant updated
else updated
in
List.fold_left merge_fields map_to_update desc
in
(* map class name of every class_decl in `cdecls` to its inherited cdecl *)
let inherited_cdecls =
let traverse_tree tree_root accum =
let tree_root_descendant = StringMap.find tree_root inheritance_forest in
let accum_with_tree_root_mapping = StringMap.add tree_root (StringMap.find tree_root
  ↪  cdecl_lookup) accum in
add_inherited_fields tree_root tree_root_descendant accum_with_tree_root_mapping
in
StringSet.fold traverse_tree roots StringMap.empty
in
(* build a list of updated cdecls corresponding to the sequence of cdecls in `cdecls` *)
let add_inherited_cdecl cdecl accum =
let inherited_cdecl =
try StringMap.find cdecl.cname inherited_cdecls
with | Not_found -> cdecl
in
inherited_cdecl::accum
in
let result = List.fold_right add_inherited_cdecl cdecls [] in
result

let convert_cdecls_to_sast class_maps reserved (cdecls:Ast.class_decl list) =
```

```
let find_main = (fun f -> match f.sfname with FName n -> n = "main" | _ -> false) in
let get_main func_list =
let mains = (List.find_all find_main func_list) in
if List.length mains < 1 then
raise Exceptions.MainNotDefined
else if List.length mains > 1 then
raise Exceptions.MultipleMainsDefined
else List.hd mains
in
let remove_main func_list =
List.filter (fun f -> not (find_main f)) func_list
in
let find_default_constructor cdecl clist =
let default_cname = cdecl.cname ^ "." ^ "constructor" in
let find_default_c f =
match f.sfname with FName n -> n = default_cname | _ -> false
in
try let _ = List.find find_default_c clist in
clist
with | Not_found ->
let default_c = default_sc cdecl.cname in
default_c :: clist
in
let handle_cdecl cdecl =
let class_map = StringMap.find cdecl.cname class_maps in
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
let sconstructor_list = find_default_constructor cdecl sconstructor_list in
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
let sfunc_list = remove_main func_list in
let scdecl = convert_cdecl_to_sast sfunc_list cdecl in
(scdecl, func_list @ sconstructor_list)
in
let iter_cdecls t c =
let scdecl = handle_cdecl c in
(fst scdecl :: fst t, snd scdecl @ snd t)
in
let scdecl_list, func_list = List.fold_left iter_cdecls ([], []) cdecls in
let main = get_main func_list in
let funcs = remove_main func_list in
(* let funcs = (add_default_constructors cdecls class_maps) @ funcs in *)
{
        classes                 = scdecl_list;
        functions                 = funcs;
        main                        = main;
        reserved                  = reserved;
}

```

```
let add_reserved_functions =
let reserved_stub name return_type formals =
{
        sfname                          = FName(name);
        sreturnType          = return_type;
        sformals                   = formals;
        sbody                           = [];
        func_type                  = Sast.Reserved;
        overrides                   = false;
        source                          = "NA";
}
in
let i32_t = Datatype(Int_t) in
let void_t = Datatype(Void_t) in
let str_t = Arraytype(Char_t, 1) in
let mf t n = Formal(t, n) in (* Make formal *)
let reserved = [
reserved_stub "print"         (void_t)         ([Many(Any)]);
reserved_stub "malloc"         (str_t)         ([mf i32_t "size"]);
reserved_stub "cast"         (Any)                 ([mf Any "in"]);
reserved_stub "sizeof"         (i32_t)         ([mf Any "in"]);
reserved_stub "open"         (i32_t)         ([mf str_t "path"; mf i32_t "flags"]);
reserved_stub "close"         (i32_t)         ([mf i32_t "fd"]);
reserved_stub "read"         (i32_t)         ([mf i32_t "fd"; mf str_t "buf"; mf i32_t "nbyte"]);
reserved_stub "write"         (i32_t)         ([mf i32_t "fd"; mf str_t "buf"; mf i32_t "nbyte"]);
reserved_stub "lseek"         (i32_t)         ([mf i32_t "fd"; mf i32_t "offset"; mf i32_t "whence"]);
reserved_stub "exit"         (void_t)         ([mf i32_t "status"]);
reserved_stub "getchar" (i32_t)            ([]);
reserved_stub "input"         (str_t)         ([]);
] in
reserved

let build_inheritance_forest cdecls cmap =
let handler a cdecl =
match cdecl.extends with
Parent(s)         ->
let new_list = if (StringMap.mem s a) then
cdecl.cname::(StringMap.find s a)
else
[cdecl.cname]
in
Hashtbl.add predecessors s new_list;
(StringMap.add s new_list a)
|         NoParent         -> a
in
let forest = List.fold_left handler StringMap.empty cdecls in
```

```

let handler key value =
if not (StringMap.mem key cmap) then
raise (Exceptions.UndefinedClass key)
in
ignore(StringMap.iter handler forest);
forest

let merge_maps m1 m2 =
StringMap.fold (fun k v a -> StringMap.add k v a) m1 m2

let update_class_maps map_type cmap_val cname cmap_to_update =
let update m map_type =
if map_type = "field_map" then
{
        field_map = cmap_val;
        func_map = m.func_map;
        constructor_map = m.constructor_map;
        reserved_map = m.reserved_map;
        cdecl = m.cdecl;
}
else m
in
let updated = StringMap.find cname cmap_to_update in
let updated = update updated map_type in
let updated = StringMap.add cname updated cmap_to_update in
updated

let inherit_fields class_maps predecessors =
(* Get basic inheritance map *)
let add_key key pred map = StringMap.add key pred map in
let cmaps_inherit = StringMap.fold add_key class_maps StringMap.empty in
(* Perform accumulation of child classes *)
let add_key key pred maps =
let elem1 = StringSet.add key (fst maps) in
let accum acc child = StringSet.add child acc in
let elem2 = List.fold_left (accum) (snd maps) pred in
(elem1, elem2)
in
let empty_s = StringSet.empty in
let res = StringMap.fold add_key predecessors (empty_s, empty_s) in
let roots = StringSet.diff (fst res) (snd res) in
(*in let _ = print_set_members roots*)
let rec add_inherited_fields predec desc cmap_to_update =
let cmap_inherit accum descendant =
let predec_field_map = (StringMap.find predec accum).field_map in
let desc_field_map = (StringMap.find descendant accum).field_map in
let merged = merge_maps predec_field_map desc_field_map in
let updated = update_class_maps "field_map" merged descendant accum in
```

```
if (StringMap.mem descendant predecessors) then
let descendants_of_descendant = StringMap.find descendant predecessors in
add_inherited_fields descendant descendants_of_descendant updated
else updated
in
List.fold_left cmap_inherit cmap_to_update desc
(* end of add_inherited_fields *)
in
let result = StringSet.fold (fun x a -> add_inherited_fields x (StringMap.find x
↪  predecessors) a) roots cmaps_inherit
(*in let _ = print_map result*)
in result

(* TODO Check that this actually works *)
let check_cyclical_inheritance cdecls predecessors =
let handle_predecessor cdecl parent predecessor =
if cdecl.cname = predecessor then
raise(Exceptions.CyclicalDependencyBetween(cdecl.cname, parent))
in
let handle_cdecl cdecl =
if StringMap.mem cdecl.cname predecessors
then
let pred_list = StringMap.find cdecl.cname predecessors in
List.iter (handle_predecessor cdecl (List.hd pred_list)) pred_list
else ()
in
List.iter handle_cdecl cdecls

let build_func_map_inherited_lookup cdecls_inherited =
let build_func_map cdecl =
let add_func m fdecl = StringMap.add (get_name cdecl.cname fdecl) fdecl m in
List.fold_left add_func StringMap.empty cdecl.cbody.methods
in
let add_class_func_map m cdecl = StringMap.add cdecl.cname (build_func_map cdecl) m in
List.fold_left add_class_func_map StringMap.empty cdecls_inherited

let add_inherited_methods cmaps cdecls func_maps_inherited =
let find_cdecl cname =
try List.find (fun cdecl -> cdecl.cname = cname) cdecls
with | Not_found -> raise Not_found
in
let update_with_inherited_methods cname cmap =
let fmap = StringMap.find cname func_maps_inherited in
let cdecl = find_cdecl cname in
{
        field_map = cmap.field_map;
        func_map = fmap;
        constructor_map = cmap.constructor_map;
        reserved_map = cmap.reserved_map;
```

```
        cdecl = cdecl;
}
in
let add_updated_cmap cname cmap accum = StringMap.add cname (update_with_inherited_methods cname cmap) accum in
StringMap.fold add_updated_cmap cmaps StringMap.empty

let handle_inheritance cdecls class_maps =
let predecessors = build_inheritance_forest cdecls class_maps in
let cdecls_inherited = inherit_fields_cdecls cdecls predecessors in
let func_maps_inherited = build_func_map_inherited_lookup cdecls_inherited in
ignore(check_cyclical_inheritance cdecls predecessors);
let cmaps_with_inherited_fields = inherit_fields class_maps predecessors in
let cmaps_inherited = add_inherited_methods cmaps_with_inherited_fields cdecls_inherited func_maps_inherited in
cmaps_inherited, cdecls_inherited

let generate_struct_indexes cdecls =
let cdecl_handler index cdecl =
Hashtbl.add struct_indexes cdecl.cname index
in
List.iteri cdecl_handler cdecls

(* Main method for analyzer *)
let analyze filename program = match program with
Program(includes, classes) ->
(* Include code from external files *)
let cdecls = process_includes filename includes classes in
ignore(generate_struct_indexes cdecls);

(* Add built-in functions *)
let reserved = add_reserved_functions in
(* Generate the class_maps for look up in checking functions *)
let class_maps = build_class_maps reserved cdecls in
let class_maps, cdecls = handle_inheritance cdecls class_maps in
let sast = convert_cdecls_to_sast class_maps reserved cdecls in
sast
```

ast.ml

```
type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq | And | Not | Or | Mod
type scope = Private | Public
type primitive = Int_t | Float_t | Void_t | Bool_t | Char_t | Objecttype of string | ConstructorType | Null_t
type datatype = Arraytype of primitive * int | Datatype of primitive | Any

type extends = NoParent | Parent of string
type fname = Constructor | FName of string
type formal = Formal of datatype * string | Many of datatype

type expr =
Int_Lit of int
|         Boolean_Lit of bool
|         Float_Lit of float
|         String_Lit of string
|         Char_Lit of char
|         This
|         Id of string
|         Binop of expr * op * expr
|         Assign of expr * expr
|         Noexpr
|         ArrayCreate of datatype * expr list
|         ArrayAccess of expr * expr list
|         ObjAccess of expr * expr
|         Call of string * expr list
|   ObjectCreate of string * expr list
|         ArrayPrimitive of expr list
|          Unop of op * expr
|         Null
|         Delete of expr

type stmt =
Block of stmt list
|         Expr of expr
|         Return of expr
|         If of expr * stmt * stmt
|         For of expr * expr * expr * stmt
|         While of expr * stmt
|          Break
|   Continue
|   Local of datatype * string * expr

type field = Field of scope * datatype * string
type include_stmt = Include of string

type func_decl = {
```

```
        scope : scope;
        fname : fname;
        returnType : datatype;
        formals : formal list;
        body : stmt list;
        overrides : bool;
        root_cname : string option;
}

type cbody = {
        fields : field list;
        constructors : func_decl list;
        methods : func_decl list;
}

type class_decl = {
        cname : string;
        extends : extends;
        cbody: cbody;
}

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
```

```c
        for(int i = 0; i < dims[dim_curr]; i++) {
                int offset = (static_offset + (dynamic_offset + i) * (dims[dim_curr + 1] + 1));

                long* sub = arr + offset;
                arr[curr_offset + 1 + i] = (long) sub;

                indexes[dim_curr] = i;
                rec_init(arr, offset, static_offsets, indexes, dims, dimc, dim_curr + 1);
        }
}

long* init_arr(int* dims, int dimc) {

        int static_offsets[dimc];
        int total = 0;
        for(int i = 0; i < dimc; i++) {
                static_offsets[i] = 1;
                for(int j = 0; j < i; j++) {
                        static_offsets[i] *= dims[j];
                }
                static_offsets[i] *= dims[i] + 1;
                static_offsets[i] += total;
                total = static_offsets[i];
        }

        int indexes[dimc];
        for(int i = 0; i < dimc; i++) {
                indexes[i] = 0;
        }

        //Get total length of array
        int length = 0;
        for(int i = 0; i < dimc; i++) {
                int tmp = 1;
                for(int j = i - 1; j >= 0; j--) {
                        tmp *= dims[j];
                }
                tmp *= dims[i] + 1;
                length += tmp;
        }

        //Malloc array
        long* arr = malloc(length);

        //Set all values to 0 initially
        for(int i = 0 ; i < length; i++) {
                arr[i] = 0;
        }
```

```

        //Initialize the entire array
        rec_init(arr, 0, static_offsets, indexes, dims, dimc, 0);

        return arr;
}

// int main() {

        //          //Array creation
        //          int dims[5] = {2, 3, 4, 5, 6};
        //          int dimc = 5;

        //          long* arr = init_arr(dims, dimc);

        //          //Get total length of array
        //          int length = 0;
        //          for(int i = 0; i < dimc; i++) {
                //                  int tmp = 1;
                //                  for(int j = i - 1; j >= 0; j--) {
                        //                          tmp *= dims[j];
                        //                  }
                //                  tmp *= dims[i] + 1;
                //                  length += tmp;
                //          }

        //          for(int i = 0; i < length; i++) {
                //                  printf("val: %ld | addr: %ld\n", arr[i], (long) arr + i);
                //          }
        //          printf("\n");
        // }
```

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)))
```

