Margaret Mallernee mlm2299
Zachary Silber zs2266
Michael Tong mct2159
Richard Zhang rz2345
Joshua Zweig jmz2135

Programming Languages and Translators, Spring 2017

C% Final Report

Contents

Chapter 1

Introduction

C% is a C-like language purposed for applications in cryptography. The issue in the current state of the art cryptography is not in the theory of Diffie Hellman, but rather in the cryptology of how the protocol is implemented in specific applications and contexts. Easing the implementation of cryptographic protocols for encryption is a large field in current research, with obvious benefits to society at large. The easier these protocols are to implement, the more likely they are to be implemented correctly or at all, thereby reducing security risks. Additionally, this ease of use primes C% for use as a teaching language for beginners in cryptographic protocol implementation.

With this in mind, C% provides built in types, operators and functionality that are designed to relieve the programmer from the burden of carrying around large primes and taking care of many modular calculations. C% programs follow the basic structure of C, including many of the same basic types, operators, and control flow statements. Also, C% leaves heap memory management to the user with `malloc()` and `free()`, just as in C. Finally, C% handles user input with `printf()` and `scanf()`.

1.1 Language Goals

1.1.1 Wrap Large Number Arithmetic

One of the main benefits of C% to users is the increased ease of use through encapsulation of the OpenSSL BigNum library. This library is the industry standard for dealing with large numbers as needed in cryptography for holding large primes. However, it is incredibly difficult to use, so C% wraps the functionality of this library in its more intuitive type system. This allows the user to implement cryptographic protocols much more easily.

1.1.2 Readability

C% was designed to allow users to save time in implementing common cryptographic protocols by encapsulating the operations over the groups of integers over a prime modulus and points over an elliptic curve. With C%'s robust type system, the user no longer has to define functions or consult external libraries to perform operations as simple as addition. This, in addition to the built in big num capability and intuitive type names, makes implementations of cryptographic protocols shorter and easier to read.

1.1.3 Encourage Correctness

Ease of use and readability are goals that clearly benefit the user, but perhaps the most important benefit of these is their ability to encourage correctness. C% was motivated

by the difficulty of using existing cryptography libraries and the vulnerabilities introduced when developers attempt to add security features without truly understanding how to properly integrate them into their software. C% is a language aimed at providing the core functionality of C with cryptographic types allowing secure protocols to be easily integrated into programs. This continuity encourages these protocols to be implemented correctly.

1.1.4 Extensibility

C% was designed and written with extensibility in mind. It's C-like syntax can be expanded to include more of the functionality C provides, and the standard library of readily available cryptographic protocols can be easily expanded to include more examples of modular and elliptic curve cryptography. This is especially useful for the language's application as a teaching language, while also being convenient for all users. Finally, the language is primed for extension to other mathematical foundations of cryptography, such as the potential for arithmetic over more general finite fields.

1.2 Background

1.2.1 Modular arithmetic

Most readers are likely familiar with modular arithmetic. It is the arithmetic of integers where only the remainder of the number when divided by a fixed modulus is considered, so for example one would take $3 * 5 = 15$ and take the remainder when divided by the modulus 7, which is 1.

Modular arithmetic lies at the heart of cryptography, which concerns itself with sending information securely such that it cannot be read by any ordinary person unless they have access to some other private information (typically referred to as a key). Indeed, the security of modern cryptography protocols relies on various properties (or at least properties which are widely believed, but yet unproven) of the multiplication of integers when taken with a large (usually prime) modulus. For example, many security protocols rely on the *hardness of the discrete log problem*: given a prime modulus p and two integers α and β taken mod p, find an integer k for which $\alpha^k \equiv \beta \pmod{p}$.

1.2.2 General setting

Much of cryptography has historically been studied and implemented within this framework of modular arithmetic. However, from a more abstract point of view, the core of what we are working with here is a set of things (in this case, non-negative integers less than p) with a binary operation (in this case, modular multiplication). In particular, the binary operation of modular multiplication enjoys nice properties: it is associative (that is, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$), it has an identity element (there is some element e so that for every a, we have $e \cdot a = a \cdot e = a$), and it can be inverted (for every a, there is some other element b so that $a \cdot b = b \cdot a = e$). Indeed, in abstract algebra, a set equipped with a binary operation satisfying these properties is called a group, and cryptography can be done with general groups instead of just on modular arithmetic. This ties nicely into our implementation of elliptic curves.

1.2.3 Elliptic curves

Our language will offer tremendous support for a newer brand of cryptography where the group in question consists of points lying on an object called an elliptic curve. How this

works is highly technical, and we leave the explanation of the intricacies to the cryptography appendix.

Chapter 2

Language Tutorial

2.1 Environment Setup

C% has several dependencies. First of all, it requires OCAML. Using Homebrew, one can install OCAML using `$brew install OCAML`. C% also requires LLVM 3.6 or higher, which can also be installed using Homebrew. The user should also ensure that the LLC variable in `cmc.sh` is correct, linking to their LLVM compiler, which will generate machine code from LLVM IR code that C%'s compiler will produce. C% also relies on python, openSSL, clang, and shell scripting (bash, sh), most of which should be already installed on the user's machine.

2.2 Using the Compiler

. Once inside the C% (C-) directory, type `make all`. Once the make is complete, you will have your very own C% compiler, which accepts '.cm' files. The actual compiler will be stored in a bin folder, and the usage is as follows:

```
$ ./bin/cmc [-h help] [-t token] [-a ast] [-l llvm] [-c ll-file] [-s s-file] [-e exe-file] <file-name>.cm
```

By default, the compiler will generate the `.ll` file corresponding to the program. However, the optional flags allow the user access to each step in the compilation process.

- **-h** - help - This option simply prints the help page and usage for the compiler.
- **-t** - token - This option prints the tokenized program to stdout.
- **-a** - ast - This option prints the abstract syntax tree of the program to stdout.
- **-l** - llvm - Compiles <file-name>.cm to llvm and prints the result to stdout.
- **-c** - ll-file - Compiles <file-name>.cm to llvm and puts the result in <file-name>.ll. This is the default option.
- **-s** - assembly - Compiles <file-name>.cm to llvm, translates to assembly, and puts the result in <file-name>.s (leaves <file-name>.ll in directory as well)
- **-e** - executable - Creates the executable version of <file-name>.cm, simply called <file-name> to be run ./<file-name> (leaves behind the corresponding .ll and .s files as well)

These options allow a variety of choices for the user. Basic compiling to LLVM can be done as follows, with the example program *helloworld.cm*:

```
$ ./bin/cmc helloworld.cm
```

2.3 Building A Basic Program in C%

2.3.1 File Extension

A C% program can be as short as three lines, to millions of lines long. It should be written into text files with the file extension ".cm"; As an example, the first program that will be shown is written in a text filed named `helloworld.cm`.

2.3.2 Parts of a C% Program

A C% Program consists of the following parts:

- Function and Variable Declarations
- Statements and Expressions
- Comments

This section will only briefly go over these parts. More in-depth documentation can be found in the Language Reference Manual (LRM) later in this report.

2.3.3 Hello World

In this subsection, we will go over a simple program that prints "Hello World!". The following is the source code for `helloworld.cm`:

```
int main() {
    printf("%s\n", "Hello World!");
    return 0;
}
```

As per the LRM's section 8 on Program Structure, each program requires a `main` function. C% is modeled after C, and thus the main function should always return 0. The line `int main()` is where program execution begins. The next line `printf(...)` uses a function available in `C%` that outputs "Hello World!" to the `stdout`. The next line `return 0;` defines what the `main()` function returns, before the function (and in this case program) terminates with the closing curly bracket.

2.3.4 Variables

In C%, variables are declared in the format of `type variableName`. Variables can be declared anywhere in a block, but unlike C, variables must be declared in a line separate from their assignment. For instance, the line of code `int c = 0;` will produce an error. A basic variable declaration can be seen below:

```
int c;
c = 0;
```

Generally, declared variables are local to the block in which they are declared. For more details on scoping, please see the section on Scope in the LRM (Section **3.7.2**).

2.3.5 Functions

Functions are declared before the main function. In C%, they have the following format:

```
returnType functionName(argsType1 args1, argsType2 args2, ...){
        declarations, statements, expressions
    return variableOfReturnType;
}
int main(){...}
```

This function declaration format is standard to C, and provides all of the functionality a user would expect: encapsulating chunks of code for sharing and reuse, with all the relevant information stored in the function signature.

2.3.6 Style and Organization

The style recommended for writing C% programs is not much different from that of writing C programs. Variable and function names should use `camelCase`. As per Wikipedia's definition, it is "is the practice of writing compound words or phrases such that each word or abbreviation in the middle of the phrase begins with a capital letter, with no intervening spaces or punctuation." In the case of C% and other languages utilizing `camelCase`, the first letter of the first word is not capitalized. Additionally, Lines of code should also not exceed 80 characters.

2.3.7 Using Pointers

Pointers are denoted in C% with the `*` symbol. Pointers are simply references to a specific memory location, and can be assigned to reference different values of the same type. Pointers are useful in the sense that they can be passed into functions, so that the entire memory block that holds an object doesn't need to be duplicated and then passed into a function.

Pointers can be dereferenced with "`*`" to modify the actual objects found at the memory pointed at. We can see this in the following example:

```
int x;
int *pointerToInt;
x = *pointerToInt;   // assigns x = 4
```

2.3.8 Malloc and Free

C% is a language that uses the `malloc` function for memory allocation. `malloc` is a function that allocates the specified amount of space on the heap and returns a pointer to the relevant address. This memory can then be freed by the `free` function. For example:

```
int *p;
p = malloc(4*2);
*p = 1;
*(p+1) = 2;
free(p);
```

In this example, we allocate 4*2 bytes of memory for two `int`s, which are 4 bytes each, hence the 4 and the 2. This is assigned to a pointer to an int `p`. It is then dereferenced, and then assigned values of 1 at the location `p` is pointing at, and then the memory location that is one over from `p` is given the value of 2. At the end of it all, the memory is freed. This memory management system is very similar to that of C.

2.3.9 Utilizing C%'s Built in Types

Most unique to C% are its built-in types. This section will describe how to use them and their various use cases. Basic types that aren't specific to C% can be viewed in the LRM.

Stones

Stones are C%'s implementation of large `int`s. Cryptographic applications commonly call for very big prime numbers which would exceed the maximum `int` capacity. They are mutable, such that they can be altered, reassigned, and do not have static values. All of the operators that work with `int`s work with Stones. Stones, however, do not have the capability of interacting with `int`s in the same way. Stones are declared with quotes, as shown below:

```
stone a;
stone = "99999999999999999999999999999999999999999999";
```

In general, Stones are the backbone of C%. They should be used whenever one is working with large numbers, or potentially large numbers, and form the basis of all other cryptographic types.

Mints

A `mint` is short for ModInt. It is a data type that holds its current value, and an immutable modulus, which are both objects of the `stone` type. They are declared using <,> brackets, as seen here:

```
stone val;
val = "10000";
stone mod;
mod = "9";
mint m;
m = <val,mod>;
```

Mints also have their own arithmetic. All the arithmetic that is applied to `mint`s can be seen in the LRM, and these include all the basic arithmetic functions that can be applied to `int`s and `stone`s. Furthermore, the cost of modular arithmetic is much lower than regular arithmetic using stones, and then applying the moduli afterwards. In particular, our algorithm for modular exponentiation is much less costly than a naive one that simply exponentiates the value and reduces it (this will be explained later in the report).

Curves

The `curve` data type holds the information used to describe an elliptic curve. Elliptic curves are becoming increasingly important as elliptic curve cryptography continues to gain popularity. Along with points, they can be used in encryption applications like the Diffie-Hellman key exchange or El-Gamal encryptions just as modular integers can. They are declared with two `mint`s that share the same moduli:

```
stone a;
stone b;
stone p;
p = "61";
a = "7";
b = "26";
```

```
mint A;
mint B;
A = <a, p>;
B = <b, p>;

curve *E;
E = <A,B>;
```

Points

The point data type is a data type that represents points on an elliptic curve. They have their own arithmetic with respect to the curve on which they were declared, and `point` arithmetic is prominent in all elliptic curve-based encryption algorithms. They are constructed with the `<curve E, stone x, stone y>` signature. Continuing the example from the Curve section above:

```
point *P;
a = "25";
b = "37";
P = <E, a, b>;
```

Note that arithmetic is done on pointers to points. This is so our helper functions can pass the points by reference.

Of course, the user needs to know whether or not this is actually a point on the curve – in all reasonable applications, the user will know since a responsible cryptographer will want to use carefully picked points on the curve which have good properties (e.g. high order in the group).

2.3.10 Code Examples

RC4 Stream Cipher in C%

```
// Key Scheduling Algorithm
// Input: state - the state used to generate the keystream
//        key - Key to use to initialize the state
//        len - length of key in bytes
void ksa(char *state, char *key, int len)
{
  mint j;
  int t;
  int i;

  for (i=0; *i < 256; i = i+1){
    *(state+i) = i;
  }
  j = 0;
  for (i=0; i < 256; i = i+1) {
    mint temp;
    temp = <i,len>;
    int key_index;
    key_index = access(temp,0);
    j = <(j + *(state+i) + *(key+key_index),256>;
```

```
    t = *(state+i);
    *(state+i) = *(state+j);
    *(state+access(j,0)) = t;
  }
}

// Pseudo-Random Generator Algorithm
// Input: state - the state used to generate the keystream
//        out - Must be of at least "len" length
//        len - number of bytes to generate
void prga(char *state, char *out, int len)
{
   mint i;
   int j;
   int x;
   int t;

   i = "0";
   j = 0;
   char key;

   for (x=0; x < len; ++x)  {
      i = <(i + 1, 256>;
      j = <(j + *(state+i)), 256>;
      t = *(state+i);
      *(state+i) = *(state+j);
      *(state+j) = t;
      mint out_temp;
      out_temp = <(*(state + *(state+i) + *(state+j))),256>;
      *(out+x) = access(out_temp,0);
   }
}
int main(){

  char *state;
  char *key;
  key = malloc(20);

  printf("Enter key: ");
  scanf(key);

  int len;
  len = 18;

  ksa(state,key,len);

  char *out;
  out = malloc(20);

  prga(state,out,len);
```

```
  printf("%s\n",out);
  return 0;
}
```