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

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

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

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

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

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

```
(* |            (Datatype(Bool_t), Datatype(Float_t))                           -> (const_uitofp lhs f_t, rhs) *)

(* float to,__) ( using fptosi for signed ints *)
|   (Datatype(Float_t), Datatype(Int_t))                          -> (lhs, build_sitofp rhs f_t "tmp" llbuilder), Datatype(Float_t)
(* |          (Datatype(Float_t), Datatype(Char_t))                          -> (lhs, const_uitofp rhs f_t) *)
(* |            (Datatype(Float_t), Datatype(Bool_t))                          -> (lhs, const_uitofp rhs f_t) *)
|   (Datatype(Float_t), Datatype(Float_t))                          -> (lhs, rhs), Datatype(Float_t)

| Datatype(Objecttype(d)), Datatype(Null_t)                          -> (lhs, rhs), lhsType
| Datatype(Null_t), Datatype(Objecttype(d))                     -> (rhs, lhs), rhsType
| Datatype(Objecttype(d)), t                                          -> raise(Exceptions.CanOnlyCompareObjectsWithNull(d, (Utils.string_of_datatype t)))

| Arraytype(d, s), Datatype(Null_t)                                   -> (lhs, rhs), lhsType
| Datatype(Null_t), Arraytype(d, s)                                 -> (rhs, lhs), rhsType
| Arraytype(d, _), t                                                          -> raise(Exceptions.CanOnlyCompareArraysWithNull(Utils.string_of_primitive d, (Utils.string_of_datatype t)))

|          _                                                                                -> raise (Exceptions.CannotCastTypeException(Utils.string_of_datatype lhsType, Utils.string_of_datatype rhsType))

let rec handle_binop e1 op e2 d llbuilder =
(* Get the types of e1 and e2 *)
let type1 = Analyzer.get_type_from_sexpr e1 in
let type2 = Analyzer.get_type_from_sexpr e2 in

(* Generate llvalues from e1 and e2 *)

let e1 = codegen_sexpr llbuilder e1 in
let e2 = codegen_sexpr llbuilder e2 in

let float_ops op e1 e2 =
match op with
Add                    -> build_fadd e1 e2 "flt_addtmp" llbuilder
|         Sub                    -> build_fsub e1 e2 "flt_subtmp" llbuilder
|         Mult                    -> build_fmul e1 e2 "flt_multmp" llbuilder
|         Div                    -> build_fdiv e1 e2 "flt_divtmp" llbuilder
|         Mod                    -> build_frem e1 e2 "flt_sremtmp" llbuilder
|         Equal                    -> build_fcmp Fcmp.Oeq e1 e2 "flt_eqtmp" llbuilder
```

```
|         Neq                   -> build_fcmp Fcmp.One e1 e2 "flt_neqtmp" llbuilder
|         Less                   -> build_fcmp Fcmp.Ult e1 e2 "flt_lesstmp" llbuilder
|         Leq                   -> build_fcmp Fcmp.Ole e1 e2 "flt_leqtmp" llbuilder
|         Greater                 -> build_fcmp Fcmp.Ogt e1 e2 "flt_sgttmp" llbuilder
|         Geq                   -> build_fcmp Fcmp.Oge e1 e2 "flt_sgetmp" llbuilder
|         _                        -> raise Exceptions.FloatOpNotSupported

in

(* chars are considered ints, so they will use int_ops as well*)
let int_ops op e1 e2 =
match op with
Add                 -> build_add e1 e2 "addtmp" llbuilder
|         Sub                   -> build_sub e1 e2 "subtmp" llbuilder
|         Mult                   -> build_mul e1 e2 "multmp" llbuilder
|         Div                   -> build_sdiv e1 e2 "divtmp" llbuilder
|         Mod                   -> build_srem e1 e2 "sremtmp" llbuilder
|         Equal                   -> build_icmp Icmp.Eq e1 e2 "eqtmp" llbuilder
|         Neq                   -> build_icmp Icmp.Ne e1 e2 "neqtmp" llbuilder
|         Less                   -> build_icmp Icmp.Slt e1 e2 "lesstmp" llbuilder
|         Leq                   -> build_icmp Icmp.Sle e1 e2 "leqtmp" llbuilder
|         Greater                 -> build_icmp Icmp.Sgt e1 e2 "sgttmp" llbuilder
|         Geq                   -> build_icmp Icmp.Sge e1 e2 "sgetmp" llbuilder
|         And                   -> build_and e1 e2 "andtmp" llbuilder
|         Or                          -> build_or  e1 e2 "ortmp" llbuilder
|         _                        -> raise Exceptions.IntOpNotSupported
in

let obj_ops op e1 e2 =
match op with
Equal -> build_is_null e1 "tmp" llbuilder
|         Neq -> build_is_not_null e1 "tmp" llbuilder
|         _          -> raise (Exceptions.ObjOpNotSupported(Utils.string_of_op op))
in

let (e1, e2), d = cast e1 e2 type1 type2 llbuilder in

let type_handler d = match d with
Datatype(Float_t)   -> float_ops op e1 e2
|        Datatype(Int_t)
|   Datatype(Bool_t)
|         Datatype(Char_t)          -> int_ops op e1 e2
|         Datatype(Objecttype(_))
|         Arraytype(_, _) -> obj_ops op e1 e2
|   _ -> raise Exceptions.InvalidBinopEvaluationType
in

type_handler d
```

```
and handle_unop op e d llbuilder =
(* Get the type of e *)
let eType = Analyzer.get_type_from_sexpr e in
(* Get llvalue  *)
let e = codegen_sexpr llbuilder e in

let unops op eType e = match (op, eType) with
(Sub, Datatype(Int_t))                    ->  build_neg e "int_unoptmp" llbuilder
|   (Sub, Datatype(Float_t))         ->         build_fneg e "flt_unoptmp" llbuilder
|   (Not, Datatype(Bool_t))         ->  build_not e "bool_unoptmp" llbuilder
|    _          -> raise Exceptions.UnopNotSupported        in

let unop_type_handler d = match d with
Datatype(Float_t)
|        Datatype(Int_t)
|   Datatype(Bool_t)        -> unops op eType e
|   _ -> raise Exceptions.InvalidUnopEvaluationType
in

unop_type_handler d

and func_lookup fname =
match (lookup_function fname the_module) with
None         -> raise (Exceptions.LLVMFunctionNotFound fname)
|          Some f         -> f

and codegen_print el llbuilder =
let printf = func_lookup "printf" in
let tmp_count = ref 0 in
let incr_tmp = fun x -> incr tmp_count in

let map_expr_to_printfexpr expr =
let exprType = Analyzer.get_type_from_sexpr expr in
match exprType with
Datatype(Bool_t) ->
incr_tmp ();
let tmp_var = "tmp" ^ (string_of_int !tmp_count) in
let trueStr = SString_Lit("true") in
let falseStr = SString_Lit("false") in
let id = SId(tmp_var, str_type) in
ignore(codegen_stmt llbuilder (SLocal(str_type, tmp_var, SNoexpr)));
ignore(codegen_stmt llbuilder (SIf(expr,
SExpr(SAssign(id, trueStr, str_type), str_type),
SExpr(SAssign(id, falseStr, str_type), str_type)
)));
codegen_sexpr llbuilder id
| _ -> codegen_sexpr llbuilder expr
in
```

```
let params = List.map map_expr_to_printfexpr el in
let param_types = List.map (Analyzer.get_type_from_sexpr) el in

let map_param_to_string = function
Arraytype(Char_t, 1)         -> "%s"
|         Datatype(Int_t)                 -> "%d"
|         Datatype(Float_t)                 -> "%f"
|         Datatype(Bool_t)                 -> "%s"
|         Datatype(Char_t)                 -> "%c"
|         _                                                  -> raise
↪  (Exceptions.InvalidTypePassedToPrintf)
in
let const_str = List.fold_left (fun s t -> s ^ map_param_to_string t) "" param_types in
let s = codegen_sexpr llbuilder (SString_Lit(const_str)) in
let zero = const_int i32_t 0 in
let s = build_in_bounds_gep s [| zero |] "tmp" llbuilder in
build_call printf (Array.of_list (s :: params)) "tmp" llbuilder

and codegen_func_call fname el d llbuilder =
let f = func_lookup fname in
let params = List.map (codegen_sexpr llbuilder) el in
match d with
Datatype(Void_t) -> build_call f (Array.of_list params) "" llbuilder
|         _ ->                              build_call f (Array.of_list params) "tmp"
↪  llbuilder

and codegen_sizeof el llbuilder =
let type_of = Analyzer.get_type_from_sexpr (List.hd el) in
let type_of = get_type type_of in
let size_of = size_of type_of in
build_bitcast size_of i32_t "tmp" llbuilder

and codegen_cast el d llbuilder =
let cast_malloc_to_objtype lhs currType newType llbuilder = match newType with
Datatype(Objecttype(x)) ->
let obj_type = get_type (Datatype(Objecttype(x))) in
build_pointercast lhs obj_type "tmp" llbuilder
|         _ as t -> raise (Exceptions.CannotCastTypeException(Utils.string_of_datatype
↪  currType, Utils.string_of_datatype t))
in
let expr = List.hd el in
let t = Analyzer.get_type_from_sexpr expr in
let lhs = match expr with
|         Sast.SId(id, d) -> codegen_id false false id d llbuilder
|          SObjAccess(e1, e2, d) -> codegen_obj_access false e1 e2 d llbuilder
|         SArrayAccess(se, sel, d) -> codegen_array_access true se sel d llbuilder
| _ -> codegen_sexpr llbuilder expr
in
cast_malloc_to_objtype lhs t d llbuilder
```

```

and codegen_call llbuilder d el = function
"print"         -> codegen_print el llbuilder
|         "sizeof"        -> codegen_sizeof el llbuilder
|         "cast"                -> codegen_cast el d llbuilder
|         "malloc"         -> codegen_func_call "malloc" el d llbuilder
|         "open"                -> codegen_func_call "open" el d llbuilder
|         "write"               -> codegen_func_call "write" el d llbuilder
|         "close"               -> codegen_func_call "close" el d llbuilder
|         "read"                -> codegen_func_call "read" el d llbuilder
|         "lseek"        -> codegen_func_call "lseek" el d llbuilder
|         "exit"                -> codegen_func_call "exit" el d llbuilder
|         "input"        -> codegen_func_call "input" el d llbuilder
|   "getchar"   -> codegen_func_call "getchar" el d llbuilder
|         _ as fname         -> raise (Exceptions.UnableToCallFunctionWithoutParent fname)(* codegen_func_call fname el llbuilder *)

and codegen_id isDeref checkParam id d llbuilder =
if isDeref then
try Hashtbl.find named_params id
with | Not_found ->
try let _val = Hashtbl.find named_values id in
build_load _val id llbuilder
with | Not_found -> raise (Exceptions.UnknownVariable id)
else
try Hashtbl.find named_values id
with | Not_found ->
try
let _val = Hashtbl.find named_params id in
if checkParam then raise (Exceptions.CannotAssignParam id)
else _val
with | Not_found -> raise (Exceptions.UnknownVariable id)

and codegen_assign lhs rhs d llbuilder =
let rhsType = Analyzer.get_type_from_sexpr rhs in
(* Special case '=' because we don't want to emit the LHS as an
* expression. *)
let lhs, isObjAccess = match lhs with
|         Sast.SId(id, d) -> codegen_id false false id d llbuilder, false
|          SObjAccess(e1, e2, d) -> codegen_obj_access false e1 e2 d llbuilder, true
|         SArrayAccess(se, sel, d) -> codegen_array_access true se sel d llbuilder, true
| _ -> raise Exceptions.AssignLHSMustBeAssignable
in
(* Codegen the rhs. *)
let rhs = match rhs with
|         Sast.SId(id, d) -> codegen_id false false id d llbuilder
|          SObjAccess(e1, e2, d) -> codegen_obj_access true e1 e2 d llbuilder
| _ -> codegen_sexpr llbuilder rhs
in
```

```
let rhs = match d with
Datatype(Objecttype(_))        ->
if isObjAccess then rhs
else build_load rhs "tmp" llbuilder
|         Datatype(Null_t) -> const_null (get_type d)
| _ -> rhs
in
let rhs = match d, rhsType with
Datatype(Char_t), Datatype(Int_t) -> build_uitofp rhs i8_t "tmp" llbuilder
|         Datatype(Int_t), Datatype(Char_t) -> build_uitofp rhs i32_t "tmp" llbuilder
|         _ -> rhs
in
(* Lookup the name. *)
ignore(build_store rhs lhs llbuilder);
rhs

and deref ptr t llbuilder =
build_gep ptr (Array.of_list [ptr]) "tmp" llbuilder

and codegen_obj_access isAssign lhs rhs d llbuilder =
let codegen_func_call param_ty fptr parent_expr el d llbuilder =
let match_sexpr se = match se with
SId(id, d) -> let isDeref = match d with
Datatype(Objecttype(_)) -> false
|         _ -> true
in codegen_id isDeref false id d llbuilder
|         se -> codegen_sexpr llbuilder se
in
let parent_expr = build_pointercast parent_expr param_ty "tmp" llbuilder in
let params = List.map match_sexpr el in
match d with
Datatype(Void_t) -> build_call fptr (Array.of_list (parent_expr :: params)) "" llbuilder
|         _ -> build_call fptr (Array.of_list (parent_expr :: params)) "tmp" llbuilder
in
let check_lhs = function
SId(s, d)                              -> codegen_id false false s d llbuilder
|         SArrayAccess(e, el, d)         -> codegen_array_access false e el d llbuilder
|         se          -> raise (Exceptions.LHSofRootAccessMustBeIDorFunc
↪  (Utils.string_of_sexpr se))
in
(* Needs to be changed *)
let rec check_rhs isLHS parent_expr parent_type =
let parent_str = Utils.string_of_object parent_type in
function
(* Check fields in parent *)
SId(field, d) ->
let search_term = (parent_str ^ "." ^ field) in
let field_index = Hashtbl.find struct_field_indexes search_term in
let _val = build_struct_gep parent_expr field_index field llbuilder in
```

```
let _val = match d with
Datatype(Objecttype(_)) ->
if not isAssign then _val
else build_load _val field llbuilder
| _ ->
if not isAssign then
_val
else
build_load _val field llbuilder
in
_val

|         SArrayAccess(e, el, d) ->

let ce = check_rhs false parent_expr parent_type e in
let index = codegen_sexpr llbuilder (List.hd el) in
let index = match d with
Datatype(Char_t) -> index
|         _ -> build_add index (const_int i32_t 1) "tmp" llbuilder
in
let _val = build_gep ce [| index |] "tmp" llbuilder in
if isLHS && isAssign
then _val
else build_load _val "tmp" llbuilder

(* Check functions in parent *)
|         SCall(fname, el, d, index)         ->
let index = const_int i32_t index in
let c_index = build_struct_gep parent_expr 0 "cindex" llbuilder in
let c_index = build_load c_index "cindex" llbuilder in
let lookup = func_lookup "lookup" in
let fptr = build_call lookup [| c_index; index |] "fptr" llbuilder in
let fptr2 = func_lookup fname in
let f_ty = type_of fptr2 in
let param1 = param fptr2 0 in
let param_ty = type_of param1 in
let fptr = build_pointercast fptr f_ty fname llbuilder in
let ret = codegen_func_call param_ty fptr parent_expr el d llbuilder in
let ret = ret
(* if not isLHS && not isAssign then
build_load ret "tmp" llbuilder
else
ret *)
in
ret
(* Set parent, check if base is field *)
|         SObjAccess(e1, e2, d)         ->
let e1_type = Analyzer.get_type_from_sexpr e1 in
let e1 = check_rhs true parent_expr parent_type e1 in
```

```
let e2 = check_rhs true e1 e1_type e2 in
e2
|          _ as e -> raise (Exceptions.InvalidAccessLHS (Utils.string_of_sexpr e))
in
let lhs_type = Analyzer.get_type_from_sexpr lhs in
match lhs_type with
Arraytype(_, _) ->
let lhs = codegen_sexpr llbuilder lhs in
let _ = match rhs with
SId("length", _) -> "length"
|          _ -> raise(Exceptions.CanOnlyAccessLengthOfArray)
in
let _val = build_gep lhs [| (const_int i32_t 0) |] "tmp" llbuilder in
build_load _val "tmp" llbuilder
|          _ ->
let lhs = check_lhs lhs in
let rhs = check_rhs true lhs lhs_type rhs in
rhs

and codegen_obj_create fname el d llbuilder =
let f = func_lookup fname in
let params = List.map (codegen_sexpr llbuilder) el in
let obj = build_call f (Array.of_list params) "tmp" llbuilder in
obj

and codegen_string_lit s llbuilder =
if s = "true" then build_global_stringptr "true" "tmp" llbuilder
else if s = "false" then build_global_stringptr "false" "tmp" llbuilder
else build_global_stringptr s "tmp" llbuilder

and codegen_array_access isAssign e el d llbuilder =
let index = codegen_sexpr llbuilder (List.hd el) in
let index = match d with
Datatype(Char_t) -> index
|          _ -> build_add index (const_int i32_t 1) "tmp" llbuilder
in
let arr = codegen_sexpr llbuilder e in
let _val = build_gep arr [| index |] "tmp" llbuilder in
if isAssign
then _val
else build_load _val "tmp" llbuilder

and initialise_array arr arr_len init_val start_pos llbuilder =
let new_block label =
let f = block_parent (insertion_block llbuilder) in
append_block (global_context ()) label f
in
let bbcurr = insertion_block llbuilder in
let bbcond = new_block "array.cond" in
```

```
let bbbody = new_block "array.init" in
let bbdone = new_block "array.done" in
ignore (build_br bbcond llbuilder);
position_at_end bbcond llbuilder;

(* Counter into the length of the array *)
let counter = build_phi [const_int i32_t start_pos, bbcurr] "counter" llbuilder in
add_incoming ((build_add counter (const_int i32_t 1) "tmp" llbuilder), bbbody) counter;
let cmp = build_icmp Icmp.Slt counter arr_len "tmp" llbuilder in
ignore (build_cond_br cmp bbbody bbdone llbuilder);
position_at_end bbbody llbuilder;

(* Assign array position to init_val *)
let arr_ptr = build_gep arr [| counter |] "tmp" llbuilder in
ignore (build_store init_val arr_ptr llbuilder);
ignore (build_br bbcond llbuilder);
position_at_end bbdone llbuilder

and codegen_array_create llbuilder t expr_type el =
if(List.length el > 1) then raise(Exceptions.ArrayLargerThan1Unsupported)
else
match expr_type with
Arraytype(Char_t, 1) ->
let e = List.hd el in
let size = (codegen_sexpr llbuilder e) in
let t = get_type t in
let arr = build_array_malloc t size "tmp" llbuilder in
let arr = build_pointercast arr (pointer_type t) "tmp" llbuilder in
(* initialise_array arr size (const_int i32_t 0) 0 llbuilder; *)
arr
|          _ ->
let e = List.hd el in
let t = get_type t in

(* This will not work for arrays of objects *)
let size = (codegen_sexpr llbuilder e) in
let size_t = build_intcast (size_of t) i32_t "tmp" llbuilder in
let size = build_mul size_t size "tmp" llbuilder in
let size_real = build_add size (const_int i32_t 1) "arr_size" llbuilder in

let arr = build_array_malloc t size_real "tmp" llbuilder in
let arr = build_pointercast arr (pointer_type t) "tmp" llbuilder in

let arr_len_ptr = build_pointercast arr (pointer_type i32_t) "tmp" llbuilder in

(* Store length at this position *)
ignore(build_store size_real arr_len_ptr llbuilder);
initialise_array arr_len_ptr size_real (const_int i32_t 0) 0 llbuilder;
arr
```

```
and codegen_array_prim d el llbuilder =
let t = d in
let size = (const_int i32_t ((List.length el))) in
let size_real = (const_int i32_t ((List.length el) + 1)) in
let t = get_type t in
let arr = build_array_malloc t size_real "tmp" llbuilder in
let arr = build_pointercast arr t "tmp" llbuilder in
let size_casted = build_bitcast size t "tmp" llbuilder in
ignore(if d = Arraytype(Char_t, 1) then ignore(build_store size_casted arr llbuilder););
↪  (* Store length at this position *)
(* initialise_array arr size_real (const_int i32_t 0) 1 llbuilder; *)

let llvalues = List.map (codegen_sexpr llbuilder) el in
List.iteri (fun i llval ->
let arr_ptr = build_gep arr [| (const_int i32_t (i+1)) |] "tmp" llbuilder in
ignore(build_store llval arr_ptr llbuilder);  ) llvalues;
arr

and codegen_delete e llbuilder =
let ce = match e with
SId(id, d) -> codegen_id false false id d llbuilder
|          _ -> codegen_sexpr llbuilder e
in
build_free ce llbuilder

and codegen_sexpr llbuilder = function
SInt_Lit(i)                       -> const_int i32_t i
|   SBoolean_Lit(b)                       -> if b then const_int i1_t 1 else const_int
↪  i1_t 0
|   SFloat_Lit(f)                         -> const_float f_t f
|   SString_Lit(s)                        -> codegen_string_lit s llbuilder
|   SChar_Lit(c)                          -> const_int i8_t (Char.code c)
|   SId(id, d)                      -> codegen_id true false id d llbuilder
|   SBinop(e1, op, e2, d)           -> handle_binop e1 op e2 d llbuilder
|   SAssign(e1, e2, d)              -> codegen_assign e1 e2 d llbuilder
|   SNoexpr                       -> build_add (const_int i32_t 0) (const_int i32_t 0)
↪  "nop" llbuilder
|   SArrayCreate(t, el, d)          -> codegen_array_create llbuilder t d el
|   SArrayAccess(e, el, d)          -> codegen_array_access false e el d llbuilder
|   SObjAccess(e1, e2, d)           -> codegen_obj_access true e1 e2 d llbuilder
|   SCall(fname, el, d, _)             -> codegen_call llbuilder d el fname
|   SObjectCreate(id, el, d)        -> codegen_obj_create id el d llbuilder
|   SArrayPrimitive(el, d)          -> codegen_array_prim d el llbuilder
|   SUnop(op, e, d)                 -> handle_unop op e d llbuilder
|   SNull                                 -> const_null i32_t
|         SDelete e                                           -> codegen_delete e
↪  llbuilder
```

```
and codegen_if_stmt exp then_ (else_:Sast.sstmt) llbuilder =
let cond_val = codegen_sexpr llbuilder exp in

(* Grab the first block so that we might later add the conditional branch
* to it at the end of the function. *)
let start_bb = insertion_block llbuilder in
let the_function = block_parent start_bb in

let then_bb = append_block context "then" the_function in

(* Emit 'then' value. *)
position_at_end then_bb llbuilder;
let _(* then_val *) = codegen_stmt llbuilder then_ in

(* Codegen of 'then' can change the current block, update then_bb for the
* phi. We create a new name because one is used for the phi node, and the
* other is used for the conditional branch. *)
let new_then_bb = insertion_block llbuilder in

(* Emit 'else' value. *)
let else_bb = append_block context "else" the_function in
position_at_end else_bb llbuilder;
let _ (* else_val *) = codegen_stmt llbuilder else_ in

(* Codegen of 'else' can change the current block, update else_bb for the
* phi. *)
let new_else_bb = insertion_block llbuilder in


let merge_bb = append_block context "ifcont" the_function in
position_at_end merge_bb llbuilder;
(* let then_bb_val = value_of_block new_then_bb in *)
let else_bb_val = value_of_block new_else_bb in
(* let incoming = [(then_bb_val, new_then_bb); (else_bb_val, new_else_bb)] in *)
(* let phi = build_phi incoming "iftmp" llbuilder in *)

(* Return to the start block to add the conditional branch. *)
position_at_end start_bb llbuilder;
ignore (build_cond_br cond_val then_bb else_bb llbuilder);

(* Set a unconditional branch at the end of the 'then' block and the
* 'else' block to the 'merge' block. *)
position_at_end new_then_bb llbuilder; ignore (build_br merge_bb llbuilder);
position_at_end new_else_bb llbuilder; ignore (build_br merge_bb llbuilder);

(* Finally, set the builder to the end of the merge block. *)
position_at_end merge_bb llbuilder;

else_bb_val (* phi *)
```

```
and codegen_for init_ cond_ inc_ body_ llbuilder =
let old_val = !is_loop in
is_loop := true;

let the_function = block_parent (insertion_block llbuilder) in

(* Emit the start code first, without 'variable' in scope. *)
let _ = codegen_sexpr llbuilder init_ in

(* Make the new basic block for the loop header, inserting after current
* block. *)
let loop_bb = append_block context "loop" the_function in
(* Insert maintenance block *)
let inc_bb = append_block context "inc" the_function in
(* Insert condition block *)
let cond_bb = append_block context "cond" the_function in
(* Create the "after loop" block and insert it. *)
let after_bb = append_block context "afterloop" the_function in

let _ = if not old_val then
cont_block := inc_bb;
br_block := after_bb;
in

(* Insert an explicit fall through from the current block to the
* loop_bb. *)
ignore (build_br cond_bb llbuilder);

(* Start insertion in loop_bb. *)
position_at_end loop_bb llbuilder;

(* Emit the body of the loop.  This, like any other expr, can change the
* current BB.  Note that we ignore the value computed by the body, but
* don't allow an error *)
ignore (codegen_stmt llbuilder body_);

let bb = insertion_block llbuilder in
move_block_after bb inc_bb;
move_block_after inc_bb cond_bb;
move_block_after cond_bb after_bb;
ignore(build_br inc_bb llbuilder);

(* Start insertion in loop_bb. *)
position_at_end inc_bb llbuilder;
(* Emit the step value. *)
let _ = codegen_sexpr llbuilder inc_ in
ignore(build_br cond_bb llbuilder);
```

```
position_at_end cond_bb llbuilder;

let cond_val = codegen_sexpr llbuilder cond_ in
ignore (build_cond_br cond_val loop_bb after_bb llbuilder);

(* Any new code will be inserted in after_bb. *)
position_at_end after_bb llbuilder;

is_loop := old_val;

(* for expr always returns 0.0. *)
const_null f_t

and codegen_while cond_ body_ llbuilder =
let null_sexpr = SInt_Lit(0) in
codegen_for null_sexpr cond_ null_sexpr body_ llbuilder

and codegen_alloca datatype var_name expr llbuilder =
let t = match datatype with
Datatype(Objecttype(name)) -> find_struct name
|           _ -> get_type datatype
in
let alloca = build_alloca t var_name llbuilder in
Hashtbl.add named_values var_name alloca;
let lhs = SId(var_name, datatype) in
match expr with
SNoexpr -> alloca
|           _ -> codegen_assign lhs expr datatype llbuilder

and codegen_ret d expr llbuilder =
match expr with
SId(name, d) ->
(match d with
| Datatype(Objecttype(_)) -> build_ret (codegen_id false false name d llbuilder) llbuilder
| _ -> build_ret (codegen_id true true name d llbuilder) llbuilder)
| SObjAccess(e1, e2, d) -> build_ret (codegen_obj_access true e1 e2 d llbuilder) llbuilder
| SNoexpr -> build_ret_void llbuilder
| _ -> build_ret (codegen_sexpr llbuilder expr) llbuilder

and codegen_break llbuilder =
let block = fun () -> !br_block in
build_br (block ()) llbuilder

and codegen_continue llbuilder =
let block = fun () -> !cont_block in
build_br (block ()) llbuilder
```

```
and codegen_stmt llbuilder = function
SBlock sl                          -> List.hd(List.map (codegen_stmt llbuilder) sl)
|   SExpr(e, d)                    -> codegen_sexpr llbuilder e
|   SReturn(e, d)                      -> codegen_ret d e llbuilder
|   SIf (e, s1, s2)                 -> codegen_if_stmt e s1 s2 llbuilder
|   SFor (e1, e2, e3, s)            -> codegen_for e1 e2 e3 s llbuilder
|   SWhile (e, s)                      -> codegen_while e s llbuilder
|   SBreak                             -> codegen_break llbuilder
|   SContinue                          -> codegen_continue llbuilder
|   SLocal(d, s, e)                    -> codegen_alloca d s e llbuilder

let codegen_funcstub sfdecl =
let fname = (Utils.string_of_fname sfdecl.sfname) in
let is_var_arg = ref false in
let params = List.rev (List.fold_left (fun l -> (function Formal(t, _) -> get_type t :: l
↪   | _ -> is_var_arg := true; l )) [] sfdecl.sformals) in
let fty = if !is_var_arg
then var_arg_function_type (get_type sfdecl.sreturnType) (Array.of_list params)
else function_type (get_type sfdecl.sreturnType) (Array.of_list params)
in
define_function fname fty the_module

let init_params f formals =
let formals = Array.of_list (formals) in
Array.iteri (fun i a ->
let n = formals.(i) in
let n = Utils.string_of_formal_name n in
set_value_name n a;
Hashtbl.add named_params n a;
) (params f)

let codegen_func sfdecl =
Hashtbl.clear named_values;
Hashtbl.clear named_params;
let fname = (Utils.string_of_fname sfdecl.sfname) in
let f = func_lookup fname in
let llbuilder = builder_at_end context (entry_block f) in
let _ = init_params f sfdecl.sformals in
let _ = if sfdecl.overrides then
let this_param = Hashtbl.find named_params "this" in
let source = Datatype(Objecttype(sfdecl.source)) in
let casted_param = build_pointercast this_param (get_type source) "casted" llbuilder in
Hashtbl.replace named_params "this" casted_param;
in
let _ = codegen_stmt llbuilder (SBlock (sfdecl.sbody)) in
if sfdecl.sreturnType = Datatype(Void_t)
then ignore(build_ret_void llbuilder);
()
```

```
let codegen_vtbl scdecls =
let rt = pointer_type i64_t in
let void_pt = pointer_type i64_t in
let void_ppt = pointer_type void_pt in

let f = func_lookup "lookup" in
let llbuilder = builder_at_end context (entry_block f) in

let len = List.length scdecls in
let total_len = ref 0 in
let scdecl_llvm_arr = build_array_alloca void_ppt (const_int i32_t len) "tmp" llbuilder
  ↪ in

let handle_scdecl scdecl =
let index = Hashtbl.find Analyzer.struct_indexes scdecl.scname in
let len = List.length scdecl.sfuncs in
let sfdecl_llvm_arr = build_array_alloca void_pt (const_int i32_t len) "tmp" llbuilder in

let handle_fdecl i sfdecl =
let fptr = func_lookup (Utils.string_of_fname sfdecl.sfname) in
let fptr = build_pointercast fptr void_pt "tmp" llbuilder in

let ep = build_gep sfdecl_llvm_arr [| (const_int i32_t i) |] "tmp" llbuilder in
ignore(build_store fptr ep llbuilder);
in
List.iteri handle_fdecl scdecl.sfuncs;
total_len := !total_len + len;

let ep = build_gep scdecl_llvm_arr [| (const_int i32_t index) |] "tmp" llbuilder in
ignore(build_store sfdecl_llvm_arr ep llbuilder);
in
List.iter handle_scdecl scdecls;

let c_index = param f 0 in
let f_index = param f 1 in
set_value_name "c_index" c_index;
set_value_name "f_index" f_index;

if !total_len == 0 then
build_ret (const_null rt) llbuilder
else
let vtbl = build_gep scdecl_llvm_arr [| c_index |] "tmp" llbuilder in
let vtbl = build_load vtbl "tmp" llbuilder in
let fptr = build_gep vtbl [| f_index |] "tmp" llbuilder in
let fptr = build_load fptr "tmp" llbuilder in

build_ret fptr llbuilder

let codegen_library_functions () =
```

```
(* C Std lib functions *)
let printf_ty = var_arg_function_type i32_t [| pointer_type i8_t |] in
let _ = declare_function "printf" printf_ty the_module in
let malloc_ty = function_type (str_t) [| i32_t |] in
let _ = declare_function "malloc" malloc_ty the_module in
let open_ty = function_type i32_t [| (pointer_type i8_t); i32_t |] in
let _ = declare_function "open" open_ty the_module in
let close_ty = function_type i32_t [| i32_t |] in
let _ = declare_function "close" close_ty the_module in
let read_ty = function_type i32_t [| i32_t; pointer_type i8_t; i32_t |] in
let _ = declare_function "read" read_ty the_module in
let write_ty = function_type i32_t [| i32_t; pointer_type i8_t; i32_t |] in
let _ = declare_function "write" write_ty the_module in
let lseek_ty = function_type i32_t [| i32_t; i32_t; i32_t |] in
let _ = declare_function "lseek" lseek_ty the_module in
let exit_ty = function_type void_t [| i32_t |] in
let _ = declare_function "exit" exit_ty the_module in
let realloc_ty = function_type str_t [| str_t; i32_t |] in
let _ = declare_function "realloc" realloc_ty the_module in
let getchar_ty = function_type (i32_t) [| |] in
let _ = declare_function "getchar" getchar_ty the_module in

(* Dice defined functions *)
let fty = function_type (pointer_type i64_t) [| i32_t; i32_t |] in
let _ = define_function "lookup" fty the_module in
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
let _ = declare_function "rec_init" rec_init_ty the_module in
let init_arr_ty = function_type (pointer_type i64_t) [| (pointer_type i32_t); i32_t |] in
let _ = declare_function "init_arr" init_arr_ty the_module in
let input_ty = function_type str_t [||] in
let _ = declare_function "input" input_ty the_module in
()

let codegen_struct_stub s =
let struct_t = named_struct_type context s.scname in
Hashtbl.add struct_types s.scname struct_t

let codegen_struct s =
let struct_t = Hashtbl.find struct_types s.scname in
let type_list = List.map (function Field(_, d, _) -> get_type d) s.sfields in
let name_list = List.map (function Field(_, _, s) -> s) s.sfields in

(* Add key field to all structs *)
let type_list = i32_t :: type_list in
let name_list = ".key" :: name_list in

let type_array = (Array.of_list type_list) in
List.iteri (fun i f ->
```

```
let n = s.scname ^ "." ^ f in
Hashtbl.add struct_field_indexes n i;
) name_list;
struct_set_body struct_t type_array true

let init_args argv args argc llbuilder =
let new_block label =
let f = block_parent (insertion_block llbuilder) in
append_block (global_context ()) label f
in
let bbcurr = insertion_block llbuilder in
let bbcond = new_block "args.cond" in
let bbbody = new_block "args.init" in
let bbdone = new_block "args.done" in
ignore (build_br bbcond llbuilder);
position_at_end bbcond llbuilder;

(* Counter into the length of the array *)
let counter = build_phi [const_int i32_t 0, bbcurr] "counter" llbuilder in
add_incoming ((build_add counter (const_int i32_t 1) "tmp" llbuilder), bbbody) counter;
let cmp = build_icmp Icmp.Slt counter argc "tmp" llbuilder in
ignore (build_cond_br cmp bbbody bbdone llbuilder);
position_at_end bbbody llbuilder;

(* Assign array position to init_val *)
let arr_ptr = build_gep args [| counter |] "tmp" llbuilder in
let argv_val = build_gep argv [| counter |] "tmp" llbuilder in
let argv_val = build_load argv_val "tmp" llbuilder in
ignore (build_store argv_val arr_ptr llbuilder);
ignore (build_br bbcond llbuilder);
position_at_end bbdone llbuilder

let construct_args argc argv llbuilder =
let str_pt = pointer_type str_t in
let size_real = build_add argc (const_int i32_t 1) "arr_size" llbuilder in

let arr = build_array_malloc str_pt size_real "args" llbuilder in
let arr = build_pointercast arr str_pt "args" llbuilder in
let arr_len_ptr = build_pointercast arr (pointer_type i32_t) "argc_len" llbuilder in
let arr_1 = build_gep arr [| const_int i32_t 1 |] "arr_1" llbuilder in

(* Store length at this position *)
ignore(build_store argc arr_len_ptr llbuilder);
ignore(init_args argv arr_1 argc llbuilder);
arr

let codegen_main main =
Hashtbl.clear named_values;
Hashtbl.clear named_params;
```

```
let fty = function_type i32_t [| i32_t; pointer_type str_t |] in
let f = define_function "main" fty the_module in
let llbuilder = builder_at_end context (entry_block f) in

let argc = param f 0 in
let argv = param f 1 in
set_value_name "argc" argc;
set_value_name "argv" argv;
let args = construct_args argc argv llbuilder in
Hashtbl.add named_params "args" args;

let _ = codegen_stmt llbuilder (SBlock (main.sbody)) in
build_ret (const_int i32_t 0) llbuilder

let linker filename =
let llctx = Llvm.global_context () in
let llmem = Llvm.MemoryBuffer.of_file filename in
let llm = Llvm_bitreader.parse_bitcode llctx llmem in
ignore(Llvm_linker.link_modules the_module llm)

let codegen_sprogram sprogram =
let _ = codegen_library_functions () in
let _ = List.map (fun s -> codegen_struct_stub s) sprogram.classes in
let _ = List.map (fun s -> codegen_struct s) sprogram.classes in
let _ = List.map (fun f -> codegen_funcstub f) sprogram.functions in
let _ = List.map (fun f -> codegen_func f) sprogram.functions in
let _ = codegen_main sprogram.main in
let _ = codegen_vtbl sprogram.classes in
let _ = linker Conf.bindings_path in
the_module

(* Need to handle assignment of two different types *)
(* Need to handle private/public access *)
```