Elgamal Encryption in C%

```
int main() {
    stone g_div;
    stone h_div;
    stone p;
    mint g;
    mint h;

    //public keys
    p = "977";
    g_div = "3";
    h_div = "249";  //alice's g^x for her secret x
    g = <g_div, p>;
    h = <h_div, p>;

    //bob's private key
    stone y;
    y = "77";

    //shared secret g^xy
    mint s;
    s = h^y;

    char *x;
    x = malloc(100);
    int msg_len;
    int i;
    scanf(x);
    msg_len = atoi(x);
    printf("%s\n", x);
    for (i = 0; i < msg_len; i = i + 1) {
        scanf(x);
        stone z_div;
        mint z;
        z_div = x;
        z = <z_div, p>;

        print_div(g^y);
        print_div(z * s);
    }
}
```

Chapter 3

Language Reference Manual

3.1 Introduction

C% is a C like language purposed for applications of cryptography.The issue in the current state of the art cryptography is not in the theory of Diffie Hellman, but rather in the cryptology of how the protocol is implemented in specific applications and contexts. With this in mind, C% provides built in types, operators and functionality that are designed to remove the burden of carrying around large primes and keeping care over many modular calculations from the programmer.

The aforementioned functionality is built into a familiar and powerful C like language specified in this manual. Subsequently, in the specification of this language and development of this manual many ideas and notions are purposed from the C Reference Manual [1] as well as Kernighan and Ritchie's *The C Programming Language*[2].

3.2 Types

Each data type will be given a name and stored in a variable. These names are case-sensitive and made up of alphanumeric characters (Including '_'). Certain keywords used elsewhere in the language will be reserved and unable to be used as variable names ("int", "if", "for"). (Cite K&R)

3.2.1 Basic Data Types

The basic data types listed will be implemented similarly to C.

`int`

An Int is a 4 byte representation of an integer. An Int ranges in value from -2,147,483,648 to 2,147,483,647. When an Int overflows in either direction behavior is not defined.

Ints can be declared with or without an initial value. If no initial value is given it will have an undefined value until something is defined.

```
int x;
x = 5;
```

[1] https://www.bell-labs.com/usr/dmr/www/cman.pdf

[2] Kernighan, Brian W., and Dennis M. Ritchie. The C Programming Language. Vol. 2. Englewood Cliffs: Prentice-Hall, 1988.

`char`

A Char is a 1 byte representation of any ASCII character.

Chars are declared by giving a character surrounded by apostrophes. If no initial value is given it will have an undefined value until something is defined.

```
char x1 = 'c';
char x2 = '\n';
```

Note that our language has string literals which will allow character arrays to be used similarly to Strings.

`void`

Void is reserved as a type to allow functions to return nothing.

3.2.2 Cryptographic types

These cryptographic types represent algebraic structures that are useful for the types of programs C% is intended for. These types of programs often deal with number much larger than a reasonably sized Integer type could handle. Thus, the building block of all of the cryptographic types is a Stone, which will be implemented similarly to BigNum. All following types are declared with stones in order to facilitate calculations on unbounded numbers.

`stone`

A Stone is the building block of most of the other cryptographic types. Stones are represented as a linked list of integers and grow in size every time they run out of space to hold the current number. Thus, a Stone is effectively an unbounded integer. Stone's are used to represent the large numbers and primes that our programs are intended to work with.

A Stone is defined as simply as string literal or `char *` and many of the same operators will apply to it. The value must be assigned as such because arbitrary size integers are to wide to fit well into any sane bit width.

```
stone x = "5";
stone x = "999999999999999";
```

`mint`

A ModInt is a data type that holds two things; a current value and an immutable modulus. ModInt's are used to represent a number under a certain modulus to be used for all of its calculations. Because it only makes sense to perform operations on ModInt's with the same modulus at any given time, behavior is undefined if a user attempts to apply binary operations on two ModInt's with differing moduli.

A ModInt will only be able to be defined with two Stones to enable to work with unbounded integers. The value will be listed before the modulus in the declaration.

```
stone s1 = "3";
stone s2 = "7";
mint x = <s1, s2>;
```

`curve`

Curves represent elliptic curves. Mathematically, the data of an elliptic curve is just its two coefficients a and b (see Appendix). Thus, Curves are declared with two ModInts. There is not any defined behavior if the two ModInt's have different moduli.

```
mint x1;
x1 = <"5", "7">;
mint x2;
x2 = <"3", "7">;
curve *c;
c = <x1, x2>;
```

Note that the numbers in the ModInt declarations will be interpreted as Stones and not Ints

`point`

Finally, Points are a fundamental data type for our language. Operations on Points are defined through a given curve. Mathematically, the data of a point consists of its two coordinates and the curve that it is on. In our language the two coordinates are represented with Stone's and the Curve will be given in the declaration as well. Note that operations will only have a defined behavior on two Point's under the same Curve.

```
point *p;
p = <c, "5", "6">;
point *pInf;
pInf = <c, ~>;
```

It is certainly possible that a point to may take on the value of "infinity" (see Appendix). In this case the user can define a point with the $\sim$ keyword for infinity.

`access`

Many cryptographic types are made up of multiple stored values. In fact, all of C%'s cryptographic types can be reduced to a set of Stones. The set of access functions is comprised of `access_mint()`, `access_curve()`, and `access_point()`, which give the user a method to read these component values for each respective type, by index. The Stones are in the same order as the type declarations. The syntax is as follows:

```
stone a;
    stone b;
    mint m;

    a = "3";
    b = "5";
    m = <a, b>;

    stone m1;
    m1 = access_mint(m, 0);  //m1 is assigned "3"
```

The access functions for Mints, Curves, and Points, accept varying index ranges corresponding to the actual number of component parts. A Mint breaks down into the two Stones used to construct it, so `access_mint()` will return those with second parameter (index) 0 or 1, respectively. Similarly, `access_curve()` breaks a Curve down into two

Mints, and then two Stones each, allowing access via indices 0-3. `access_point()`, following the same logic, allows access to Stones at indices 0-5, the first four corresponding to the Curve broken down, and the last two corresponding to the 'coordinates' of the Point used in declaration. If the user enters an index value that is outside the range of component parts, the access functions return the last stone by default.

3.2.3 Grouping

Pointers and Arrays will both generally be handled as they are in C.

`pointer`

Pointers are stored with 8 bytes and represent a specific address in memory. Pointers can be incremented or decremented to look at adjacent blocks of memory. Pointers will specifically be useful in referencing Stone's which can be arbitrarily large.

Pointers are declared by naming the type of data to be pointed to and using the dereferencing (*) and referencing (&) unary operators.

```
int x = 7;
int* ptr = &x;

stone s = "17"
stone* sPtr = &s;
```

`array`

Arrays can be used to store multiple instances of the same data type. An Array of any type can be declared with any specific size. Use square brackets to declare.

```
int[] arr[10];
arr[0] = 2;
arr[1] = 4;
```

3.3 Lexical Conventions

Keywords

The following keywords are reserved in the language. They are enumerated here by type and are each discusses in later sections.

- Types
  - `int`
  - `char`
  - `stone`
  - `void`
  - `mint`
  - `curve`
  - `point`
- Statements
  - `if`
  - `else`
  - `for`
  - `return`
- Functions
  - `printf`
  - `scanf`
  - `malloc`
  - `free`
  - `etc.`

Comments

Single line comments are denoted by //. Any text on a line after a // will not be processed as part of the program.
Multiline comments are opened with `"/*"` and closed with `"*/"`.

3.4 Expressions

3.4.1 Primary Expressions

Primary expressions are in many cases the fundamental expression type that is operated on. This section enumerates the class of primary expressions.

Identifiers

Identifiers are typed expressions that have been declared. For example in

```
int a = 5;
int *b = &a;
```

a and *b* are identifiers with types int and int pointer respectively. Additional identifiers include functions. In this case, the expression's type is that which the function returns.

Literals

Literals come in the form of numeric constants or string literals. If a constant fits into the size of an int, it will be of type int. If not, it will automatically be of type stone, as will be the case if a large prime is hard-coded (not recommended).
String literals will be expressed by any characters set between two double quotes. They will be of type `char*`. They can be defined as in the example below.

```
char *x;
x = "Hello, World!"
```

All literals are strictly rvalues.

`Infinity`

$\sim$ is how infinity will be expressed in the language. It being mathematically valid for a point to have value of infinity per section 3.2.2, a point can be defined by `point p = {c,` $\sim$`}` where c is a Curve.

(*expression*)

A parenthesized expression. The expressions type and value is exactly that of the expression bound by the parentheses.

lvalue Expressions

There are exactly 3 forms of lvalues in this language. They are

1. `identifier` (non function pointers)
2. `( lvalue )`
3. `*expression`

All other expressions in this language are rvalues.

primary − expression (*comma separated expression arguments*)

This primary expression takes the form of a function call, with the specified *primary − expression* being a function as specified in 3.4.1. Functions can but do not need to accept a sequence of expressions, where each expressions type matches that specified in the function declaration. Each parameter/expression is to be separated from the previous by a "," (comma).

Also, C% is a *copy by value* language, just like C. Of course, one can still pass in a pointer to an object to simulate passing by reference.

Other Expression Types

Other expression types are built upon the Primary Expression types specified in the previous section. The remaining expression types are operations over the general expression types and are specified in section 3.5.

3.4.2 Order of Evaluation

There is exactly one operator on primary expressions, (). This operator has the highest priority in the language and groups left to right. For example, in

```
f(g(x, y))
```

function `f` receives the result of the evaluation of `g(x, y)` as its parameter.
The unary operators have the next highest precedence in the language and group right to left. Following, binary operators have the next highest priority, grouping left to right. The unary and binary operators have precedence following the order that they are presented in section 3.5, with the first operator presented having the highest precedence. The assignment operators have the next highest level of precedence, with equal precedence amongst all operators.

3.5 Operators

3.5.1 Unary operators

These operators work exactly as they do in C.

***** *expression*

The unary $*$ operator represents dereference: *expression* must be a pointer, and this returns the object to which the expression points. Indeed, if the expression is a pointer to a type T, the type of the result will be T.

& *lvalue − expression*

The result of the & operator is a pointer to the object referred to by the lvalue-expression. If the type of *lvalue − expression* is T, the type of the result of this operation is pointer to T.

- *expression*

The result of the unary $-$ operator is the negative of the expression. If *expression* is an integer type, this is the typical additive inverse. If *expression* is of type point, this returns the inverse of the point on the curve (as defined in **9.3.1**).

! *expression*

The unary ! operator is the logical negation, i.e. it takes an integer type (or a pointer type) and returns (as an int) 0 if *expression* is non-zero and 1 otherwise.

3.5.2 Exponential operator

expression ^ *expression*

The ^ operator represents exponentiation and requires an `int`, `stone` or `mint` on the left as a base and an `int` or `stone` as the exponent. Let the numerical value of the right expression be n (if it is of type modint, it returns the value, not the modulus). If n is non-negative, this finds the result of its value taken to the nth power (and takes the remainder if the left expression is a modint). Also, in the case that the left expression is a modint, this uses the squaring method for modular exponentiation to perform the operation efficiently.

If n is negative, this is only defined when the left expression is a modint with a multiplicative inverse, in which case this finds the inverse of the integer and takes that to the nth power (using the same algorithm as above).

3.5.3 Multiplicative operators

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

expression ***** *expression*

Depending on the type of the expressions, this operator has very different uses.

If the expressions are both of type modint, this returns the product of the two numbers under that modulus. The behavior is defined only when the two modints have the same modulus. If only one expression has type modint, it returns the product of two numbers under that modulus. Otherwise, it returns the typical numerical product of the two.

Now, if one expression has type point, then the other must be an integer type. In this case, if the integer type has value n, it returns the result of the point added to itself n times (see §9 for further explanation). No other type combinations are allowed.

expression / *expression*

The / operator only works when the expressions are of type int or stone. It implements integer division, returning the type of the first expression.

expression **%** *expression*

The % operator only works when the expressions are of type int or stone, and returns the remainder of the first expression when divided by the second, returning the type of the first expression.

3.5.4 Additive operators

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

expression + *expression*

The + operator returns the sum of the expressions. If the expressions are both integer or modint types, conditions are analogous to those applying to $*$.

If the expressions are both of type point, this computes the elliptic-curve addition of the two points (see §9), the behavior of which is only defined when the two points are relative to the same elliptic curve. No other type combinations are allowed.

expression **-** *expression*

The binary − operator returns the difference of the expressions. It calculates exactly *expression* + (−*expression*) and is valid for all types for which this expression is valid.

3.5.5 Relational operators

expression < *expression*

expression > *expression*

expression >= *expression*

expression <= *expression*

These expressions return 0 or 1 based on whether the specific relation is false or true, respectively. The operators are only defined when the expressions are of integer type or are a pointer, though both expressions need not be of the same type. In the case of a pointer, it uses the memory location that it is pointing to (interpreted as a number) to compare, and in the case of a modint it uses its numerical value (without the modulus).

Note that the statement $a < b < c$ does not seem to mean what it does, even though the operators do group left-to-right.

3.5.6 Equality operators

expression == *expression*

expression **!=** *expression*

These are analogous to the relational operators, except they have lower precedence. Note that this implies that they implement structural equality, not physical equality.