conf.ml

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

dice.ml

```
open Llvm
open Llvm_analysis
open Analyzer
open Utils
open Ast
open Yojson
open Exceptions
open Filepath

type action = Tokens | TokenEndl | PrettyPrint | Ast | Sast | Compile | CompileToFile | Help

let get_action = function
"-tendl"         -> TokenEndl
|         "-t"                  -> Tokens
|         "-p"                  -> PrettyPrint
|         "-ast"                  -> Ast
|         "-sast"          -> Sast
|         "-h"                  -> Help
|         "-c"                  -> Compile
|         "-f"                  -> CompileToFile
|          _ as s                  -> raise (Exceptions.InvalidCompilerArgument s)

let check_single_argument = function
"-h"         -> Help, ""
|         "-tendl"
|         "-t"
|         "-p"
|         "-ast"
|         "-sast"
|         "-c"
|         "-f"         -> raise (Exceptions.NoFileArgument)
|          _ as s          -> CompileToFile, s

let dice_name filename =
let basename = Filename.basename filename in
let filename = Filename.chop_extension basename in
filename ^ ".ll"

let help_string = (
"Usage: dice [optional-option] <source file>\n" ^
"optional-option:\n" ^
"\t-h: Print help text\n" ^
"\t-tendl: Prints tokens with newlines intact\n" ^
"\t-t: Prints token stream\n" ^
"\t-p: Pretty prints Ast as a program\n" ^
"\t-ast: Prints abstract syntax tree as json\n" ^
```

```
"\t-sast: Prints semantically checked syntax tree as json\n" ^
"\t-c: Compiles source\n" ^
"\t-f: Compiles source to file (<filename>.<ext> -> <filename>.ll)\n" ^
"Option defaults to \"-f\"\n"
)

let _ =
ignore(Printexc.record_backtrace true);
try
let action, filename =
if Array.length Sys.argv = 1 then
Help, ""
else if Array.length Sys.argv = 2 then
check_single_argument (Sys.argv.(1))
else if Array.length Sys.argv = 3 then
get_action Sys.argv.(1), Sys.argv.(2)
else raise (Exceptions.InvalidNumberCompilerArguments (Array.length Sys.argv))
in
(* Added fun () -> <x> so that each is evaluated only when requested *)
let filename         = Filepath.realpath filename in
let file_in         = fun () -> open_in filename in
let lexbuf                = fun () ->        Lexing.from_channel (file_in ()) in
let token_list          = fun () -> Processor.build_token_list (lexbuf ()) in
let program         = fun () -> Processor.parser filename (token_list ()) in
let sprogram         = fun () -> Analyzer.analyze filename (program ()) in
let llm                 = fun () -> Codegen.codegen_sprogram (sprogram ()) in
(* let _ = Llvm_analysis.assert_valid_module llm in *)
match action with
Help                        -> print_string help_string
|         Tokens                           -> print_string (Utils.token_list_to_string
↪  (token_list ()))
|         TokenEndl                   -> print_string (Utils.token_list_to_string_endl
↪  (token_list ()))
|         Ast                            -> print_string (pretty_to_string
↪  (Utils.print_tree (program ())))
|         Sast                            -> print_string (pretty_to_string
↪  (Utils.map_sprogram_to_json (sprogram ())))
|         PrettyPrint         -> print_string (Utils.string_of_program (program ()))
|         Compile                   -> dump_module (llm ())
|         CompileToFile         -> print_module (dice_name filename) (llm ())
with
Exceptions.IllegalCharacter(filename, c, ln) ->
print_string
(
"In \"" ^ filename ^ "\", Illegal Character, '" ^
Char.escaped c ^ "', line " ^ string_of_int ln ^ "\n"
)
|         Exceptions.UnmatchedQuotation(ln)         -> print_endline("Unmatched
↪  Quotation, line " ^ string_of_int ln)
```

```
|        Exceptions.IllegalToken(tok)                    -> print_endline("Illegal token " ^ tok)
|        Exceptions.MissingEOF                                     -> print_endline("Missing EOF")
|        Parsing.Parse_error ->
print_string
(
"File \"" ^ !Processor.filename ^ "\", " ^
"line " ^ string_of_int !Processor.line_number ^ ", " ^
"character " ^ string_of_int !Processor.char_num ^ ", " ^
"Syntax Error, token " ^ Utils.string_of_token !Processor.last_token ^ "\n"
)

|         Exceptions.InvalidNumberCompilerArguments i -> print_endline ("Invalid argument passed " ^ (string_of_int i)); print_string help_string
|        Exceptions.InvalidCompilerArgument s                 -> print_endline ("Invalid argument passed " ^ s); print_string help_string
|        Exceptions.NoFileArgument                                          -> print_string ("Must include file argument\n" ^ help_string)

|        Exceptions.IncorrectNumberOfArgumentsException                      -> print_endline("Incorrect number of arguments passed to function")
|        Exceptions.ConstructorNotFound(cname)                                         -> print_endline("Constructor" ^ cname ^ " not found")
|        Exceptions.DuplicateClassName(cname)                                     -> print_endline("Class " ^ cname ^ " not found")
|        Exceptions.DuplicateField                                                             -> print_endline("Duplicate field defined")
|        Exceptions.DuplicateFunction(fname)                                     -> print_endline("Duplicate function defined " ^ fname)
|        Exceptions.DuplicateConstructor                                                   -> print_endline("Duplicate constructor found")
|        Exceptions.DuplicateLocal(lname)                                                  -> print_endline("Duplicate local variable defined " ^ lname)
|        Exceptions.UndefinedClass(cname)                                                  -> print_endline("Undefined class " ^ cname)
|        Exceptions.UnknownIdentifier(id)                                                  -> print_endline("Unkown identifier " ^ id)
|        Exceptions.InvalidBinopExpression(binop)                                   -> print_endline("Invalid binary expression " ^ binop)
|        Exceptions.InvalidIfStatementType                                                 -> print_endline("Invalid type passed to if statement, must be bool")
```

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

```
|       Exceptions.ArrayAccessExpressionNotArray(a)                        -> print_endline("This expression is not an array " ^ a)
|       Exceptions.CanOnlyAccessLengthOfArray                                  -> print_endline("Can only access the length of an array")
|       Exceptions.CanOnlyDeleteObjectsOrArrays                              -> print_endline("Can only delete objects or arrays")
|       Exceptions.CannotAccessLengthOfCharArray                             -> print_endline("Cannot access the length of a char array")
|       Exceptions.AllNonVoidFunctionsMustEndWithReturn(f)                  -> print_endline("Non-void function " ^ f ^ " does not end in return")
|       Exceptions.CyclicalDependencyBetween(c1, c2)                        -> print_endline("Class " ^ c1 ^ " and " ^ c2 ^ " have a cylical dependence")
|       Exceptions.CannotAccessPrivateFieldInNonProperScope(f, cp, cc) -> print_endline("Cannot access private field " ^ f ^ " in scope " ^ cp ^ " from object " ^ cc)
|       Exceptions.CannotCallBreakOutsideOfLoop                              -> print_endline("Cannot call break outside of loop")
|       Exceptions.CannotCallContinueOutsideOfLoop                            -> print_endline("Cannot call continue outside of loop")
|       Exceptions.CannotAccessPrivateFunctionInNonProperScope(f, cp, cc) -> print_endline("Cannot access private function " ^ f ^ " in scope " ^ cp ^ " from object " ^ cc)
|       Exceptions.CannotPassNonInheritedClassesInPlaceOfOthers(c1, c2)        -> print_endline("Cannot pass non-inherited classe" ^ c1 ^ " to parameter " ^ c2)
|       Exceptions.IncorrectTypePassedToFunction(id, t)                                  -> print_endline("Canot pass type " ^ t ^ " to " ^ id)
|       Exceptions.IncorrectNumberOfArguments(f, a1, a2) -> print_endline("Cannot pass " ^ string_of_int a1 ^ " args when expecting " ^ string_of_int a2 ^ " in " ^f)
|       Exceptions.ClassIsNotExtendedBy(c1, c2)                            -> print_endline("Class " ^ c1 ^ " not extended by " ^ c2)

|       Exceptions.InvalidTypePassedToPrintf                               -> print_endline("Invalid type passed to print")
|       Exceptions.InvalidBinaryOperator                                     -> print_endline("Invalid binary operator")
|       Exceptions.UnknownVariable(id)                                               -> print_endline("Unknown variable " ^ id)
|       Exceptions.AssignLHSMustBeAssignable                                 -> print_endline("Assignment lhs must be assignable")
|       Exceptions.CannotCastTypeException(t1, t2)                          -> print_endline("Cannot cast " ^ t1 ^ " to " ^ t2)
|       Exceptions.InvalidBinopEvaluationType                                 -> print_endline("Invalid binary expression evaluation type")
|       Exceptions.FloatOpNotSupported                                               -> print_endline("Float operation not supported")
```