Note that if the two expressions are of type modint, it does not check that the moduli are equal. To do so, call `access()` (see **2.2.5**). Also, == is defined for points – it checks that the curve they are defined over is the same and that its x and y coordinates are equal as modints.

expression **&&** *expression*

This is the logical AND operator. It returns 1 if both values are non zero and 0 otherwise.

expression || *expression*

This is the logical OR operator. It returns 0 if both arguments are 0, 1 otherwise.

3.5.7 Assignment operator

lvalue = *expression*

At the end of the evaluation, the value of the right operand is stored in the left operand. The value of this operation is exactly the value of the left operand after the assignment has been completed. The operand on the left of the assignment operator must be an lvalue.

lvalue =% *expression*

At the end of the assignment, lvalue will store the result of *lvalue* % *expression*.

3.6 Statements

Statements allow the user to control when expressions are executed in their programs.

3.6.1 Statement Terminator & Blocks

An expression is turned into an expression statement when it is terminated by a semicolon:

expression ;

Any expression becomes an *expression statement* when terminated by a semicolon, however expression statements are only useful for their side effects, such as a calling functions or assigning a value to a variable.

Statements may be grouped together into a *compound statement* or *block* using braces { }. Blocks are used to group multiple statements together, and the resulting block is syntactically equivalent to a single statement. Blocks are commonly seen in the definition of a function body, or with many of the control flow statements defined in this section, such as for, while, and if. Unlike normal statements, blocks do not need to be terminated by a semicolon. Blocks may be defined within other blocks, and variables may be declared within any block. When a variable is declared within a block, its scope is defined by that block, meaning that the variable is local to the block in which it is defined.

3.6.2 Control flow

Control flow statements determine the order in which expressions are evaluated.

`if`, `else`

The keywords `if` and `else` are used to create conditional statements, which selectively execute statements based on the truth of a given expression. The simplest constructions are

`if` (*expression*) $statement_1$

`if` (*expression*) $statement_1$ `else` $statement_2$

In each of these expressions, $statement_1$ is executed if (and only if) *expression* evaluates to true (a non-zero value). If *expression* `== 0`, nothing happens as a result of the `if` statement unless there is an `else` following $statement_1$, in which case $statement_2$ is executed instead of $statement_1$. For example,

```
if (n > 3)
        x = 1;
else
        x = 5;
```

If `n > 3` evaluates to true, then the variable `x` is assigned the value 1, otherwise `x` is assigned the value 5. Note that the expression `n > 3` is not terminated by a semicolon, as it is not a *statement*, while both `x = 1` and `x = 5` are semicolon terminated, in accordance with the expected syntax of `if` statements listed above.

`if` statements can be nested and combined to test for multiple conditions:

```
if (n > 3)
        x = 1;
else if (n == 3)
        x = 3;
else
        x = 5;
```

In nested statements such as these, each `else` matches the closest preceding `else`-less `if`. This rule clarifies the ambiguity resulting from the fact that not every `if` must have an `else`. For example, in

```
if (n > 3)
        if (n < 10)
                x = 1;
        else
                x = 5;
```

`x` will be assigned the value 5 if `n` is greater than 3 but not less than 10, i.e. the `else` is matched with the most recent `if` which tests if `n < 10`. If you want to match the `else` with the first `if` instead, you must use braces:

```
if (n > 3) {
        if (n < 10)
                x = 1;
}
else
        x = 5;
```

Loops – `while`

The `while` statement is used to create a loop that runs so long as a given expression is true. The formal syntax is:

`while` (*expression*)
    statement

In a `while` statement, *expression* is evaluated first. If it evaluates to a non-zero value, *statement* is executed and then *expression* is re-evaluated. This cycle continues until *expression* evaluates to 0, or is false, at which point the `while` statement is over, and *statement* will not be executed anymore. A `while` statement may also be exited using a `break` statement, as discussed in **5.2.5**.

The following example computes the sum of the first 9 integers:

```
int sum = 0;
int n = 1;
while(n < 10) {
        sum = sum + n;
        n = n + 1;
}
```

Loops – `for`

The `for` statement has the syntax

`for` ($expr_1$; $expr_2$; $expr_3$)
statement

and is equivalent to a `while` statement of the form

$expr_1$;
`while` ($expr_2$) {
statement
$expr_3$;
}

except only in relation to `continue`, discussed in **5.2.6**.

The three elements of a `for` loop are any expressions, and any or all of these expressions may be omitted. Specifically, $expr_1$ is executed once at the beginning of the loop, as an initialization step; $expr_2$ is a test expression that is executed before each iteration of the loop, such that if $expr_2$ evaluates to false the loop exits; and $expr_3$ is called after each iteration of the loop body, as an incremental step. Omission of $expr_1$ or $expr_3$ simply means that the loop runs without calling those expressions. In the event that $expr_2$ is omitted, the loop runs as if $expr_2$ is true permanently, as in an 'infinite' loop (`while(1)`). This makes sense only if the loop is to be broken in some other way, such as with a `break` or `return` statement. It is also important to note that, in accordance with the behavior of the comma operator (see **5.9.1**), the use of a comma separated expression for $expr_2$ will have the effect of only using the rightmost expression as a check for termination of the loop. In order to check multiple conditions, you must combine the expressions using `&&`.

For example, the following loop will terminate when `x` reaches 10, not when `y` reaches 5:

```
int sum = 0;
for(int y = 0, x = 0; y < 5, x < 10; y = y + 1, x = x + 1)
        sum = sum + x + y;
```

To actually check both conditions, replace $expr_2$ with `y < 5 && x < 10`.

The `for` loop makes more sense than the `while` loop in cases where there are present and simple initialization and incremental steps, as when you want to count the number of times the loop runs.

Our previous example is one of those cases:

```
int sum = 0;
for (int n = 1; n < 10; n = n + 1)
        sum = sum + n;
```

Null

The *null statement* is a single semicolon.

;

A null statement does nothing, and has no impact on your program, but is useful in the body of a loop statement. For example, the following snippet sets `x` to the largest integer less than the square root of `a`:

```
int x;
for (x = 2; x*x < a; x = x + 1)
        ;
```

return

The `return` statement is used to end the execution of the surrounding function body and return control back to the calling function. The statement has the form

`return` *value* ;

where *value* is optional, and only valid if the function has a return type other than `void`. If the function has a return type other than `void`, the function return type and the type of *value* must match. In this case, it is also valid to omit *value* so long as the calling function does not require a return value. It is generally not a good idea to omit *value* unless the function return type is `void`.

3.7 Program Structure

A Program is composed of variables and functions. Information can be shared between functions by parameters, return values and global variables. The functions can occur in any order in the source file, so long as a function's prototype or definition is provided before or at the time of its use.

Programs require a `main()` function, and all other functions used within `main()` must be declared and defined before it to be used in the main function.

```
int otherFunction()
{
    declarations and statements
}
int main()
{
    declarations and statements
    otherfunction()
    return 0
}
```

How functions interact with the declarations and statements of other functions will be discussed in the section 3.7.2.

3.7.1 Functions

Basic Functions

Functions have the following form:

```
return-type function-name ( argument declarations )
{
    declarations and statements
}
```

Functions need a `return` statement to return a value from the called function to its caller, with the exception of functions that have `void` as the return-type (refer to section 2.1.3). If a return type is not explicit, then it will be assumed to be an `int` [**?**].