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

exceptions.ml

```
(* Dice Exceptions *)
exception InvalidNumberCompilerArguments of int
exception InvalidCompilerArgument of string
exception NoFileArgument

(* Processor Exceptions *)
exception MissingEOF

(* Scanner Exceptions *)
exception IllegalCharacter of string * char * int
exception UnmatchedQuotation of int
exception IllegalToken of string

(* Analyzer Exceptions *)
exception IncorrectNumberOfArgumentsException
exception ConstructorNotFound of string
exception DuplicateClassName of string
exception DuplicateField
exception DuplicateFunction of string
exception DuplicateConstructor
exception DuplicateLocal of string
exception UndefinedClass of string
exception UnknownIdentifier of string
exception InvalidBinopExpression of string
exception InvalidIfStatementType
exception InvalidForStatementType
exception ReturnTypeMismatch of string * string
exception MainNotDefined
exception MultipleMainsDefined
exception InvalidWhileStatementType
exception LocalAssignTypeMismatch of string * string
exception InvalidUnaryOperation
exception AssignmentTypeMismatch of string * string
exception FunctionNotFound of string * string
exception UndefinedID of string
exception InvalidAccessLHS of string
exception LHSofRootAccessMustBeIDorFunc of string
exception ObjAccessMustHaveObjectType of string
exception UnknownIdentifierForClass of string * string
exception CannotUseReservedFuncName of string
exception InvalidArrayPrimitiveConsecutiveTypes of string * string
exception InvalidArrayPrimitiveType of string
exception MustPassIntegerTypeToArrayCreate
exception ArrayInitTypeInvalid of string
exception MustPassIntegerTypeToArrayAccess
exception ArrayAccessInvalidParamLength of string * string
exception ArrayAccessExpressionNotArray of string
```

```
exception CanOnlyAccessLengthOfArray
exception CanOnlyDeleteObjectsOrArrays
exception CannotAccessLengthOfCharArray
exception AllNonVoidFunctionsMustEndWithReturn of string
exception CyclicalDependencyBetween of string * string
exception CannotAccessPrivateFieldInNonProperScope of string * string * string
exception CannotCallBreakOutsideOfLoop
exception CannotCallContinueOutsideOfLoop
exception CannotAccessPrivateFunctionInNonProperScope of string * string * string
exception CannotPassNonInheritedClassesInPlaceOfOthers of string * string
exception IncorrectTypePassedToFunction of string * string
exception IncorrectNumberOfArguments of string * int * int
exception ClassIsNotExtendedBy of string * string

(* Codegen Exceptions *)
exception InvalidTypePassedToPrintf
exception InvalidBinaryOperator
exception UnknownVariable of string
exception AssignLHSMustBeAssignable
exception CannotCastTypeException of string * string
exception InvalidBinopEvaluationType
exception FloatOpNotSupported
exception IntOpNotSupported
exception LLVMFunctionNotFound of string
exception InvalidStructType of string
exception UnableToCallFunctionWithoutParent of string
exception CannotAssignParam of string
exception InvalidUnopEvaluationType
exception UnopNotSupported
exception ArrayLargerThan1Unsupported
exception CanOnlyCompareObjectsWithNull of string * string
exception ObjOpNotSupported of string
exception CanOnlyCompareArraysWithNull of string * string
```

filepath.ml

```
open Filename
open Unix

exception Safe_exception of (string * string list ref)

let raise_safe fmt =
let do_raise msg = raise @@ Safe_exception (msg, ref []) in
Printf.ksprintf do_raise fmt

let reraise_with_context ex fmt =
let do_raise context =
let () = match ex with
| Safe_exception (_, old_contexts) -> old_contexts := context :: !old_contexts
| _ -> Printf.eprintf "warning: Attempt to add note '%s' to non-Safe_exception!" context
in
raise ex
in Printf.ksprintf do_raise fmt

module StringMap = struct
include Map.Make(String)
let find_nf = find
let find_safe key map = try find key map with Not_found -> raise_safe "BUG: Key '%s' not found in StringMap!" key
let find key map = try Some (find key map) with Not_found -> None
let map_bindings fn map = fold (fun key value acc -> fn key value :: acc) map []
end

type path_component =
| Filename of string  (* foo/ *)
| ParentDir           (* ../ *)
| CurrentDir          (* ./ *)
| EmptyComponent      (* / *)

type filepath = string


let on_windows = Filename.dir_sep <> "/"

let path_is_absolute path = not (Filename.is_relative path)

let string_tail s i =
let len = String.length s in
if i > len then failwith ("String '" ^ s ^ "' too short to split at " ^ (string_of_int i))
else String.sub s i (len - i)

let split_path_str path =
```

```
let l = String.length path in
let is_sep c = (c = '/' || (on_windows && c = '\\')) in

(* Skip any leading slashes and return the rest *)
let rec find_rest i =
if i < l then (
if is_sep path.[i] then find_rest (i + 1)
else string_tail path i
) else (
""
) in

let rec find_slash i =
if i < l then (
if is_sep path.[i] then (String.sub path 0 i, find_rest (i + 1))
else find_slash (i + 1)
) else (
(path, "")
)
in
find_slash 0

let split_first path =
if path = "" then
(CurrentDir, "")
else (
let (first, rest) = split_path_str path in
let parsed =
if first = Filename.parent_dir_name then ParentDir
else if first = Filename.current_dir_name then CurrentDir
else if first = "" then EmptyComponent
else Filename first in
(parsed, rest)
)

let normpath path : filepath =
let rec explode path =
match split_first path with
| CurrentDir, "" -> []
| CurrentDir, rest -> explode rest
| first, "" -> [first]
| first, rest -> first :: explode rest in

let rec remove_parents = function
| checked, [] -> checked
| (Filename _name :: checked), (ParentDir :: rest) -> remove_parents (checked, rest)
| checked, (first :: rest) -> remove_parents ((first :: checked), rest) in

let to_string = function
```

```
| Filename name -> name
| ParentDir -> Filename.parent_dir_name
| EmptyComponent -> ""
| CurrentDir -> assert false in
String.concat Filename.dir_sep @@ List.rev_map to_string @@ remove_parents ([], explode
↪ path)


let abspath path =
let (+/) = Filename.concat in
normpath (
if path_is_absolute path then path
else (Sys.getcwd ()) +/ path
)

let realpath path =
let (+/) = Filename.concat in   (* Faster version, since we know the path is relative *)

(* Based on Python's version *)
let rec join_realpath path rest seen =
(* Printf.printf "join_realpath <%s> + <%s>\n" path rest; *)
(* [path] is already a realpath (no symlinks). [rest] is the bit to join to it. *)
match split_first rest with
| Filename name, rest -> (
(* path + name/rest *)
let newpath = path +/ name in
let link = try Some (Unix.readlink newpath) with Unix.Unix_error _ -> None in
match link with
| Some target ->
(* path + symlink/rest *)
begin match StringMap.find newpath seen with
| Some (Some cached_path) -> join_realpath cached_path rest seen
| Some None -> (normpath (newpath +/ rest), false)    (* Loop; give up *)
| None ->
(* path + symlink/rest -> realpath(path + target) + rest *)
match join_realpath path target (StringMap.add newpath None seen) with
| path, false ->
(normpath (path +/ rest), false)   (* Loop; give up *)
| path, true -> join_realpath path rest (StringMap.add newpath (Some path) seen)
end
| None ->
(* path + name/rest -> path/name + rest (name is not a symlink) *)
join_realpath newpath rest seen
)
| CurrentDir, "" ->
(path, true)
| CurrentDir, rest ->
(* path + ./rest *)
join_realpath path rest seen
```

```
| ParentDir, rest ->
(* path + ../rest *)
if String.length path > 0 then (
let name = Filename.basename path in
let path = Filename.dirname path in
if name = Filename.parent_dir_name then
join_realpath (path +/ name +/ name) rest seen    (* path/.. +  ../rest -> path/../.. + rest *)
else
join_realpath path rest seen                      (* path/name + ../rest -> path + rest *)
) else (
join_realpath Filename.parent_dir_name rest seen    (* "" + ../rest -> .. + rest *)
)
| EmptyComponent, rest ->
(* [rest] is absolute; discard [path] and start again *)
join_realpath Filename.dir_sep rest seen
in

try
if on_windows then
abspath path
else (
fst @@ join_realpath (Sys.getcwd ()) path StringMap.empty
)
with Safe_exception _ as ex -> reraise_with_context ex "... in realpath(%s)" path
```

Makefile

```
TARGET=src/dice
LIBS=-I,/usr/lib/ocaml/
FLAGS= -j 0 -r -use-ocamlfind -pkgs yojson,llvm,llvm.analysis,llvm.bitwriter,llvm.bitreader,llvm.linker,llvm.target,batteries
OCAMLBUILD=ocamlbuild
OPAM=opam config env
CLIBEXT=_includes


all: native
	@clang-3.7 -c -emit-llvm src/bindings.c
	@mkdir -p $(CLIBEXT)
	@mv bindings.bc $(CLIBEXT)/bindings.bc
	@cp src/stdlib.dice $(CLIBEXT)/stdlib.dice
	@mv dice.native dice
	@echo Compilation Complete

clean:
	@cd src
	$(OCAMLBUILD) -clean
	@cd ..
	@rm -rf $(CLIBEXT)
	@echo cleaning complete

native:
	@cd src
	@eval `opam config env`
	$(OCAMLBUILD) $(FLAGS) $(TARGET).native
	@cd ..

byte:
	$(OCAMLBUILD) $(FLAGS) $(TARGET).byte

depend:
	echo "Not needed."
```

parser.mly

```
%{  open Ast  %}

%token CLASS EXTENDS CONSTRUCTOR INCLUDE DOT THIS PRIVATE PUBLIC
%token INT FLOAT BOOL CHAR VOID NULL TRUE FALSE
%token SEMI LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET COMMA
%token AND NOT OR PLUS MINUS TIMES DIVIDE ASSIGN MODULO
%token EQ NEQ LT LEQ GT GEQ BAR
%token RETURN IF ELSE FOR WHILE BREAK CONTINUE NEW DELETE
%token <int> INT_LITERAL
%token <float> FLOAT_LITERAL
%token <string> STRING_LITERAL
%token <string> ID
%token <char> CHAR_LITERAL
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left AND OR
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MODULO
%right NOT
%right DELETE
%right RBRACKET
%left LBRACKET
%right DOT

%start program
%type <Ast.program> program

%%

program:
includes cdecls EOF { Program($1, $2) }

/******************
INCLUDE
******************/

includes:
/* nothing */ { [] }
|          include_list  { List.rev $1 }

include_list:
include_decl              { [$1] }
```

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

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


/******************
CLASSES
******************/
cdecls:
cdecl_list    { List.rev $1 }

cdecl_list:
cdecl             { [$1] }
| cdecl_list cdecl  { $2::$1 }

cdecl:
CLASS ID LBRACE cbody RBRACE { {
                cname = $2;
                extends = NoParent;
                cbody = $4
        } }
|          CLASS ID EXTENDS ID LBRACE cbody RBRACE { {
                cname = $2;
                extends = Parent($4);
                cbody = $6
} }

cbody:
/* nothing */ { {
                fields = [];
                constructors = [];
                methods = [];
} }
|          cbody field { {
                fields = $2 :: $1.fields;
                constructors = $1.constructors;
                methods = $1.methods;
} }
|          cbody constructor { {
                fields = $1.fields;
                constructors = $2 :: $1.constructors;
                methods = $1.methods;
} }
|          cbody fdecl { {
        fields = $1.fields;
        constructors = $1.constructors;
        methods = $2 :: $1.methods;
} }
```

```


/******************
CONSTRUCTORS
*****************/

constructor:
CONSTRUCTOR LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE {
        {
                scope = Public;
                fname = Constructor;
                returnType = Datatype(ConstructorType);
                formals = $3;
                body = List.rev $6;
                overrides = false;
                root_cname = None;
        }
}

/******************
FIELDS
*****************/

scope:
PRIVATE { Private }
|         PUBLIC  { Public }

/* public UserObj name; */
field:
scope datatype ID SEMI { Field($1, $2, $3) }

/******************
METHODS
*****************/

fname:
ID { $1 }

fdecl:
scope datatype fname LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
{
        {
                scope = $1;
                fname = FName($3);
                returnType = $2;
                formals = $5;
                body = List.rev $8;
                overrides = false;
                root_cname = None;
```

```
        }
}

/*****************
FORMALS/PARAMETERS & VARIABLES & ACTUALS
*****************/

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

formal_list:
formal                    { [$1] }
|         formal_list COMMA formal { $3 :: $1 }

formal:
datatype ID { Formal($1, $2) }

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

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


/***************
DATATYPES
***************/
primitive:
INT                  { Int_t }
|         FLOAT                 { Float_t }
|         CHAR                 { Char_t }
|         BOOL                  { Bool_t }
|         VOID             { Void_t }

name:
CLASS ID { Objecttype($2) }

type_tag:
primitive { $1 }
|        name           { $1 }

array_type:
type_tag LBRACKET brackets RBRACKET { Arraytype($1, $3) }

datatype:
type_tag   { Datatype($1) }
```

```
|         array_type { $1 }

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

/*****************
EXPRESSIONS
*****************/

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

stmt:
expr SEMI { Expr($1) }
|         RETURN expr SEMI { Return($2) }
|        RETURN SEMI                 { Return(Noexpr) }
|         LBRACE stmt_list RBRACE { Block(List.rev $2) }
|         IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([Expr(Noexpr)])) }
|         IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
|         FOR LPAREN expr_opt SEMI expr_opt SEMI expr_opt RPAREN stmt
{ For($3, $5, $7, $9) }
|         WHILE LPAREN expr RPAREN stmt         { While($3, $5) }
|        BREAK SEMI                                                { Break }
|        CONTINUE SEMI                                        { Continue }
|   datatype ID SEMI                                 { Local($1, $2, Noexpr) }
|         datatype ID ASSIGN expr SEMI         { Local($1, $2, $4) }

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

expr:
literals                                                    { $1 }
|         expr PLUS   expr                                   { Binop($1, Add,   $3)
↪  }
|         expr MINUS  expr                                   { Binop($1, Sub,   $3)
↪  }
|         expr TIMES  expr                                   { Binop($1, Mult,  $3)
↪  }
|         expr DIVIDE expr                                   { Binop($1, Div,   $3)
↪  }
|         expr EQ     expr                                   { Binop($1, Equal, $3)
↪  }
|         expr NEQ    expr                                   { Binop($1, Neq,   $3)
↪  }
|         expr LT     expr                                   { Binop($1, Less,  $3)
↪  }
```

```
|         expr LEQ    expr                                        { Binop($1, Leq,    $3) }
|         expr GT     expr                                        { Binop($1, Greater, $3) }
|         expr GEQ    expr                                        { Binop($1, Geq,    $3) }
|         expr AND    expr                                        { Binop($1, And,    $3) }
|         expr MODULO expr                                        { Binop($1, Mod, $3)}
|         NOT  expr                                                        { Unop (Not, $2) }
|         expr OR     expr                                        { Binop($1, Or,     $3) }
|         expr DOT    expr                                        { ObjAccess($1, $3) }
|         expr ASSIGN expr                                        { Assign($1, $3) }
|         DELETE expr                                               { Delete($2) }
|   MINUS expr                                                      { Unop (Sub, $2) }
|         ID LPAREN actuals_opt RPAREN                { Call($1, $3) }
|         NEW ID LPAREN actuals_opt RPAREN        { ObjectCreate($2, $4) }
|        NEW type_tag bracket_args RBRACKET        { ArrayCreate(Datatype($2), List.rev $3) }
|         expr bracket_args RBRACKET                          { ArrayAccess($1, List.rev $2) }
|         LPAREN expr RPAREN                                      { $2 }

bracket_args:
LBRACKET expr                                                     { [$2] }
|         bracket_args RBRACKET LBRACKET expr { $4 :: $1 }

literals:
INT_LITERAL                         { Int_Lit($1) }
| FLOAT_LITERAL                       { Float_Lit($1) }
| TRUE                                              { Boolean_Lit(true) }
| FALSE                                              { Boolean_Lit(false) }
| STRING_LITERAL                      { String_Lit($1) }
| CHAR_LITERAL                           { Char_Lit($1) }
| THIS                                               { This }
| ID                                               { Id($1) }
| NULL                                       { Null }
| BAR array_prim BAR          { ArrayPrimitive($2) }

/* ARRAY LITERALS */

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

processor.ml

```
open Parser

type token_attr = {
        lineno: int;
        cnum: int;
}

let line_number = ref 1
let last_token = ref EOF
let char_num = ref 1
let filename = ref ""

let build_token_list lexbuf =
Scanner.filename := !filename;
let rec helper prev_cnum prev_lineno lexbuf token_list =
let token = Scanner.token lexbuf in
let lineno = !Scanner.lineno in
let cnum = (Lexing.lexeme_start_p lexbuf).Lexing.pos_cnum in
let prev_cnum = if lineno > prev_lineno then cnum else prev_cnum in
let cnum = cnum - prev_cnum in
match token with
EOF as eof -> (eof, { lineno = lineno; cnum = cnum } )::token_list
|   t          -> (t, { lineno = lineno; cnum = cnum } )::(helper prev_cnum lineno lexbuf token_list)
in helper 0 0 lexbuf []

let parser filen token_list =
let token_list = ref(token_list) in
let tokenizer _ =
match !token_list with
| (head, curr) :: tail ->
filename := filen;
line_number := curr.lineno;
char_num    := curr.cnum;
last_token := head;
token_list := tail;
head
| [] -> raise (Exceptions.MissingEOF)
in
let program = Parser.program tokenizer (Lexing.from_string "") in
program
```

sast.ml

```
open Ast

type sexpr =
SInt_Lit of int
|         SBoolean_Lit of bool
|         SFloat_Lit of float
|         SString_Lit of string
|         SChar_Lit of char
|         SId of string * datatype
|         SBinop of sexpr * op * sexpr * datatype
|         SAssign of sexpr * sexpr * datatype
|         SNoexpr
|         SArrayCreate of datatype * sexpr list * datatype
|         SArrayAccess of sexpr * sexpr list * datatype
|         SObjAccess of sexpr * sexpr * datatype
|         SCall of string * sexpr list * datatype * int
|   SObjectCreate of string * sexpr list * datatype
|         SArrayPrimitive of sexpr list * datatype
|          SUnop of op * sexpr * datatype
|         SNull
|         SDelete of sexpr

type sstmt =
SBlock of sstmt list
|         SExpr of sexpr * datatype
|         SReturn of sexpr  * datatype
|         SIf of sexpr * sstmt * sstmt
|         SFor of sexpr * sexpr * sexpr * sstmt
|         SWhile of sexpr * sstmt
|          SBreak
|   SContinue
|   SLocal of datatype * string * sexpr

type func_type = User | Reserved

type sfunc_decl = {
        sfname : fname;
        sreturnType : datatype;
        sformals : formal list;
        sbody : sstmt list;
        func_type : func_type;
        source : string;
        overrides : bool;
}

type sclass_decl = {
        scname : string;
```

```
        sfields : field list;
        sfuncs: sfunc_decl list;
}

(* Class Declarations | All method declarations | Main entry method *)
type sprogram =  {
        classes : sclass_decl list;
        functions : sfunc_decl list;
        main : sfunc_decl;
        reserved : sfunc_decl list;
}
```

scanner.mll

```
{
        open Parser
        let lineno = ref 1
        let depth = ref 0
        let filename = ref ""

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

let alpha = ['a'-'z' 'A'-'Z']
let escape = '\\' ['\\' ''' '"' 'n' 'r' 't']
let escape_char = ''' (escape) '''
let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let digit = ['0'-'9']
let id = alpha (alpha | digit | '_')*
let string = '"' ( (ascii | escape)* as s) '"'
let char = ''' ( ascii | digit ) '''
let float = (digit+) ['.'] digit+
let int = digit+
let whitespace = [' ' '\t' '\r']
let return = '\n'

rule token = parse
whitespace { token lexbuf }
| return          { incr lineno; token lexbuf}
| "(*"      { incr depth; comment lexbuf }

| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
        | '}'      { RBRACE }
| ';'      { SEMI }
| ','      { COMMA }

(* Operators *)
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '%'      { MODULO }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
```

```
| ">="     { GEQ }
| "and"    { AND }
| "or"     { OR }
| "not"    { NOT }
| '.'      { DOT }
| '['      { LBRACKET }
| ']'      { RBRACKET }
| '|'           { BAR }

(* Branch Control *)
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }

(* Data Types *)
| "int"    { INT }
| "float"  { FLOAT }
| "bool"   { BOOL }
| "char"   { CHAR }
| "void"   { VOID }
| "null"   { NULL }
| "true"   { TRUE }
| "false"  { FALSE }

(* Classes *)
| "class"       { CLASS }
| "constructor" { CONSTRUCTOR }
| "public"      { PUBLIC }
| "private"     { PRIVATE }
| "extends"     { EXTENDS }
| "include"     { INCLUDE }
| "this"        { THIS }
| "break"                 { BREAK }
| "continue"        { CONTINUE }
| "new"                 { NEW }
| "delete"                 { DELETE }

| int as lxm                   { INT_LITERAL(int_of_string lxm) }
| float as lxm                 { FLOAT_LITERAL(float_of_string lxm) }
| char as lxm                  { CHAR_LITERAL( String.get lxm 1 ) }
| escape_char as lxm{ CHAR_LITERAL( String.get (unescape lxm) 1) }
| string                       { STRING_LITERAL(unescape s) }
| id as lxm                    { ID(lxm) }
| eof                          { EOF }

| '"'                         { raise (Exceptions.UnmatchedQuotation(!lineno)) }
| _ as illegal  { raise (Exceptions.IllegalCharacter(!filename, illegal, !lineno)) }
```

```

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

stdlibe.dice

```
class Integer {

        private int my_int;

        constructor(int input) {
                this.my_int = input;
        }

        public int num() {
                return this.my_int;
        }


        public char toChar(int digit) {

                if (digit == 0) {
                        return '0';
                } else if (digit == 1) {
                        return '1';
                } else if (digit == 2) {
                        return '2';
                } else if (digit == 3) {
                return '3';
                } else if (digit == 4) {
                return '4';
                } else if (digit == 5) {
                return '5';
                } else if (digit == 6) {
                return '6';
                } else if (digit == 7) {
                return '7';
                } else if (digit == 8) {
                return '8';
                } else if (digit == 9) {
                return '9';
                }

        return 'z';
}





public class String toString() {

        (* integer cannot be greater than 10 digits in 32 bit *)
```

```
        int temp = this.my_int;
        int i = 0;
        char[] str = new char[9];

        int digit = temp % 10;
        str[i] = this.toChar(digit);
        i = i + 1;
        temp = temp / 10;
        while (temp > 0) {

                digit = temp % 10;
                str[i] = this.toChar(digit);
                temp = temp / 10;
                i = i + 1;
        }

        str[i] = 0;
        class String newString = new String(str);
        class String a = newString.reverse();
        return newString.reverse();
}
}



class String {

        private char[] my_string;
        private int length;

        constructor(char[] input) {

                this.my_string = this.copy_internal(input);

                this.length = this.length();
        }

        (* PRIVATE CLASSES ----------------------------------------   *)

        private int length_internal(char[] input) {
                int length = 0;

                while(input[length] != 0) {
                        length = length + 1;
                }

                return length;
        }

```

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

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

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

		newString[i] = 0;
		return newString;
	}

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

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

	public char getChar(int index) {

		return this.my_string[index];
	}

	public int length() {

		int length = 0;

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

		return length;
	}

	public int toInteger() {

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

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

```
                        ndigit = ndigit + xdigit;
                }

                return ndigit;
        }

        public int toDigit(char digit) {

                if (digit == '0') {
                        return 0;
                } else if (digit == '1') {
                return 1;
        } else if (digit == '2') {
        return 2;
} else if (digit == '3') {
return 3;
} else if (digit == '4') {
return 4;
} else if (digit == '5') {
return 5;
} else if (digit == '6') {
return 6;
} else if (digit == '7') {
return 7;
} else if (digit == '8') {
return 8;
} else if (digit == '9') {
return 9;
}

return -1;
}


public class String copy(class String input) {

        char[] newArray = this.copy_internal(input.string());
        class String newString = new String(newArray);
        return newString;
}

public int indexOf(char x) {

        int i = 0;
        for (; this.getChar(i) != x and this.getChar(i) != 0; i = i + 1) {
        }

        (* If the char was not found, return -1 *)
        if (i == this.length()) {
```

```
                return -1;
        }

        return i;
}

public class String reverse() {

        class String newString;

        char[] temp = new char[this.length + 1];
        int i = this.length;
        for (; i > 0; i = i - 1) {

                temp[this.length - i] = this.getChar(i-1);
        }
        temp[this.length] = 0;
        newString = new String(temp);
        return newString;
}

public class String concat(class String temp) {

        char[] temparray = new char[this.length() + temp.length() + 1];

        (* Copy over the current string into a new char array *)
        int i = 0;
        for (; this.getChar(i) != 0; i = i + 1) {
                temparray[i] = this.getChar(i);
        }

        (* Append the new string *)
        int j = 0;
        for (; temp.getChar(j) != 0; j = j + 1) {
                temparray[i+j] = temp.getChar(j);
        }

        temparray[this.length() + temp.length()] = 0;
        class String newString = new String(temparray);
        return newString;
}

public bool compare(class String check) {

        if (check.length != this.length) {
                return false;
        }

        int i = 0;
```

```

        for (; i < check.length(); i = i + 1) {

                if (check.getChar(i) != this.getChar(i)) {
                        return false;
                }
        }

        return true;
}

public bool contains(class String check) {


        if (this.length < check.length) {
                return false;
        } else if (this.compare(check)) {
        return true;
} else {

int diff = this.length - check.length + 1;
int i;
int j;
for ( i = 0; i < diff; i = i + 1)

for ( j = 0; j < check.length; j = j + 1) {

        if (this.getChar(i+j) != check.getChar(j)) {
                break;
        }

        if (j == check.length - 1) {
                return true;
        }
}
}
return false;
}

public void free() {

        delete(this.my_string);
}

}



class File {
```

```
        private class String filePath;
        private bool isWriteEnabled;
        private int fd;

        constructor(char[] path, bool isWriteEnabled) {

                this.filePath = new String(path);
                this.isWriteEnabled = isWriteEnabled;
                class String a = this.filePath;
                this.fd = this.openfile(a, this.isWriteEnabled);
                if (this.fd < 0) {
                        print("open failed");
                        exit(1);
                }
        }

        (* PRIVATE CLASSES ----------------------------------------   *)

        private int openfile(class String path, bool isWriteEnabled) {

                if (isWriteEnabled) {
                        (* 2 is the value for O_RDWR *)
                        return open(path.string(), 2);
                }

                (* 0 is the value for O_RDONLY *)
                return open(path.string(), 0);
        }

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

        public void closefile() {

                if (close(this.fd) < 0) {
                        print("close failed");
                }
        }

        public char[] readfile(int bytes) {

                char[] buf = new char[bytes];

                int ret = read(this.fd, buf, bytes);

                if (ret < 0) {
                        print("read failed");
                }

```

```
                return buf;
        }

        public int writefile(char[] buf, int offset) {

                class String temp = new String(buf);
                int err;
                (* seek to desired offset from beginning of file *)
                if (offset > 0) {
                        err = lseek(this.fd, offset, 0);
                } else if (offset == -1) {
                err = lseek(this.fd, 0, 0);
        } else {
        (* Seek to the end of the file by default *)
        err = lseek(this.fd, 0, 2);
}

if (err < 0) {
        print("seek failed");
}

err = write(this.fd, temp.string(), temp.length());
if (err < 0) {
        print("write failed");
}
return err;
}

}
```

utils.ml

```
(* Pretty Printer *)
open Ast
open Sast
open Parser
open Processor
open Yojson

let save file string =
let channel = open_out file in
output_string channel string;
close_out channel

let replace input output =
Str.global_replace (Str.regexp_string input) output

(* Print data types *)

let string_of_scope = function
Public         -> "public"
|          Private -> "private"

let string_of_primitive = function
Int_t                                                    -> "int"
|          Float_t                                           -> "float"
|          Void_t                                                  -> "void"
|          Bool_t                                                   -> "bool"
|          Char_t                                                   -> "char"
|          Objecttype(s)                                    -> "class " ^ s
|          ConstructorType                                    -> "constructor"
|           Null_t                                                   -> "null"

let string_of_object = function
Datatype(Objecttype(s))        -> s
|          _ -> ""

let rec print_brackets = function
1 -> "[]"
|          a -> "[]" ^ print_brackets (a - 1)

let string_of_datatype = function
Arraytype(p, i)        -> (string_of_primitive p) ^ (print_brackets i)
|          Datatype(p)                -> (string_of_primitive p)
|           Any                          -> "Any"

(* Print expressions *)

let string_of_op = function
```

```
Add                          -> "+"
|          Sub                          -> "-"
|          Mult                  -> "*"
|          Div                          -> "/"
|          Equal                  -> "=="
|          Neq                          -> "!="
|          Less                  -> "<"
|          Leq                          -> "<="
|          Greater                  -> ">"
|          Geq                          -> ">="
|          And                          -> "and"
|          Not                          -> "not"
|          Or                          -> "or"
|          Mod                  -> "%"

let rec string_of_bracket_expr = function
[]                                  -> ""
|          head :: tail          -> "[" ^ (string_of_expr head) ^ "]" ^
↪    (string_of_bracket_expr tail)
and string_of_array_primitive = function
[]                                  -> ""
|    [last]                          -> (string_of_expr last)
|          head :: tail          -> (string_of_expr head) ^ ", " ^
↪    (string_of_array_primitive tail)
and string_of_expr = function
Int_Lit(i)                                  -> string_of_int i
|         Boolean_Lit(b)                          -> if b then "true" else "false"
|         Float_Lit(f)                          -> string_of_float f
|         String_Lit(s)                          -> "\"" ^ (String.escaped s) ^ "\""
|         Char_Lit(c)                                  -> Char.escaped c
|         This                                          -> "this"
|         Id(s)                                          -> s
|         Binop(e1, o, e2)                  -> (string_of_expr e1) ^ " " ^ (string_of_op o)
↪    ^ " " ^ (string_of_expr e2)
|         Assign(e1, e2)                          -> (string_of_expr e1) ^ " = " ^
↪    (string_of_expr e2)
|         Noexpr                                          -> ""
|         ObjAccess(e1, e2)                  -> (string_of_expr e1) ^ "." ^ (string_of_expr
↪    e2)
|         Call(f, el)                                  -> f ^ "(" ^ String.concat ", "
↪    (List.map string_of_expr el) ^ ")"
|         ArrayPrimitive(el)                  -> "|" ^ (string_of_array_primitive el) ^ "|"
|           Unop(op, e)                                  -> (string_of_op op) ^ "(" ^
↪    string_of_expr e ^ ")"
|         Null                                          -> "null"
|    ArrayCreate(d, el)          -> "new " ^ string_of_datatype d ^ string_of_bracket_expr
↪    el
|    ArrayAccess(e, el)          -> (string_of_expr e) ^ (string_of_bracket_expr el)
```

```
|   ObjectCreate(s, el)         -> "new " ^ s ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
|         Delete(e)                            -> "delete (" ^ (string_of_expr e) ^ ")"
;;

let rec string_of_bracket_sexpr = function
[]                                -> ""
|         head :: tail          -> "[" ^ (string_of_sexpr head) ^ "]" ^ (string_of_bracket_sexpr tail)
and string_of_sarray_primitive = function
[]                                -> ""
|   [last]                        -> (string_of_sexpr last)
|         head :: tail          -> (string_of_sexpr head) ^ ", " ^ (string_of_sarray_primitive tail)
and string_of_sexpr = function
SInt_Lit(i)                                        -> string_of_int i
|        SBoolean_Lit(b)                                  -> if b then "true" else "false"
|        SFloat_Lit(f)                                    -> string_of_float f
|        SString_Lit(s)                                    -> "\"" ^ (String.escaped s) ^ "\""
|        SChar_Lit(c)                                    -> Char.escaped c
|        SId(s, _)                                            -> s
|        SBinop(e1, o, e2, _)                   -> (string_of_sexpr e1) ^ " " ^ (string_of_op o) ^ " " ^ (string_of_sexpr e2)
|        SAssign(e1, e2, _)                        -> (string_of_sexpr e1) ^ " = " ^ (string_of_sexpr e2)
|        SNoexpr                                                      -> ""
|        SObjAccess(e1, e2, _)                  -> (string_of_sexpr e1) ^ "." ^ (string_of_sexpr e2)
|        SCall(f, el, _, _)                         -> f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
|        SArrayPrimitive(el, _)                  -> "|" ^ (string_of_sarray_primitive el) ^ "|"
|          SUnop(op, e, _)                                 -> (string_of_op op) ^ "(" ^ string_of_sexpr e ^ ")"
|        SNull                                                      -> "null"
|   SArrayCreate(d, el, _)          -> "new " ^ string_of_datatype d ^ string_of_bracket_sexpr el
|   SArrayAccess(e, el, _)          -> (string_of_sexpr e) ^ (string_of_bracket_sexpr el)
|   SObjectCreate(s, el, _)         -> "new " ^ s ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
|         SDelete(e)                                       -> "delete (" ^ (string_of_sexpr e) ^ ")"
;;

let string_of_local_expr = function
Noexpr -> ""
|          e              -> " = " ^ string_of_expr e
```

```

(* Print statements *)

let rec string_of_stmt indent =
let indent_string = String.make indent '\t' in
let get_stmt_string = function

Block(stmts)                          ->
indent_string ^ "{\n" ^
        String.concat "" (List.map (string_of_stmt (indent+1)) stmts) ^
        indent_string ^ "}\n"

|          Expr(expr)                                  ->
indent_string ^ string_of_expr expr ^ ";\n";

|          Return(expr)                          ->
indent_string ^ "return " ^ string_of_expr expr ^ ";\n";

|          If(e, s, Block([Expr(Noexpr)]))          ->
indent_string ^ "if (" ^ string_of_expr e ^ ")\n" ^
(string_of_stmt (indent+1) s)

|          If(e, s1, s2)                         ->
indent_string ^ "if (" ^ string_of_expr e ^ ")\n" ^
string_of_stmt (indent+1) s1 ^
indent_string ^ "else\n" ^
string_of_stmt (indent+1) s2

|          For(e1, e2, e3, s)                  ->
indent_string ^ "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
↪  string_of_expr e3  ^ ")\n" ^
string_of_stmt (indent) s

|          While(e, s)                          ->
indent_string ^ "while (" ^ string_of_expr e ^ ")\n" ^
string_of_stmt (indent) s

|           Break                                          -> indent_string ^ "break;\n"
|           Continue                                  -> indent_string ^ "continue;\n"
|   Local(d, s, e)                        -> indent_string ^ string_of_datatype d ^ " "
↪  ^ s ^ string_of_local_expr e ^ ";\n"
in get_stmt_string

let string_of_local_sexpr = function
SNoexpr          -> ""
|           e                            -> " = " ^ string_of_sexpr e

let rec string_of_sstmt indent =
let indent_string = String.make indent '\t' in
```

```
let get_stmt_string = function

SBlock(stmts)                        ->
indent_string ^ "{\n" ^
        String.concat "" (List.map (string_of_sstmt (indent+1)) stmts) ^
        indent_string ^ "}\n"

|         SExpr(expr, _)                                 ->
indent_string ^ string_of_sexpr expr ^ ";\n";

|         SReturn(expr, _)                         ->
indent_string ^ "return " ^ string_of_sexpr expr ^ ";\n";

|         SIf(e, s, SBlock([SExpr(SNoexpr, _)]))         ->
indent_string ^ "if (" ^ string_of_sexpr e ^ ")\n" ^
(string_of_sstmt (indent+1) s)

|         SIf(e, s1, s2)                        ->
indent_string ^ "if (" ^ string_of_sexpr e ^ ")\n" ^
string_of_sstmt (indent+1) s1 ^
indent_string ^ "else\n" ^
string_of_sstmt (indent+1) s2

|         SFor(e1, e2, e3, s)                ->
indent_string ^ "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
↪  string_of_sexpr e3  ^ ")\n" ^
string_of_sstmt (indent) s

|         SWhile(e, s)                        ->
indent_string ^ "while (" ^ string_of_sexpr e ^ ")\n" ^
string_of_sstmt (indent) s

|          SBreak                                        -> indent_string ^ "break;\n"
|          SContinue                                -> indent_string ^ "continue;\n"
|   SLocal(d, s, e)                        -> indent_string ^ string_of_datatype d ^ " "
↪  ^ s ^ string_of_local_sexpr e ^ ";\n"
in get_stmt_string

(* Print Function *)

let string_of_fname = function
Constructor -> "constructor"
|        FName(s)        -> s

let string_of_formal = function
Formal(d, s) -> (string_of_datatype d) ^ " " ^ s
|          _                         -> ""

let string_of_formal_name = function
```

```
Formal(_, s) -> s
|           _ -> ""

let string_of_func_decl fdecl =
"" ^ (string_of_scope fdecl.scope) ^ " " ^ (string_of_datatype fdecl.returnType) ^ " " ^
↪  (string_of_fname fdecl.fname) ^ " " ^
(* Formals *)
"(" ^ String.concat "," (List.map string_of_formal fdecl.formals) ^ ") {\n" ^
        (* body *)
        String.concat "" (List.map (string_of_stmt 2) fdecl.body) ^
        "\t}\n\n"

(* Class Printing *)

let string_of_extends = function
NoParent          -> ""
|          Parent(s)          -> "extends " ^ s ^ " "
let string_of_field = function
Field(s, d, id) -> (string_of_scope s) ^ " " ^ (string_of_datatype d) ^ " " ^ id ^ ";\n"

let string_of_cbody cbody =
String.concat "" (List.map (fun s -> "\t" ^ s) (List.map string_of_field cbody.fields)) ^
String.concat "" (List.map (fun s -> "\t" ^ s) (List.map string_of_func_decl
↪  cbody.constructors)) ^
String.concat "" (List.map (fun s -> "\t" ^ s) (List.map string_of_func_decl
↪  cbody.methods))

let string_of_class_decl cdecl =
"class " ^ cdecl.cname ^ " " ^ (string_of_extends cdecl.extends) ^ "{\n" ^
        (string_of_cbody cdecl.cbody) ^
        "}\n"

(* Include Printing *)

let rec string_of_include = function
Include(s) -> "include(" ^ s ^ ");\n"

(* Print whole program *)

let string_of_program = function
Program(includes, cdecls) ->
String.concat "" (List.map string_of_include includes) ^ "\n" ^
String.concat "\n" (List.map string_of_class_decl cdecls)

(* Print AST tree representation *)

let includes_tree includes =
`List (List.map (function Include s -> `String s) includes)
```