Returning Expressions

Functions may return expressions as follows:

`return` *`expression`* `;`

The *`expression`* will be converted to the function's return type if they do not match.

Void Functions

Functions may return what appears to be the return-type `void` like in the following:

```
void function-name ( argument declarations )
{
    declarations and statements
}
```

In `C%`, a `void` function type indicates that the function does not return a value.

3.7.2 Scope

Lexical Scope

The lexical scope of names declared in external definitions (in a file, but not within a function or other specified scope) extends from their definitions to the end of their respective file. Within functions, the lexical scope of names is the body of the function itself [**?**].

```
main() { int a; ...}
//a falls out of scope

int mainCantSeeThis = 0;

void mainCantSeeThisEither(int a) {...}
```

In the above example, `main()` cannot access the int declared after it, nor can it access the `void mainCantSeeThisEither`. In the same example, `void mainCantSeeThisEither` can access the declared `int mainCantSeeThis`. Furthermore, a is not accessible outside of the scope of its definition.
It is illegal to redeclare a name within its context unless it is of the same type and class it previously held [**?**].

3.8 File I/O

3.8.1 I/O Channels

C% supports three I/O channels: standard input (stdin), standard output (stdout), and standard error (stderr). They are represented by the ints that describe their system file descriptors in C%'s standard I/O functions. stdin is represented by 0, stdout is represented by 1, and stderr is represented by 2.

3.8.2 printf()

printf directly outputs the string associated with a char pointer to stdout.

```
int  printf(const char *format, ...);
```

It is also worth noting that our language has multiple special printing functions for displaying the data in our custom types. This functions are mostly used for debugging and have names following the format of print_stone.

3.8.3 scanf()

scanf directly reads from the stdin.

```
int scanf(const char *format, ...);
```

The scanf function allows us to read input from stdin and store it in a character pointer The following code reads in a string from stdin and prints it to stdout. This use case demonstrates the basic use of our File I/O.

```
int main() {
    char *a;

    a = malloc(10);
    scanf(a);
    printf("%s\n", a);

    return 0;
}
```

Chapter 4

Project Plan

4.1 Planning Process

We tried to break up our project into discrete chunks as much as possible. In the beginning, this was more difficult while we all worked on the same files from MicroC in order to implement our own grammar. However, as the project went on we were each able to find our niche where we could contribute the most efficiently. A lot of our deliverables were self driven with everyone contributing where they felt that they would be the most helpful. This worked out well, especially because our manager, Josh, and TA, Jacob, both helped to make sure that we were always working toward productive goals. This freedom in deliverables was made possible by weekly meetings in order to keep our entire project on track.

We set up issues to be picked up on github in order to track who would tackle each problem.

4.2 Specification Process

We had a very specific goal when we began the development of our project. We wanted to make a language that would make cryptologist's jobs as easy as possible. This lead us to create an initial list of features that would make processes like encryption and secure handshakes as intuitive as possible. We created an initial list of features that would all be useful for cryptography and modified them as we developed and realized more use cases and limitations.

4.3 Development Process

Our development was structured in order to meet goals given by our manager and TA and to meet the deadlines of the course. After the initial stages of development that required getting our scanner and parser working. After that our development became feature driven with everyone splitting up to implement separate features into the language.

4.4 Testing Process

Our testing process is explained in depth in its own chapter of this report. We made sure to include tests for each feature that we added to make sure that their functionality was maintained with every update. We also added tests at the beginning of each branch to make sure that the added functionality did indeed take something that did not previously work and make it work. No PR was ever merged unless it completed our entire test suite.

4.5 Team Responsibilities

We did not strictly adhere to our initial roles. Here is a quick summary of individual responsibilities.

- Margaret Mallernee - I worked mainly on aspects of codegen, unary operators, the access functions, and documentation. I also got our logo and ordered shirts.
- Zachary Silber - Test Suite. Continuous Integration. Merge Quality Assurance. Link to C functions. Documentation.
- Michael Tong - Semantic Checker. Point and stone arithmetic implementation. Variable declaration flexibility. Preprocessor. String and char literals. Line comments.
- Richard Zhang - Helped out on testing throughout the project and wrote a majority of the standard library.
- Joshua Zweig - I worked largely on the integration openssl/bn with codegen, including problems of architecture, memory management and translation. I also worked broadly on many other aspects of codegeneration. As manager, I also took on the resoponsibilities of setting meeting times, agendas and individual tasks for our weekly sprints.

4.6 Github Stats

Our members contributed with the following usernames.

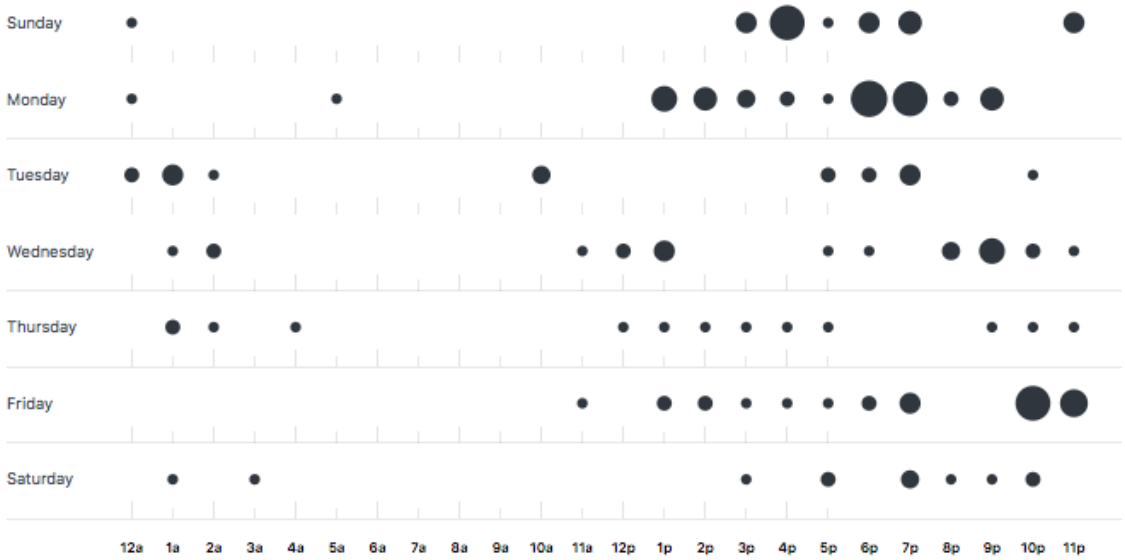
- Margaret Mallernee - mlm2299
- Zachary Silber - zsilber
- Michael Tong - mikecmtong
- Richard Zhang - richurdzhang
- Joshua Zweig - joshuazweig

We can see some of the contributions of each contributor by utilizing a small script.

```
cat ~/gitauth.sh
git log --author="$1" --shortstat $BRANCH | awk '/^ [0-9]/ { f += $1; i += $4; d
    += $6 } END { printf("%d files changed, %d insertions(+), %d
    deletions(-)\n", f, i, d) }'
```

```
$ sh ~/gitauth.sh Joshua Zweig
323 files changed, 77299 insertions(+), 717 deletions(-)
$ sh ~/gitauth.sh Zack Silber
848 files changed, 12295 insertions(+), 4606 deletions(-)
$ sh ~/gitauth.sh Michael Tong
191 files changed, 75866 insertions(+), 75265 deletions(-)
$ sh ~/gitauth.sh Maggie Mallernee
23 files changed, 844 insertions(+), 268 deletions(-)
$ sh ~/gitauth.sh Richard Zhang
73 files changed, 1060 insertions(+), 91 deletions(-)
```

Additionally, we can utilize charts given by github to see that our efforts were focused around the afternoon and that (thankfully) no commits were ever pushed between 6am and 9am.



4.7 Project Log

Here is a dump of all of the commits for our project. Squashing was utilized on a few big PR's. We ran the command 'git log –no-pager' to get this log.

4.8 Development Environments

We tried to keep software consistent among our team as much as possible and necessary.

- Github - We used git for version control which made collaborating on the same files and keeping track of changes much more convenient. We made sure to always have a functioning master branch.
- OSX - We all ran OSX which helped keep everything consistent with the Unix development environment.
- LLVM 3.8 - This seemed to be a widely supported version and was easy to install and utilize.
- TravisCI - This third party software was used to host our continuous integration.
- Slack - We used this for official discussion of the project, while occasionally defaulting to Facebook Messenger for more brief communication.
- Overleaf/Latex - We used Latex to produce all of our documentation and collaborated on the same document via Overleaf.
- Vim/etc. - We mostly used vim since it is so well integrated with the OSX terminal, however there were times when other editors were used and it was always left up to the individual developer.

4.9 Style Guide

While, admittedly, there were no concrete rules put into place during development, we tried to stick to a few consistent principles when writing OCaml and C% code.

- No lines should be longer than 80 characters
- Points, Curves should have capital variable names. If possible, the name of a Curve should be taken from the beginning of the alphabet (e.g. E or C) while points should be taken from the latter third (e.g. P, Q, R). In the case that one wants to make the variable name more descriptive, descriptors should be added on with underscores (e.g. E_public or P_alice)
- When a mint is to be a coefficient for the initialization of a Curve, it should likewise have a capital variable name. In such a case, if possible, the first mint should be called A and the second mint B (as above, one may add other descriptors using _, e.g. A_public).
- Otherwise, variable names should be in camelCase.
- Tabs should be consistent and 4 spaces each
- Curly braces should start on the same line as the code before

```
commit 58ea88bc1a81da22bc0855d5aacc56a99c82bc84
Author: mlm2299 <mlm2299@columbia.edu>
Date:  Tue May 9 22:45:37 2017 -0400

    Mm access (#51)

    Mainly: Adding access functionality for mints/curves/points, fixing some
        unops, miscellaneous other

    * filled in pattern matching at line 27 for ltype_of_type, will fill our
        unique types with C structs linked in

    * Added skeleton for all missing pattern match options in the expr builder
        function

    * Added skeleton for stmt builder, should complete all pattern match errors

    * Starting codegen on dowhile

    * Brought over changes from mm_codegen

    * Added some of the type stuff and prepped for linking in access

    * Added tested access.c and types.h

    * Updated Makefile to include access, codegen ready to test access,
        special_arith now links in types instead of declaring them separately

    * Added type matching for unops, to return proper type

    * Added test for unop NOT for int types

    * ***previously added test was actually for NEG on int types
```

```
* Added test for unop NOT on int types -- currently fails

* finalized unop NOT test for ints -- revealed implementation is wrong

* Fixed NOT for int types, modifying from the microC version

* Left a note on what's left to test

* Added function for point inversion

* Updated codegen for point inversion (Neg on Point types)

* Added instruction for AddrOf

* Small changes to access

* Added Makefile changes to include access / correct a typo

* Added access.c

* Added types.h

* Updated access to allow index access to mints

* Updated access for index access to curves

* Added index access to points, got rid of testing print statements

* Commented out untested unops, updated access to take an index in codegen

* Changed back to access - need to merge up

* Updated to conform w new styles from master / stone as void *

* Updated access mint

* Updated to be access1, updated and tested access1 with BN values

* Freed BNs in access.c

* Debug test for multi-parameter syntax in semant/codegen for access

* Access working for mints

* Updated to have separate access functions for mints, curves, points

* Updated to include access_curve, but takes curve ptr because curve
    constructor returns this for some reason

* Added access_point, also takes a point_ptr even though those can't be
    declared

* Cleaned up codegen comments, updated compile/test scripts to link access.o

* Tests for access on mints, curves, points

* Updated compiler script to actually default to .ll file production
```

```
    * Cleaning up comments in codegen, commenting access

    * Commented out point_inverse_func - needs a fix

    * Misc. cleaning

    * Final cleanup

commit a6ebdc5b988191ae40e77625b88b667d4a38bba4
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Tue May 9 22:29:00 2017 -0400

    Jz stone compar (#52)

    * Comparators working

    * add test

commit 1eed8b78112b051048784119dc80f2c54b75c2fd
Merge: e5f2ee3 c5abb98
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Mon May 8 20:34:02 2017 -0400

    Merge pull request #49 from joshuazweig/rz_test_clean

    Broke down large tests into separate tests

commit e5f2ee38b77d4f1da2802add72eb0bdedc522508
Merge: 9a3fdb4 6af8125
Author: mikecmtong <mct2159@columbia.edu>
Date:  Mon May 8 20:24:28 2017 -0400

    Merge pull request #50 from joshuazweig/cleanup

    cleanup dir

commit c5abb9821a46d7c5f09a30e869f12b9b229fd0e6
Author: Zack Silber <zs2266@columbia.edu>
Date:  Mon May 8 14:25:24 2017 -0400

    Fixing up some testing problems

commit 6af8125b287f03c024c30f344369264df57ae1b2
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Mon May 8 12:22:06 2017 -0400

    cleanup dir

commit 9a3fdb461f897670e24d844548adf617bea16a71