```
let map_fields_to_json fields =
`List (List.map (function Field(scope, datatype, s) ->
`Assoc [
("name", `String s);
("scope", `String (string_of_scope scope));
("datatype", `String (string_of_datatype datatype));
]) fields)

let map_formals_to_json formals =
`List (List.map (function Formal(d, s) -> `Assoc [
("name", `String s);
("datatype", `String (string_of_datatype d));
]
| Many d -> `Assoc [("Many", `String (string_of_datatype d));]
) formals)

let rec map_expr_to_json = function
Int_Lit(i)                                -> `Assoc [("int_lit", `Int i)]
|        Boolean_Lit(b)                            -> `Assoc [("bool_lit", `Bool b)]
|        Float_Lit(f)                            -> `Assoc [("float_lit", `Float f)]
|        String_Lit(s)                            -> `Assoc [("string_lit", `String s)]
|        Char_Lit(c)                                    -> `Assoc [("char_lit", `String (Char.escaped c))]
|        This                                            -> `String "this"
|        Id(s)                                            -> `Assoc [("id", `String s)]
|        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)])]
|        Assign(e1, e2)                            -> `Assoc [("assign", `Assoc [("lhs", map_expr_to_json e1); ("op", `String "="); ("rhs", map_expr_to_json e2)])]
|        Noexpr                                            -> `String "noexpr"
|        ObjAccess(e1, e2)                    -> `Assoc [("objaccess", `Assoc [("lhs", map_expr_to_json e1); ("op", `String "."); ("rhs", map_expr_to_json e2)])]
|        Call(f, el)                                    -> `Assoc [("call", `Assoc ([("name", `String f); ("params", `List (List.map map_expr_to_json el)); ]) )]
|        ArrayPrimitive(el)                    -> `Assoc [("arrayprimitive", `List(List.map map_expr_to_json el))]
|          Unop(op, e)                                    -> `Assoc [("Unop", `Assoc [("op", `String (string_of_op op)); ("operand", map_expr_to_json e)])]
|        Null                                            -> `String "null"
|    ArrayCreate(d, el)            -> `Assoc [("arraycreate", `Assoc [("datatype", `String (string_of_datatype d)); ("args", `List (List.map map_expr_to_json el))])]
|    ArrayAccess(e, el)            -> `Assoc [("arrayaccess", `Assoc [("array", map_expr_to_json e); ("args", `List (List.map map_expr_to_json el))])]
|    ObjectCreate(s, el)            -> `Assoc [("objectcreate", `Assoc [("type", `String s); ("args", `List (List.map map_expr_to_json el))])]
|         Delete(e)                                    -> `Assoc [("delete", `Assoc [("expr", map_expr_to_json e)])]
```

```
let rec map_stmt_to_json = function
Block(stmts)                           -> `Assoc [("block", `List (List.map (map_stmt_to_json) stmts))]
|         Expr(expr)                                -> `Assoc [("expr", map_expr_to_json expr)]
|         Return(expr)                        -> `Assoc [("return", map_expr_to_json expr)]
|         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)]
|         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)])]
|         While(e, s)                        -> `Assoc [("while", `Assoc [("cond", map_expr_to_json e); ("body", map_stmt_to_json s)])]
|          Break                                        -> `String "break"
|          Continue                                -> `String "continue"
|    Local(d, s, e)                        -> `Assoc [("local", `Assoc [("datatype", `String (string_of_datatype d)); ("name", `String s); ("val", map_expr_to_json e)])]

let map_methods_to_json methods =
`List (List.map (fun (fdecl:Ast.func_decl) ->
`Assoc [
("name", `String (string_of_fname fdecl.fname));
("scope", `String (string_of_scope fdecl.scope));
("returnType", `String (string_of_datatype fdecl.returnType));
("formals", map_formals_to_json fdecl.formals);
("body", `List (List.map (map_stmt_to_json) fdecl.body));
]) methods)


let cdecls_tree cdecls =
let map_cdecl_to_json cdecl =
`Assoc [
("cname", `String cdecl.cname);
("extends", `String (string_of_extends cdecl.extends));
("fields", map_fields_to_json cdecl.cbody.fields);
("methods", map_methods_to_json cdecl.cbody.methods);
("constructors", map_methods_to_json cdecl.cbody.constructors)
]
in
`List (List.map (map_cdecl_to_json) cdecls)

let print_tree = function
Program(includes, cdecls) ->
`Assoc [("program",
`Assoc([
("includes", includes_tree includes);
("classes", cdecls_tree cdecls)
```

```
])
)]

(* Print SAST tree representation *)

let rec map_sexpr_to_json =
let datatype d = [("datatype", `String (string_of_datatype d))] in
function
SInt_Lit(i)                 -> `Assoc [("int_lit", `Assoc ([("val", `Int i)] @ (datatype (Datatype(Int_t)))))]
|   SBoolean_Lit(b)             -> `Assoc [("bool_lit", `Assoc ([("val", `Bool b)] @ (datatype (Datatype(Bool_t)))))]
|   SFloat_Lit(f)               -> `Assoc [("float_lit", `Assoc ([("val", `Float f)]  @ (datatype (Datatype(Float_t)))))]
|   SString_Lit(s)              -> `Assoc [("string_lit", `Assoc ([("val", `String s)] @ (datatype (Arraytype(Char_t, 1)))))]
|   SChar_Lit(c)                -> `Assoc [("char_lit", `Assoc ([("val", `String (Char.escaped c))] @ (datatype (Datatype(Char_t)))))]
|   SId(s, d)               -> `Assoc [("id", `Assoc ([("name", `String s)] @ (datatype d)))]
|   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)))]
|   SAssign(e1, e2, d)      -> `Assoc [("assign", `Assoc ([("lhs", map_sexpr_to_json e1); ("op", `String "="); ("rhs", map_sexpr_to_json e2)] @ (datatype d)))]
|   SNoexpr                     -> `Assoc [("noexpr", `Assoc (datatype (Datatype(Void_t))))]
|   SArrayCreate(t, el, d)  -> `Assoc [("arraycreate", `Assoc ([("datatype", `String (string_of_datatype d)); ("args", `List (List.map map_sexpr_to_json el))] @ (datatype d)))]
|   SArrayAccess(e, el, d)  -> `Assoc [("arrayaccess", `Assoc ([("array", map_sexpr_to_json e); ("args", `List (List.map map_sexpr_to_json el))] @ (datatype d)))]
|   SObjAccess(e1, e2, d)   -> `Assoc [("objaccess", `Assoc ([("lhs", map_sexpr_to_json e1); ("op", `String "."); ("rhs", map_sexpr_to_json e2)] @ (datatype d)))]
|   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)) )]
|   SObjectCreate(s, el, d) -> `Assoc [("objectcreate", `Assoc ([("type", `String s); ("args", `List (List.map map_sexpr_to_json el))] @ (datatype d)))]
|   SArrayPrimitive(el, d)  -> `Assoc [("arrayprimitive", `Assoc ([("expressions", `List(List.map map_sexpr_to_json el))] @ (datatype d)))]
|   SUnop(op, e, d)         -> `Assoc [("Unop", `Assoc ([("op", `String (string_of_op op)); ("operand", map_sexpr_to_json e)] @ (datatype d)))]
|   SNull                      -> `Assoc [("null", `Assoc (datatype (Datatype(Void_t))))]
|         SDelete(e)                                  -> `Assoc [("delete", `Assoc ([("expr", map_sexpr_to_json e)] @ (datatype (Datatype(Void_t)))))]

let rec map_sstmt_to_json =
```

```
let datatype d = [("datatype", `String (string_of_datatype d))] in
function
SBlock sl                               -> `Assoc [("sblock", `List (List.map (map_sstmt_to_json) sl))]
|   SExpr(e, d)                         -> `Assoc [("sexpr", `Assoc ([("expr", map_sexpr_to_json e)] @ (datatype d)))]
|   SReturn(e, d)                           -> `Assoc [("sreturn", `Assoc ([("return", map_sexpr_to_json e)] @ (datatype d)))]
|   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)]
|   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)])]
|   SWhile (e, s)                           -> `Assoc [("swhile", `Assoc [("cond", map_sexpr_to_json e); ("body", map_sstmt_to_json s)])]
|   SBreak                                  -> `String "sbreak"
|   SContinue                               -> `String "scontinue"
|   SLocal(d, s, e)                         -> `Assoc [("slocal", `Assoc [("datatype", `String (string_of_datatype d)); ("name", `String s); ("val", map_sexpr_to_json e)])]

let string_of_func_type = function
User -> "user" | Reserved -> "reserved"

let map_sfdecl_to_json sfdecl =
`Assoc[("sfdecl", `Assoc[
("sfname", `String (string_of_fname sfdecl.sfname));
("sreturnType", `String (string_of_datatype sfdecl.sreturnType));
("sformals", map_formals_to_json sfdecl.sformals);
("sbody", `List (List.map (map_sstmt_to_json) sfdecl.sbody));
("func_type", `String(string_of_func_type sfdecl.func_type));
])]

let map_sfdecls_to_json sfdecls =
`List(List.map map_sfdecl_to_json sfdecls)

let map_scdecls_to_json scdecls =
`List(List.map (fun scdecl ->
`Assoc [("scdecl",
`Assoc[
("scname", `String scdecl.scname);
("sfields", map_fields_to_json scdecl.sfields);
("sfuncs", map_sfdecls_to_json scdecl.sfuncs);
])
])
scdecls)

let map_sprogram_to_json sprogram =
`Assoc [("sprogram", `Assoc [
```

```
("classes", map_scdecls_to_json sprogram.classes);
("functions", map_sfdecls_to_json sprogram.functions);
("main", map_sfdecl_to_json sprogram.main);
("reserved", map_sfdecls_to_json sprogram.reserved);
])]

(* Print tokens *)

let string_of_token = function
LPAREN                              -> "LPAREN"
|          RPAREN                              -> "RPAREN"
|          LBRACE                              -> "LBRACE"
|          RBRACE                              -> "RBRACE"
|          SEMI                              -> "SEMI"
|          COMMA                              -> "COMMA"
|          PLUS                              -> "PLUS"
|          MINUS                              -> "MINUS"
|          TIMES                              -> "TIMES"
|          DIVIDE                              -> "DIVIDE"
|          ASSIGN                              -> "ASSIGN"
|          EQ                                  -> "EQ"
|          NEQ                                  -> "NEQ"
|          LT                                  -> "LT"
|          LEQ                                  -> "LEQ"
|          GT                                  -> "GT"
|          GEQ                                  -> "GEQ"
|          AND                                  -> "AND"
|          OR                                  -> "OR"
|          NOT                                  -> "NOT"
|          DOT                                  -> "DOT"
|          LBRACKET                      -> "LBRACKET"
|          RBRACKET                      -> "RBRACKET"
|          BAR                                  -> "BAR"
|          IF                                  -> "IF"
|          ELSE                              -> "ELSE"
|          FOR                                  -> "FOR"
|          WHILE                              -> "WHILE"
|          RETURN                              -> "RETURN"
|          INT                                  -> "INT"
|          FLOAT                              -> "FLOAT"
|          BOOL                              -> "BOOL"
|          CHAR                              -> "CHAR"
|          VOID                              -> "VOID"
|          NULL                              -> "NULL"
|          TRUE                              -> "TRUE"
|          FALSE                              -> "FALSE"
|          CLASS                              -> "CLASS"
|          CONSTRUCTOR                      -> "CONSTRUCTOR"
|          PUBLIC                              -> "PUBLIC"
```

```
|         PRIVATE                                    -> "PRIVATE"
|         EXTENDS                                    -> "EXTENDS"
|         INCLUDE                                    -> "INCLUDE"
|         THIS                                    -> "THIS"
|         BREAK                                    -> "BREAK"
|         CONTINUE                             -> "CONTINUE"
|   NEW                                 -> "NEW"
|         INT_LITERAL(i)                 -> "INT_LITERAL(" ^ string_of_int i ^ ")"
|         FLOAT_LITERAL(f)         -> "FLOAT_LITERAL(" ^ string_of_float f ^ ")"
|         CHAR_LITERAL(c)                 -> "CHAR_LITERAL(" ^ Char.escaped c ^ ")"
|         STRING_LITERAL(s)         -> "STRING_LITERAL(" ^ s ^ ")"
|         ID(s)                                    -> "ID(" ^ s ^ ")"
|         DELETE                                     -> "DELETE"
|         MODULO                                     -> "MODULO"
|          EOF                                            -> "EOF"

let string_of_token_no_id = function
LPAREN                                      -> "LPAREN"
|         RPAREN                                    -> "RPAREN"
|         LBRACE                                    -> "LBRACE"
|         RBRACE                                    -> "RBRACE"
|         SEMI                                    -> "SEMI"
|         COMMA                                    -> "COMMA"
|         PLUS                                    -> "PLUS"
|         MINUS                                    -> "MINUS"
|         TIMES                                    -> "TIMES"
|         DIVIDE                                    -> "DIVIDE"
|         ASSIGN                                    -> "ASSIGN"
|         EQ                                            -> "EQ"
|         NEQ                                            -> "NEQ"
|         LT                                            -> "LT"
|         LEQ                                            -> "LEQ"
|         GT                                            -> "GT"
|         GEQ                                            -> "GEQ"
|         AND                                            -> "AND"
|         OR                                            -> "OR"
|         NOT                                            -> "NOT"
|         DOT                                            -> "DOT"
|         LBRACKET                             -> "LBRACKET"
|         RBRACKET                             -> "RBRACKET"
|         BAR                                            -> "BAR"
|         IF                                            -> "IF"
|         ELSE                                    -> "ELSE"
|         FOR                                            -> "FOR"
|         WHILE                                    -> "WHILE"
|         RETURN                                    -> "RETURN"
|         INT                                            -> "INT"
|         FLOAT                                    -> "FLOAT"
|         BOOL                                    -> "BOOL"
```

```
|         CHAR                                  -> "CHAR"
|         VOID                                  -> "VOID"
|         NULL                                  -> "NULL"
|         TRUE                                  -> "TRUE"
|         FALSE                                  -> "FALSE"
|         CLASS                                  -> "CLASS"
|         CONSTRUCTOR                          -> "CONSTRUCTOR"
|         PUBLIC                                  -> "PUBLIC"
|         PRIVATE                                  -> "PRIVATE"
|         EXTENDS                                  -> "EXTENDS"
|         INCLUDE                                  -> "INCLUDE"
|         THIS                                  -> "THIS"
|         BREAK                                  -> "BREAK"
|         CONTINUE                          -> "CONTINUE"
|   NEW                                  -> "NEW"
|         INT_LITERAL(i)                  -> "INT_LITERAL"
|         FLOAT_LITERAL(f)          -> "FLOAT_LITERAL"
|         CHAR_LITERAL(c)                  -> "CHAR_LITERAL"
|         STRING_LITERAL(s)          -> "STRING_LITERAL"
|         ID(s)                                  -> "ID"
|         DELETE                                  -> "DELETE"
|         MODULO                                  -> "MODULO"
|          EOF                                          -> "EOF"

let token_list_to_string_endl token_list =
let rec helper last_line_number = function
(token, curr)::tail ->
let line = curr.lineno in
(if line != last_line_number then "\n" ^ string_of_int line ^ ". " else " ") ^
string_of_token token ^ helper line tail
|         [] -> "\n"
in helper 0 token_list

let token_list_to_string token_list =
let rec helper = function
(token, line)::tail ->
string_of_token_no_id token ^ " " ^ helper tail
|         [] -> "\n"
in helper token_list
```

Test Suite Code

tester.sh

```
#!/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

```

```
                        if [ $excpTestFlag -eq 0 ];        then
                                cat "$testPath"$filename$testExtension >> session_file
                        else
                                cat "$testExceptionsPath"$filename$testExtension >> session_file
                        fi
                        echo "" >> session_file
                        echo "Generated Output:" >> session_file
                        cat temp_Dice_Tester  >> session_file
                        echo "" >> session_file
                        ((fail++))
                fi
}

header(){
        echo ""
        echo "***********************************************" >> session_file
        echo "Dice Test Script Results:" >> session_file
        date >> session_file
        echo "" >> session_file
}

test_function(){
        header #func

        for testFile in "$testPath"*.dice; do

                filename=$(basename "$testFile")

                echo "==================================" >> session_file
                echo "Testing: $filename" >> session_file

                if [ "$testOption" == "-s" ]; then
                        #Create file to be tested (with tokens)
                        $diceExecPath $diceOption "$testFile" > temp_Dice_Tester
                        #Test output differences use the diff command and neglect screen output
                        diff temp_Dice_Tester "$testPath"$filename$testExtension > /dev/null
                        confirmation #function
                else #Only other option is -c or -d which perform the same function except where noted below
                        #extract filename without extension for exectuable
                        name=$(echo $filename | cut -f 1 -d '.')

                        if [ "$testOption" == "-d" ]; then
                                #run the executable and port output (stderr) to temp test file
                                #port stdout (compiler msgs) to screen with color
```

```
                echo -e -n "${CYAN}"
                $diceExecPath $diceOption "$testFile" 2> temp.ll
                echo -e -n "${NC}"
                echo ""

            else
                #Create header for any messages coming from Dice compiler

                echo -e "${CYAN}Dice Compiler Messages (if any):" >> session_file
                #run the executable and port output (stderr) to temp test file
                #port stdout (compiler msgs) to log file
                $diceExecPath $diceOption "$testFile" 2> temp.ll 1>> session_file
                echo -e "${NC}">> session_file
                echo "" >> session_file
            fi

            #Run the llvm executable and port output to temp test file
            lli temp.ll > temp_Dice_Tester

            #Send all error messages this script generates (if any) to error log file
            exec 2> $errorFile

            #Perform comparison of outputs
            diff temp_Dice_Tester "$testPath"$filename$testExtension > /dev/null
            confirmation #function
        fi
    done

    #The following portion is only to test compiler errors
    if [ "$testOption" == "-c" ] || [ "$testOption" == "-d" ] || [ "$testOption" == "-m" ] || [ $# -eq 0 ]; then

        #set flag to prevent
        excpTestFlag=1
        for testFile in "$testExceptionsPath"*.dice; do

            filename=$(basename "$testFile")

            echo "==================================" >> session_file
            echo "Testing: $filename" >> session_file

            #Only other option is -c or -d which perform the same function except where noted below
            #extract filename without extension for exectuable
```

```
            name=$(echo $filename | cut -f 1 -d '.')

            #run the executable and port error  output (stdout) to temp test file
            #port stdout (compiler msgs) to log file
            $diceExecPath $diceOption "$testFile" 1> temp_Dice_Tester 2>/dev/null

            #Perform comparison of outputs
            diff temp_Dice_Tester "$testExceptionsPath"$filename$testExtension >> /dev/null
            confirmation #function
        done

        #Test if our executable can take in command line arguments:
        filename=test-args.dice
        $diceExecPath $diceOption "$argsPath"test-args.dice 2>temp.ll
        lli temp.ll david emily phil > tempArgs
        diff tempArgs "$argsPath"test-args.dice.out >/dev/null
        confirmation
        rm tempArgs

    fi
    echo "" >> session_file

    #Verbose flag actuated
    if [ "$vFlag" == "-v" ]; then
        cat session_file
    fi

    #Copy session output to historical log
    cat session_file >> "$logFile"

    #Test status output
    echo ""
    echo -e "${GREEN}Tests Passed: $pass ${NC}"
    echo -e "${RED}Tests Failed: $fail ${NC}"
    echo "View $logFile for more information"

    #Clean up temp files
    rm temp_Dice_Tester;
    rm session_file;
}

createDice(){
    echo "Compiling dice executable"
    cd ..
    make clean 2>&1 > /dev/null
    make
```

```
        #cp dice ../Test\ Suite/Hello_World_Demo/dice
        # cd Test\ Suite
        echo "Compilation of dice executable complete"
}

#-----------Script starts flag checking here ------------------
if [ "$testOption" == "-s" ]; then
        echo "Scanner Test Started"
        createDice
        logFile=Test\ Suite/scanner_tests.log
        testPath=Test\ Suite/Scanner\ Test\ Suite/
        diceOption=-tendl
        testExtension=.ManualTokens
        test_function

elif [ "$testOption" == "-c" ] || [ "$testOption" == "-d" ] || [ "$testOption" == "-m" ]
↪  || [ $# -eq 0 ]; then
        echo "Compiler Test Started"

        if [ "$testOption" == "-m" ]; then
                if [ -f ../dice ]; then
                        echo "Skipping Dice recompilation"
                        cd ..
                else
                        createDice
                fi
        else
                createDice

        fi

        logFile=Test\ Suite/compiler_tests.log
        testPath=Test\ Suite/Compiler_Test_Suite/
        testExceptionsPath=Test\ Suite/Compiler_Test_Suite/Exceptions/
        argsPath=Test\ Suite/Compiler_Test_Suite/Args/
        diceOption=-c
        testExtension=.out
        test_function
        rm temp.ll;

else
        usage
fi

#Print out number of bash script errors and
if [ "$testOption" != "-s" ]; then
        errorLines=$(cat $errorFile | wc -l)
        mv $errorFile Test\ Suite/$errorFile
        if [ $errorLines -ne 0 ]; then
```

```
        echo "$errorLines lines of script errors reported. Please check $errorFile!"
        else
                mv Test\ Suite/$errorFile
        fi
fi

exit 0
```

test-var1.dice.out

```
42
```

test-stdlib-stringclass.dice.out

```
hi
```

test-stdlib-integerclass1.dice

```
include("stdlib");

class Two {
        public void main(char[][] args) {
        class Integer x = new Integer(128);
        print(x.num(), "\n");
      }
}
```

test-constructorInherited.dice

```
class shape {
  public int xCoord;
  public int yCoord;

  constructor(){
  this.xCoord = 0;
  this.yCoord = 0;
  }

  constructor(int x, int y){
  this.xCoord = x;
  this.yCoord = y;
  }
}

class circle extends shape {
  public int radius;

  constructor(){
          this.radius = 0;
  }
  constructor(int r){
          this.radius = r;
  }
  constructor(int x, int y, int r){
          this.radius = r;
          this.xCoord = x;
          this.yCoord = y;
  }
}

class test {
  public void main(char[][] args) {
      class circle a = new circle(0,0,7);
      print(a.xCoord);
      print(a.yCoord);
      print(a.radius);
  }
}
```

test-ifEmptyBlock2.dice.out

```
17
```

test-global1.dice.out

```
42214322
```

test-if7.dice

```
class test {
        public void main(char[][] args) {

                if(false) {
                        print("if");
                }

                else if(false) {
                        print("elseif");
                }

                else if(false) {
                        print("elseif2");
                }

                else {
                        print("else");
                }
        }
}
```

test-var3.dice

```
class test {

        public int a;

        public void print2(int x, int y) {
          print(x);
          print(y);
        }

        public void main(char[][] args) {
          int b;
          this.a = 42;
          b = 57;
          this.print2(this.a + b * 3, 77);
        }
}
```

test-classFunctionOverload1.dice.out

```
10
```

test-applicative.dice

```
class test {

        public int p(int i){
                print(i);
                return i;
        }

        public void q(int a, int b, int c){
                int total = a ;
                print(b);
                total = total + c ;
        }

        public void main(char[][] args) {
                this.q( this.p(1), 2, this.p(3));
        }
}
```

test-forEmptyBlock2.dice

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

test-if1.dice

```
class test {
        public void main(char[][] args) {
                if (true) print(42);
                print(17);
        }
}
```

test-func5.dice

```
class test {

        public void foo(int a, int b){
          int c;
          int d;
          int e;
          print(a);
          e = a + b + 10;
          print(e);
        }

        public void main(char[][] args) {
                  this.foo(1,2);
        }

}
```

test-arith5.dice

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

test-bool5.dice

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

test-constructor2.dice

```
class shape {
        public int xCoord;
        public int yCoord;

        constructor(int x, int y){
                this.xCoord = x;
                this.yCoord = y;
        }

        constructor(float x, float y){
                this.xCoord = 0;
                this.yCoord = 0;
        }
}

 class test {
         public void main(char[][] args) {
                 class shape a = new shape(5,10);
                 print (a.xCoord);
                 print (a.yCoord);
         }
 }
```

test-arithSigned2.dice.out

```
-3-3-3.000000-3.000000
```

test-classExtends2.dice

```
class person {
  public int ssn;
}

class worker extends person {
  public int workid;
}

class programmer extends worker {
        public int nerdCred;
}

class test {
  public void main(char[][] args) {
      class programmer david = new programmer();
      david.ssn = 123456789;
      david.workid = 57;
      david.nerdCred = 99;

      print(david.ssn);
      print(david.workid);
      print(david.nerdCred);
  }
}
```

test-arithSigned1.dice.out

```
-5-5-5.000000-5.000000
```

test-forEmptyBlock.dice

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

test-func5.dice.out

```
113
```

test-float.dice.out

```
1.500000
```

test-stdlib-integerclass1.dice.out

```
128
```

test-for2.dice.out

```
5432142
```

test-if4.dice

```
class test {
        public void main(char[][] args) {
          if (false)
                  print(42);
          else
                  print(8);
            print(17);
        }
}
```

test-arith7.dice

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

test-if5.dice

```
class test {
        public void main(char[][] args) {
                this.foo(3,5,6);
        }

        public void foo(int a, int b, int c) {
                 int d;
                 if (a == 3)
                   d = b;
                 else
                   d = c;
                 print(d);
                }
}
```

test-arithSigned3.dice

```
class test {
        public void main(char[][] args) {
                print(-1+3);
                print(1+-3);
                print(-1.0+3.0);
                print(1.0+-3.0);
        }
}
```

test-if7.dice.out

```
else
```

test-classGetter.dice.out

```
13
```

test-stdlib-compare.dice

```
include("stdlib");

class Two {
        public void main(char[][] args) {
        class String b = new String("phil");
        class String c = new String("khal");
        class String d = c.copy(c);
        print(b.string(), " == ", c.string(), " is ", b.compare(c));
        print(c.string(), " == ", d.string(), " is ", c.compare(d));
        }
}
```

test-class.dice.out

```
13
```

test-for1.dice.out

```
0123442
```

test-classInheritanceArgument.dice

```
class shape {
  public int xCoord;
  public int yCoord;
}

class circle extends shape {
  public int radius;
}

class test {

  public void main(char[][] args) {
      class circle a = new circle();
      this.inheritanceTest(a);
  }

  public void inheritanceTest(class shape a){
    print("pass");
  }

}
```

test-whileBreak.dice

```
class test {
	public void main(char[][] args) {
	  int i;
	  i = 5;
	  while (i > 0) {
	    print(i);
	    if(i==3){
		break;
		}
	    i = i - 1;
	  }
	}
}
```

test-while1.dice

```
class test {
        public void main(char[][] args) {
          int i;
          i = 5;
          while (i > 0) {
            print(i);
            i = i - 1;
          }
          print(42);
        }
}
```

test-fileio.dice.out

```
include("stdlib");

class Two {

        public void main(char[][] args) {
        class File a = new File("Test Suite/Compiler_Test_Suite/test-fileio.dice", true);
        char[] buf = a.readfile(243);
        a.closefile();
        print(buf);
        }
}
```

test-classExtends2.dice.out

```
1234567895799
```

test-forContinue.dice.out

```
23420
```

test-fib.dice

```
class test {

	public int fib(int x) {
		if (x < 2)
			return 1;
		return this.fib(x-1) + this.fib(x-2);
	}

	public void main(char[][] args) {
		print(this.fib(0));
		print(this.fib(1));
		print(this.fib(2));
		print(this.fib(3));
		print(this.fib(4));
		print(this.fib(5));
	}
}
```

test-bool1.dice

```
class test {
	public void main(char[][] args) {
		print(1<2);
		print(1.0<2);
		print(1<2.0);
		print(1.0<2.0);
	}
}
```

test-forBreak.dice

```
class test {
        public void main(char[][] args) {
          int i;
          for (i = 0 ; i < 5 ; i = i + 1) {
                  if(i==3){
                          break;
                  }
            print(i);
          }
          print(100);
        }
}
```

test-bool6.dice

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

test-bool4.dice.out

```
truetruetruefalse
```

test-stdlib-stringclassContains2.dice

```
include("stdlib");

class Two {
        public void main(char[][] args) {
        class String b = new String("philkhal");
        class String c = new String("butts");
        print(b.contains(c));
        }
}
```

test-classGetter.dice

```
class shape {
  public int xCoord;
  public int yCoord;

  public int getX(){
          return this.xCoord;
  }
  public int getY(){
           return this.yCoord;
  }

}

class test {
  public void main(char[][] args) {
      class shape a = new shape();
      a.xCoord = 1;
      a.yCoord = 3;
      print(a.getX());
      print(a.getY());
  }
}
```

test-var3.dice.out

```
21377
```

test-forContinue.dice

```
class test {
        public void main(char[][] args) {
          int i;
          for (i = 0 ; i < 5 ; i = i + 1) {
                  if(i<2){ continue; }
                  else{
            print(i);
            }
          }
          print(20);
        }
}
```

test-stdlib-stringclassReverse.dice.out

```
olleh
```

test-while1.dice.out

```
5432142
```

test-float.dice

```
class test {
  public void main(char[][] args) {
      float a = 1.5;
      print(a);

    }
}
```

test-arith5.dice.out

```
10
```

test-array4.dice

```
class shape {
        public int x;
        public int y;

        constructor(int a, int b){
        this.x = a;
        this.y = b;
        }

}

class test {
        public void main(char[][] args) {
                class shape[] a = new class shape[5];
                class shape b = new shape(2,3);
                a[1] = b;
                print(a[1].x);
        }
}
```

test-arithSigned1.dice

```
class test {
	public void main(char[][] args) {
		print(-15/3);
		print(15/-3);
		print(-15.0/3.0);
		print(15.0/-3.0);
	}
}
```

test-if2.dice.out

```
4217
```

test-stdlib-concat.dice

```
include("stdlib");

class Two {
        public void main(char[][] args) {
        class String b = new String("phil");
        class String c = new String("khal");
        class String a = b.concat(c);
        print(b.string(), "\n");
        print(c.string(), "\n");
        print(a.string(), "\n");
        }
}
```

test-classReturnObjects.dice.out

```
12
```

test-if8.dice

```
class test {
	public void main(char[][] args) {

		if(false) {
			print("if");
		}

		else if(true) {
			print("elseif");
		}

		else if(false) {
			print("elseif2");
		}

		else {
			print("else");
		}
	}
}
```

test-stmts1.dice

```
class test {
        public void main(char[][] args) {
                print(this.foo(1,42));
                 print(this.foo(0,37));
        }

        public int foo(int a, int b) {
          int i;
          int j = b;
          if ( a == 1)
            return b + 3;
          else
            for (i = 0 ; i < 5 ; i = i + 1)
               j = j + 5;
          return j;
        }
}
```

test-if6.dice.out

```
42278
```

test-classExtendsGetter.dice.out

```
13
```

test-ops1.dice.out

```
3-125099falsetrue99truefalse99truefalse99truetruefalse99falsetrue99falsetruetrue
```

test-arith4.dice

```
(* Test side-effect sequence in a series of statement *)

class test {
        public int g;

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

          int l;
          l = 1;
          print(l);

          this.g = 3;
          print(this.g);

          l = 5;
          print(l+100);

          this.g = 7;
          print(this.g+100);
        }
}
```

test-func3.dice.out

```
42171928
```

test-class.dice

```
class shape {
  public int xCoord;
  public int yCoord;

  constructor (){
  }
}

class test {
  public void main(char[][] args) {
      class shape a = new shape();
      a.xCoord = 1;
      a.yCoord = 3;
      print(a.xCoord);
      print(a.yCoord);
  }
}
```

test-bool9.dice.out

```
truetruetruefalsefalsetruetruefalse
```

test-whileContinue.dice.out

```
543
```

test-stdlib-copy.dice.out

```
philkhalkhal
```

test-stdlib-integerclass2.dice.out

```
128
```

test-classExtends.dice

```
class shape {
  public float xCoord;
  public float yCoord;
}

class circle extends shape {
  public float radius;
}

class test {
  public void main(char[][] args) {
      class circle a = new circle();
      a.xCoord = 1.5;
      print(a.xCoord);
  }
}
```

test-if3.dice

```
class test {
	public void main(char[][] args) {
		if (false)
			print(42);
		print(17);
	}
}
```

test-bool8.dice.out

```
falsetrue
falsefalse
```

test-scope.dice.out

```
12321
```

test-constructor1.dice

```
class shape {
        public int xCoord;
        public int yCoord;

        constructor(){
                this.xCoord = 0;
                this.yCoord = 0;
        }

        constructor(int x, int y){
                this.xCoord = x;
                this.yCoord = y;
        }
}

 class test {
         public void main(char[][] args) {
                 class shape a = new shape();
                 class shape b = new shape(5,10);
                 print (a.xCoord);
                 print (a.yCoord);
                 print (b.xCoord);
                 print (b.yCoord);
         }
 }
```

test-stdlib-concat.dice.out

```
phil
khal
philkhal
```

test-forEmptyBlock2.dice.out

```
1
```

test-if4.dice.out

```
817
```

test-array.dice.out

```
04
```

test-array2.dice.out

```
1.5000004.500000
```

test-objectDeclarationInheritance.dice.out

```
pass
```

test-if5.dice.out

```
5
```

test-forEmptyBlock.dice.out

```
1
```

test-var4.dice.out

```
1242
```

test-whileContinue.dice

```
class test {
        public void main(char[][] args) {
          int i;
          i = 6;
          while (i > 0) {
                   i = i - 1;

            if(i<3){
                    continue;
            }

           print(i);
          }
        }
}
```

test-array3.dice

```
class test {
        public void main(char[][] args) {
                int[] a = new int[10];
                a[0] = 1;
                print(a[0]);
                a[0] = 10;
                print(a[0]);
                a[9] = 2;
                print(a[9]);
        }
}
```

test-if3.dice.out

```
17
```

test-arith6.dice

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

test-helloTwice.dice.out

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

test-stdlib-stringclassLength.dice.out

```
9
```

test-bool3.dice.out

```
falsetruefalsetrue
```

test-hello.dice

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

test-array.dice

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

test-exit.dice

```
class test {
        public void main(char[][] args) {

                print(1);
                exit(1);
                print(2);

        }
 }
```

test-helloTwice.dice

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

test-arithSigned2.dice

```
class test {
        public void main(char[][] args) {
                print(-1*3);
                print(1*-3);
                print(-1.0*3.0);
                print(1.0*-3.0);
        }
}
```

test-cyclicalIncludes2.dice

```
include("Test Suite/Compiler_Test_Suite/test-cyclicalIncludes.dice");

class test2 {
        constructor(){
                this.output2();
        }
        public void output2(){
                print("b");
        }
}
```

test-if2.dice

```
class test {
        public void main(char[][] args) {
                  if (true) print(42);
                  else print(8);
                    print(17);
          }
}
```

test-constructorDefault.dice

```
class shape {
        public int xCoord;
        public int yCoord;

}

 class test {
         public void main(char[][] args) {
                 class shape a = new shape();
                 a.xCoord = 5;
                 print (a.xCoord);
         }
 }
```

test-var1.dice

```
class test {
        public void main(char[][] args) {

          int a;
          a = 42;
          print(a);
        }
}
```

test-arithSigned4.dice.out

```
-44-4.0000004.000000
```

test-ifEmptyBlock.dice.out

```
17
```

test-stdlib-compare.dice.out

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

test-cyclicalIncludes.dice.out

```
ba
```

test-bool7.dice.out

```
truefalsefalsefalse
truetruetruefalse
```

test-classSetter.dice.out

```
13
```

test-stdlib-stringclassReverse.dice

```
include("stdlib");

class Test {
    public void main(char[][] args) {
        class String a = new String("hello");
        class String reverse = a.reverse();

        print(reverse.string());
    }
}
```

test-factorialRecursive.dice

```
class Factorial {

   public void main(char[][] args) {
      print(this.factorial(5));
   }

   public int factorial(int n) {
    int temp;
    if(n <= 1) return 1;
    temp = n * this.factorial(n - 1);
    return temp;
   }
}
```

test-classInheritanceArgument.dice.out

```
pass
```

test-constructorInherited.dice.out

```
007
```

test-bool8.dice

```
class test {
        public void main(char[][] args) {
                print(not true);
                print(not false);
                print("\n");
                print(not true and true);
                print(not (true and true));
        }
}
```

test-classFunctionOverload.dice

```
class shape {
  public int xCoord;
  public int yCoord;

  constructor(){
  this.xCoord = 0;
  this.yCoord = 0;
  }

  constructor(int x, int y){
  this.xCoord = x;
  this.yCoord = y;
  }

  public int getArea(){
    return 10;
  }
}

class circle extends shape {
  public int radius;

  constructor(){
          this.radius = 0;
  }
  constructor(int r){
          this.radius = r;
  }
  constructor(int x, int y, int r){
          this.radius = r;
          this.xCoord = x;
          this.yCoord = y;
  }

  public int getArea(){
    return 3*this.radius*this.radius;
  }
}

class test {
  public void main(char[][] args) {
      class circle a = new circle(0,0,2);
      print(a.getArea());
  }
}
```

test-stdlib-stringclassContains.dice.out

```
true
```

test-arith8.dice

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

test-array4.dice.out

```
2
```

test-stdlib-copy.dice

```
include("stdlib");

class Two {
        public void main(char[][] args) {
        class String b = new String("phil");
        class String c = new String("khal");
        class String d = c.copy(c);
        print(b.string());
        print(c.string());
        print(d.string());
        }
}
```

test-arith7.dice.out

```
3
```

test-classFunctionOverload1.dice

```
class shape {
  public int xCoord;
  public int yCoord;

  constructor(){
  this.xCoord = 0;
  this.yCoord = 0;
  }

  constructor(int x, int y){
  this.xCoord = x;
  this.yCoord = y;
  }

  public int getArea(){
    return 10;
  }
}

class circle extends shape {
  public int radius;

  constructor(){
          this.radius = 0;
  }
  constructor(int r){
          this.radius = r;
  }
  constructor(int x, int y){
          this.radius = 0;
          this.xCoord = x;
          this.yCoord = y;
  }

  public int getArea(){
    return 3*this.radius*this.radius;
  }
}

class test {
  public void main(char[][] args) {
      class shape a = new shape(0,0);
      print(a.getArea());
  }
}
```

test-stdlib-stringclassContains.dice

```
include("stdlib");

class Two {
	public void main(char[][] args) {
	class String b = new String("philkhal");
	class String c = new String("khal");
	print(b.contains(c));
	}
}
```

test-factorialRecursive.dice.out

```
120
```

test-stdlib-integerclass2.dice

```
include("stdlib");

class Two {
        public void main(char[][] args) {
        class Integer x = new Integer(128);
        class String str = x.toString();
        print(str.string(), "\n");
        }
}
```

test-bool6.dice.out

```
truefalse
```

test-cyclicalIncludes.dice

```
include("Test Suite/Compiler_Test_Suite/test-cyclicalIncludes2.dice");

class test {
        public void main(char[][] args) {
        class test2 a = new test2();
        this.output();
        }

        public void output(){
                print("a");
        }
}
```

test-bool1.dice.out

```
truetruetruetrue
```

test-stdlib-stringclass3.dice

```
include("stdlib");

class test{

        private class String x;

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

        class String a = new String("goodBye");
        this.x = a;
        print(this.x.string());

        }
}
```

test-arith3.dice

```
(* Test left-to-right evaluation of expressions *)

class test {

        public int a; (* Global variable *)

        public int inca() {
                this.a = this.a + 1;  (* Increment a; return its new value *)
                return this.a;
        }

        public void main(char[][] args) {
                  this.a = 42;    (* Initialize a *)
                  print(this.inca() + this.a);
        }
}
```

test-emptyBlock.dice

```
class test {
	public void main(char[][] args) {
		{
		(* Nothing in the following blocks*) {} {}
		}
		{ null; }
		print(1);
	}
}
```

test-intOverflow.dice.out

```
passpass
```

test-stdlib.dice.out

```
hi
```

test-classFunctionOverload.dice.out

```
12
```

test-exit.dice.out

```
1
```

test-if1.dice.out

```
4217
```

test-stdlib-stringclass2.dice

```
include("stdlib");

class test {
        public void main(char[][] args) {
        class String s = new String("StringDoesn'tStartWithH");
        print(s.string());
        }
}
```

test-arith6.dice.out

```
50
```

test-stdlib-stringclassLength.dice

```
include("stdlib");

class Two {
	public void main(char[][] args) {
	class String s = new String("123456789");
	print(s.length());
	}
}
```

test-stdlib-stringclassContains2.dice.out

```
false
```

test-ops1.dice

```
class test {
  public void main(char[][] args) {
      print(1 + 2);
      print(1 - 2);
      print(1 * 2);
      print(100 / 2);
      print(99);
      print(1 == 2);
      print(1 == 1);
      print(99);
      print(1 != 2);
      print(1 != 1);
      print(99);
      print(1 < 2);
      print(2 < 1);
      print(99);
      print(1 <= 2);
      print(1 <= 1);
      print(2 <= 1);
      print(99);
      print(1 > 2);
      print(2 > 1);
      print(99);
      print(1 >= 2);
      print(1 >= 1);
      print(2 >= 1);
  }
}
```

test-stdlib-stringclass.dice

```
include("stdlib");

class Two {
        public void main(char[][] args) {
        class String s = new String("hi");
        print(s.string());
        }
}
```

test-arith2.dice

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

test-float-max.dice

```
class test {
  public void main(char[][] args) {
      float a = 0.01175494;
      float b = 1010123.45;
      print(a);
      print("\n");
      print(b);

     }
}
```

test-arith1.dice

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

test-stdlib-stringclass3.dice.out

```
goodBye
```

test-ifEmptyBlock2.dice

```
class test {
        public void main(char[][] args) {
                if (false){}
                else {}
                print(17);
        }
}
```

test-array3.dice.out

```
1102
```

test-arithSigned4.dice

```
class test {
	public void main(char[][] args) {
		print(-1-3);
		print(1--3);
		print(-1.0-3.0);
		print(1.0--3.0);
	}
}
```

test-classSetter.dice

```
class shape {
  public int xCoord;
  public int yCoord;

  public void setX(int x){
          this.xCoord = x;
  }
  public void setY(int y){
           this.yCoord = y;
  }

}

class test {
  public void main(char[][] args) {
      class shape a = new shape();
      a.setX(1);
      a.setY(3);
      print(a.xCoord);
      print(a.yCoord);
  }
}
```

test-classExtendsSetter.dice

```
class shape {
  public int xCoord;
  public int yCoord;

  public void setX(int x){
          this.xCoord = x;
  }
  public void setY(int y){
           this.yCoord = y;
  }

}

class circle extends shape {
  public int radius;

}

class test {
  public void main(char[][] args) {
      class circle a = new circle();
      a.setX(1);
      a.setY(3);
      print(a.xCoord);
      print(a.yCoord);
  }
}
```

test-gcd.dice

```
class test {

	public void main(char[][] args) {
		print(this.gcd(2,14));
		print(this.gcd(3,15));
		print(this.gcd(99,121));
	}

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

test-bool7.dice

```
class test {
        public void main(char[][] args) {
                print(true and true);
                print(false and true);
                print(true and false);
                print(false and false);
                print("\n");
                print(true or true);
                print(false or true);
                print(true or false);
                print(false or false);
        }
}
```

test-classExtendsGetter.dice

```
class shape {
  public int xCoord;
  public int yCoord;

  public int getX(){
          return this.xCoord;
  }
  public int getY(){
           return this.yCoord;
  }

}

class circle extends shape {
  public int radius;

}

class test {
  public void main(char[][] args) {
      class circle a = new circle();
      a.xCoord = 1;
      a.yCoord = 3;
      print(a.getX());
      print(a.getY());
  }
}
```

test-func4.dice.out

```
371
```

test-constructor1.dice.out

```
00510
```

test-fib.dice.out

```
112358
```

test-forBreak.dice.out

```
012100
```

test-func3.dice

```
class test {
	public void main(char[][] args) {
		this.printem(42,17,192,8);
	}

	public void printem(int a, int b, int c, int d) {
		print(a);
		print(b);
		print(c);
		print(d);
	}
}
```

test-scope.dice

```
class test {
        public void main(char[][] args) {
          int a;
          a = 1;
          {
                  int b = 2;
                  {
                          int c = 3;
                          print(a);
                          print(b);
                          print(c);
                  }
                  print(b);
          }
          print(a);
        }
}
```

test-objectDeclarationInheritance.dice

```
class A {}
class B extends A {}
class C extends B {}

class test {

        public void main(char[][] args) {
                class A myCObj = new C();
                print("pass");
         }
 }
```

test-bool9.dice

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

test-if8.dice.out

```
elseif
```

test-hello.dice.out

```
Hello, World!
```

test-fileio.dice

```
include("stdlib");

class Two {

        public void main(char[][] args) {
        class File a = new File("Test Suite/Compiler_Test_Suite/test-fileio.dice", true);
        char[] buf = a.readfile(243);
        a.closefile();
        print(buf);
        }
}
```

test-arith3.dice.out

```
86
```

test-float-max.dice.out

```
0.011755
1010123.450000
```

test-var4.dice

```
class test {
        public int a;

        public void foo(int b) {
          int c;
          c = this.a;
          print(c);
          this.a = b;
          print(this.a);
        }

        public void main(char[][] args) {
          this.a = 12;
          this.foo(42);
        }
}
```

test-cyclicalIncludes2.dice.out

```
ba
```

test-classExtendsSetter.dice.out

```
13
```

test-bool4.dice

```
class test {
	public void main(char[][] args) {
		print(1<=2);
		print(1<=1);
		print(1<=2.0);
		print(2.1<=2.0);
	}
}
```

test-bool2.dice

```
class test {
	public void main(char[][] args) {
		print(1>2);
		print(1.0>2);
		print(1>2.0);
		print(1.0>2.0);
	}
}
```

test-classExtends.dice.out

```
1.500000
```

test-gcd.dice.out

```
2311
```

test-bool2.dice.out

```
falsefalsefalsefalse
```

test-func4.dice

```
class test {
	public int a;

	constructor() {}

	public int inca() {
		this.a = 124;
		return this.a + 124;
	}

	public int add2(int x, int y) {
		return x + y;
	}

	public void main(char[][] args) {
		class test b = new test();
		  print(b.add2(b.inca(), 123));
	}
}
```

test-emptyBlock.dice.out

```
1
```

test-constructor2.dice.out

```
510
```

test-for2.dice

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

test-array2.dice

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

test-constructorDefault.dice.out

```
5
```

test-applicative.dice.out

```
132
```

test-stmts1.dice.out

```
4562
```

test-global1.dice

```
class test {
  public int a;
  public int b;

  public void printa(){
    print(this.a);
  }

  public void printb(){
    print(this.b);
  }

  public void incab(){
    this.a = this.a + 1;
    this.b = this.b + 1;
  }

  public void main(char[][] args) {
      this.a = 42;
      this.b = 21;
      this.printa();
      this.printb();
      this.incab();
      this.printa();
      this.printb();
  }
}
```

test-intOverflow.dice

```
class test {
  public void main(char[][] args) {
      int a = 2147483648; (*More than an int can hold should overflow*)
      if(a<2147483647){
              print("pass");
      }
      else{
      print(a);
      }

      int b = -2147483649; (*More than an int can hold should overflow*)
      if(b>-2147483648){
              print("pass");
      }
      else{
      print(b);
      }
     }
}
```

test-if6.dice

```
class test {
        public void main(char[][] args) {
          if (true){
                  if(true)
                          print(42);
                  print(27);
          }
          else
                  print(8);

          if (false){
                  if(true)
                          print(42);
                  print(27);
          }
          else
                  print(8);

        }
}
```

test-ifEmptyBlock.dice

```
class test {
        public void main(char[][] args) {
                if (true){}
                print(17);
        }
}
```

test-arith1.dice.out

```
20
```

test-arith4.dice.out

```
13105107
```

test-whileBreak.dice.out

```
543
```

test-classReturnObjects.dice

```
class shape {
  public int xCoord;
  public int yCoord;

  constructor (){
  this.xCoord = 1;
  this.yCoord = 2;
  }

}

class test {
  public void main(char[][] args) {
      class shape a = this.returnMe();
      print(a.xCoord);
      print(a.yCoord);
  }

  public class shape returnMe(){
    class shape b = new shape();
    return b;
  }
}
```

test-stdlib-stringclass2.dice.out

```
StringDoesn'tStartWithH
```

test-intMax.dice.out

```
2147483647
-2147483648
```

test-arith2.dice.out

```
11
```

test-for1.dice

```
class test {
        public void main(char[][] args) {
          int i;
          for (i = 0 ; i < 5 ; i = i + 1) {
            print(i);
          }
          print(42);
        }
}
```

test-bool3.dice

```
class test {
        public void main(char[][] args) {
                print(1>=2);
                print(1>=1);
                print(1>=2.0);
                print(2.0>=2.0);
        }
}
```

test-arithSigned3.dice.out

```
2-22.000000-2.000000
```

test-arith8.dice.out

```
20.000000
2.500000
```

test-intMax.dice

```
class test {
  public void main(char[][] args) {
      int a = 2147483647;
      int b = -2147483648;
      print(a);
      print("\n");
      print(b);
     }
}
```

test-bool5.dice.out

```
falsetrue
```

test-args.dice.out

```
davidemilyphil4
```

test-args.dice

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

E-test-cyclicalIncludesDuplicate.dice.out

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

E-test-objectCreation2.dice.out

E-test-scope3.dice

```
class test {

	public void main(char[][] args) {
		int x;
		for(x = 0; x < 3; x = x+1){
			int y = 10;
			print(y);
		}
		print(y);
	}
}
```

E-test-objectCreation2.dice

```
class Bar {
constructor(char c, float f) {}
}

class Foo {
constructor(bool b, char c, float f) {}
constructor(int a, bool b, char c, float f) {}
}

class test {
public void main(char[][] args) {
char myc = 'z';
float myf = 4.5;
class Bar myb = new Bar(myc, myf);
class Foo myFooObj = new Foo(5, true, myc, myf);
}
}
```

E-test-objectAssignMistmatch.dice.out

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

E-test-cyclicalIncludes.dice.out

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

E-test-scope1.dice.out

```
UndefinedID(x)
```

E-test-objectCreation1.dice.out

E-test-scope2.dice.out

```
UndefinedID(x)
```

E-test-assignMismatch.dice.out

```
AssignmentTypeMismatch (float,int)
```

E-test-duplicate.dice

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

E-test-scope3.dice.out

```
UndefinedID(y)
```

E-test-objectCreation4.dice

```
class Bar {
constructor(char c, float f) {}
constructor(bool b, char c, float f) {}
}
class Foo {
constructor(int a, bool b, char c, float f) {}
}
 class test {
 public void main(char[][] args) {
 char myc = 'z';
 float myf = 4.5;
 class Bar myb = new Bar(myc, myf);
 class Foo myFooObj = new Foo(5, true, myc, myf);
 }
 }
```

E-test-constructor.dice

```
class Foo {
constructor(char c, float f) {}
constructor(bool b, char c, float f) {}
}

class test {
public void main(char[][] args) {
int mya = 2;
bool myb = false;
char myc = 'z';
float myf = 3.5;
class Foo myFooObj = new Foo(mya, myb, myc, myf);
}
}
```

E-test-scope2.dice

```
class test {

	public void main(char[][] args) {
		if(true){
			int x = 10;
			print(x);
		}
		print(x);
	}
}
```

E-test-constructor.dice.out

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

E-test-noReturn.dice

```
class test {

        public int increment(int x){
                x = x+1;
        }
        public void main(char[][] args) {
               int x = this.increment(5);
        }
}
```

E-test-cyclicalIncludesDuplicate2.dice.out

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

E-test-objectCreation1.dice

```
class Bar {
constructor(char c, float f) {}
constructor(bool b, char c, float f) {}
}

class Foo {
constructor(bool a, int b) {}
constructor(int a, bool b, char c, float f) {}
}

 class test {
 public void main(char[][] args) {
 int mya = 2;
 bool myb = false;
 char myc = 'z';
 float myf = 3.5;
 class Foo myFooObj = new Foo(mya, myb, myc, myf);
 }
 }
```

E-test-cyclicalIncludes.dice

```
include("Test Suite/Compiler_Test_Suite/test-cyclicalIncludes.dice");

class test {
        public void main(char[][] args) {
        this.output();
        }

        public void output(){
                print("a");
        }
}
```

E-test-undefinedClass2.dice

```
class Foo {}

class Bar {}

class test {
public void main(char[][] args) {
class Baz b;
}
}
```

E-test-mainClassNotDefined.dice

```
class test{

}
```

E-test-privateFieldsAccess.dice

```
class shape {
        private int area;

        constructor(){
        this.area = 100;
        }

        public void setArea(int x){
                this.area = x;
        }

        public int getArea(){
                return this.area;
        }

}

 class test {
         public void main(char[][] args) {
                 class shape a = new shape();
                 a.area = 50;

                 }
 }
```

E-test-duplicate.dice.out

```
DuplicateLocal: myc
```

E-test-stdlib-overload.dice.out

```
CannotUseReservedFuncName(print)
```

E-test-noReturn.dice.out

```
Exceptions.AllNonVoidFunctionsMustEndWithReturn(test.increment)
```

E-test-undefinedClass.dice

```
class D {
  public void main(char[][] args) {}
}
class A extends B {}
class B extends C {}
class C extends D {}
class G extends H {}
class I extends H {}
```

E-test-objectAssignMistmatch.dice

```
class A {}
class B extends A {}
class C {}
class test {
public void main(char[][] args) {
class A myBObj = new B();
 class B mySecondBObj = new C();
 }
 }
```

E-test-privateFunctionAccess.dice.out

```
CannotAccessPrivateFunctionInNonProperScope(something.hi,something,test)
```

E-test-objectCreation3.dice.out

E-test-objectCreation3.dice

```
class Foo {}

class Baz {}

class test {
public void main(char[][] args) {
class Baz b;
}
}
```

E-test-privateFieldsAccess.dice.out

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

E-test-assignMismatch2.dice.out

```
AssignmentTypeMismatch (int,float)
```

E-test-scope1.dice

```
class test {

	public void main(char[][] args) {
		{
			int x = 10;
			print(x);
		}
		print(x);
	}
}
```

E-test-stdlib-overload.dice

```
class test {

        public void print(){

        }

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

  }
}
```

E-test-objectCreation4.dice.out

```
```

E-test-cyclicalIncludesDuplicate.dice

```
include("Test
↪ Suite/Compiler_Test_Suite/Exceptions/E-test-cyclicalIncludesDuplicate2.dice");

class test {
        public void main(char[][] args) {

        }
}
```

E-test-undefinedClass2.dice.out

```
UndefinedClass: Baz
```

E-test-cyclicalIncludesDuplicate2.dice

```
include("Test
↪ Suite/Compiler_Test_Suite/Exceptions/E-test-cyclicalIncludesDuplicate.dice");

class test {

}
```

E-test-assignMismatch2.dice

```
class test {
  public void main(char[][] args) {
      int a;
      a = 1.0;
      print(a);
  }
}
```

E-test-privateFunctionAccess.dice

```
class shape {
}

class something {

        private void hi(){
        }
}

 class test {
         public void main(char[][] args) {
                 class something a = new something();
                 a.hi();
                 }
 }
```

E-test-constructor1.dice

```
class shape {
        public int xCoord;
        public int yCoord;

        constructor(int x, int y){
                xCoord = 0;
                yCoord = 0;
        }

        constructor(int x, int y){
                xCoord = x;
                yCoord = y;
        }
}

 class test {
         public void main(char[][] args) {
                 (* Constructor clash *)
         }
 }
```

E-test-assignMismatch.dice

```
class test {
  public void main(char[][] args) {
      float a;
      a = 1;
      print(a);
  }
}
```

E-test-mainClassNotDefined.dice.out

```
MainNotDefined
```

E-test-undefinedClass.dice.out

```
UndefinedClass: H
```

E-test-constructor1.dice.out

```
DuplicateConstructor
```

test_pretty.dice

```
class test  {
        public void main (char[][] args) {
                print("Hello World");
        }
}
```

test.dice

```
class test {
	public void main(char[][] args) {
		print("Hello World");
	}
}
```

primitives.dice

```
class testPrims {
        public int a;
        public float b;
        private char c;
        private bool d;
        public void main(char[][] args) {
        int e;
        float f;
        char g;
        bool h;
        a = -2147483648;
        e =  2147483647;
        b = 1.0;
        f = 2.222222;
        c = '0';
        g = '\t';
        d = true;
        h = false;
        }
}
```

test_pretty.dice.ManualTokens

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

Demo_Animals.dice

```
include("stdlib");

class Animal{
	public int weight;
	constructor(){
		this.weight = 0;
	}

	constructor(int w){
		this.weight = w;
	}

	public void move(){
		print("Animals move in many ways");
	}
}

class Bird extends Animal {
	public int maxFlyingHeight;

	constructor(){
		this.weight = 0;
		this.maxFlyingHeight = 0;
	}

	constructor(int w, int h){
		this.weight = w;
		this.maxFlyingHeight = h;
	}

	public void move(){
		print("Birds fly!");
	}

}

class Dog extends Animal {
	public int speed;

	constructor(){
		this.weight = 0;
		this.speed = 0;
	}

	constructor(int w, int s){
		this.weight = w;
		this.speed = s;
```

```
        }

        public void move(){
                print("Dogs run!");
        }
}

class Stephen extends Animal {
        private bool isDone;

        constructor() {
                this.isDone = true;
        }

        constructor(bool isDone) {
                this.isDone = isDone;
        }

        public void move() {
                if(not this.isDone) {
                        print("I am a techer!");
                } else {
                        print("Also my favorite number is 42");
                }
                this.isDone = true;
        }

}

class Snake extends Animal {
        public int slitherSpeed;

        constructor(){
                this.weight = 0;
                this.slitherSpeed = 0;
        }

        constructor(int w, int s){
                this.weight = w;
                this.slitherSpeed = s;
        }

        public void move(){
                print("Snakes slither!");
        }
}

class Marnie extends Dog {
        public int cuteness;
```

```

        constructor(){
                this.weight = 0;
                this.speed = 0;
        }

        constructor(int w, int s){
                this.weight = w;
                this.speed = s;
        }

        constructor(int w, int s, int c){
                this.weight = w;
                this.speed = s;
                this.cuteness = c;
        }

        public void move(){
                class File a = new File("Demo/marnie1.txt", true);
        char[] buf = a.readfile(4500);
        a.closefile();
        print(buf);
        print("\n");
        }
}

class test {
        private bool isDone;
        public void main(char[][] args) {
                this.logo();
                this.isDone = false;

                bool keepGoing = true;
                while(keepGoing){
                        this.animalsToChoose();
                        char[] buf = input();
                        print("\n");

                        int choice = this.getInt(buf[0]);

                        if(choice==5)
                                break;
                        else
                                this.printMovement(choice);

                        print("\n");
                }

        class Marnie a = new Marnie();
```

```
        a.move();
        }

        public int getInt(char num){
                if(num=='1')
                        return 1;
                else if(num=='2')
                        return 2;
                else if(num=='3')
                        return 3;
                else if(num=='4')
                        return 4;
                else if(num=='5')
                        return 5;

                return 0;

        }

        public void printMovement(int choice){

                class Animal b = new Bird();
                class Animal d = new Dog();
                class Animal s = new Snake();
                class Animal stephen = new Stephen(this.isDone);

                if(choice == 1)
                        b.move();
                else if(choice == 2)
                        d.move();
                else if(choice == 3)
                        s.move();
                else if(choice == 4) {
                        stephen.move();
                        this.isDone = true;
                }
                else
                        print("Animal not selected!\n");

                print("\n");
        }

        public void animalsToChoose(){
                print("1-Bird\n2-Dog\n3-Snake\n4-Stephen\n5-Exit\nPlease choose an animal
                ↪  or exit(by selecting a number):");

        }

        public void logo(){
```

```
                class File a = new File("Demo/logo.txt", true);
        char[] buf = a.readfile(4500);
        a.closefile();
        print(buf);

        int i;
                for(i=0;i<3;i=i+1){
                print("\n");
                }

                print("Welcome to the animal farm!\n\n");
        }
}
```

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.