Merge: fe0b7b8 26b9e79
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Mon May 8 11:49:01 2017 -0400

    Merge pull request #48 from joshuazweig/mt_elgamal_ec

    ElGamal normal
```

```
commit 26b9e7995eed6a1b64fa4ebb3d59e747998705b4
Author: Michael Tong <mct2159@columbia.edu>
Date:  Mon May 8 11:34:39 2017 -0400

    updated how to use, changed file extensions

commit 6b18e0f8fe8cc9cf32b2aef3c9298d0a794c0b06
Author: Richard Zhang <richard.zhang@columbia.edu>
Date:  Mon May 8 00:30:48 2017 -0400

    Seperated large tests into different tests

commit d81b8dfb05fd4f3c724e60b724c65b931197ecf0
Author: Michael Tong <mct2159@columbia.edu>
Date:  Sun May 7 14:46:43 2017 -0400

    added to instructions

commit 46333727f8d732aa97627ae0fba1c74f6a8255b6
Author: Michael Tong <mct2159@columbia.edu>
Date:  Sun May 7 12:54:50 2017 -0400

    named it for the right protocol this time

commit d5617320bceecb358f41165a4d83488110de9db9
Author: Michael Tong <mct2159@columbia.edu>
Date:  Sun May 7 12:52:57 2017 -0400

    added the files to a folder

commit fe0b7b803519cf5ee59074a2e89b3c46184c7125
Merge: 9644e9d b4d1721
Author: mikecmtong <mct2159@columbia.edu>
Date:  Sun May 7 12:49:31 2017 -0400

    Merge pull request #47 from joshuazweig/mt_diffie_ec

    Mt diffie ec

commit d79c515625e62dba5f76c0f07d50b48eef1e482b
Author: Michael Tong <mct2159@columbia.edu>
Date:  Sun May 7 12:45:34 2017 -0400

    naive RSA thing working

commit 5fbc8c1a3c1e102bae2329c9ea95574c29b95f89
Author: Michael Tong <mct2159@columbia.edu>
Date:  Sun May 7 12:42:31 2017 -0400

    added atoi and changed some rsa files

commit 38be86789a2b9a96af5ff559d0640188a6c22c4e
Author: Michael Tong <mct2159@columbia.edu>
Date:  Sun May 7 12:21:59 2017 -0400

    rsa stuff

commit b4d1721f2bd8d08999801a308826d1fb37f8de3a
```

```
Author: Michael Tong <mct2159@columbia.edu>
Date:   Sat May 6 23:05:33 2017 -0400

    added new print functions for communication

commit 94c2cc648b89061bd866ecfd1eb78353bc64840f
Author: Michael Tong <mct2159@columbia.edu>
Date:   Sat May 6 22:53:41 2017 -0400

    arithmetic working, though can't talk to each other because of how points
        are printed

commit 9644e9de37229ec9d90a0877e4e15eaf9a570fbf
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Sat May 6 13:18:12 2017 -0400

    This doesn't need to be here

commit ecd3853e745fb8441761affa48c73f74194bc5d5
Merge: 5871e98 65fbc7f
Author: mlm2299 <mlm2299@columbia.edu>
Date:   Sat May 6 13:18:06 2017 -0400

    Merge pull request #46 from joshuazweig/mm_compiler_flags

    Mm compiler flags

commit 65fbc7f1d05a96946cc19da89bef081b4f6dac8a
Author: Maggie <mlm2299@columbia.edu>
Date:   Sat May 6 13:10:36 2017 -0400

    bin directory

commit 420cf11d2536f55159218ada1aa0d308f17120fa
Author: Maggie <mlm2299@columbia.edu>
Date:   Sat May 6 12:57:48 2017 -0400

    Cut bin directory

commit 487e360c2f9ea56f27d61d501b5b4205396571a7
Author: Maggie <mlm2299@columbia.edu>
Date:   Sat May 6 12:51:59 2017 -0400

    Cut tabbing in help message

commit 34fad6cbc230fd211dff32308ae4b5bfc6b78654
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Sat May 6 12:43:11 2017 -0400

    Force remove bin dir

commit 258b70c6530e79666d7e13aae2483fa9d79ee0f2
Merge: 223c303 f6590e8
Author: Maggie <mlm2299@columbia.edu>
Date:   Sat May 6 12:38:46 2017 -0400

    Merged conflict
```

```
commit 223c303370be74f5e1c9c80210a0c709ed444ffe
Author: Maggie <mlm2299@columbia.edu>
Date:  Sat May 6 12:33:08 2017 -0400

    Makefile fixes, to create bin

commit f6590e89d736323f9c958dea524db6dc8c6d2e0e
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Sat May 6 12:32:28 2017 -0400

    Update makefile so compiler palce in bin dir

commit ccf1369de19071cc37b27ff7208cb4a91b7555dc
Author: Maggie <mlm2299@columbia.edu>
Date:  Sat May 6 11:47:15 2017 -0400

    Modified .sh added

commit a457fa4f2a766533bd7845dc8e280b07e11fb9c9
Author: Maggie <mlm2299@columbia.edu>
Date:  Sat May 6 11:46:27 2017 -0400

    Modified to handle permissions properly

commit dd49da96f6a8b18b333fc6e5650840a3b453d7e8
Author: Maggie <mlm2299@columbia.edu>
Date:  Sat May 6 11:38:26 2017 -0400

    New cmc.sh script

commit 85134a54c1d34baa004f39c620c69a7f9b4f5d8e
Author: Maggie <mlm2299@columbia.edu>
Date:  Sat May 6 11:37:48 2017 -0400

    Updated cmc to be a shell script, added target in Makefile that handles
        permissions

commit 2e4bc9ce3c23fbd227cdf7a34baccd22cbb0af23
Author: Maggie <mlm2299@columbia.edu>
Date:  Fri May 5 18:15:34 2017 -0400

    Removed extra comments

commit 612ef48ec0caa0decd58444d83494f64c4d64956
Author: Maggie <mlm2299@columbia.edu>
Date:  Fri May 5 18:13:49 2017 -0400

    Added -c as default option

commit 8886dd9523913afc529c2bc6712905536d1b4fab
Author: Maggie <mlm2299@columbia.edu>
Date:  Fri May 5 18:02:04 2017 -0400

    Updated help description

commit a07b5e70bc17f91527ebc3d9f13b454ff502a0e0
Author: Maggie <mlm2299@columbia.edu>
Date:  Fri May 5 17:55:51 2017 -0400
```

```
    Added token flag, updated ast flag to use menhir

commit 5871e987f0077840776748eab5f2d2cf8cee0604
Merge: 728bfa6 65fe75b
Author: mikecmtong <mct2159@columbia.edu>
Date:   Fri May 5 13:55:52 2017 -0400

    Merge pull request #45 from joshuazweig/mt_point_hacky

    Point multiplication working

commit 245289aecbb3ab77a6267a06310b44c6d98e2e88
Author: Maggie <mlm2299@columbia.edu>
Date:   Fri May 5 12:44:13 2017 -0400

    Small changes to help listing

commit bd6da7b662c94f9d2f914d84b574bfdfd7f06ae4
Author: Maggie <mlm2299@columbia.edu>
Date:   Fri May 5 02:50:04 2017 -0400

    Added more complete help message

commit 2b3922e5ec5c0833dce0cadcae031e7f467f05f1
Author: Maggie <mlm2299@columbia.edu>
Date:   Fri May 5 02:24:53 2017 -0400

    Added cmc to compile with options

commit 11f2cbca47cef26e355e5a5e5aae9b70954f1927
Author: Maggie <mlm2299@columbia.edu>
Date:   Thu May 4 22:14:52 2017 -0400

    Test change (added whitespace)

commit 65fe75b0eb83615270d15925108b5a802588cbc4
Author: Michael Tong <mct2159@columbia.edu>
Date:   Thu May 4 19:21:44 2017 -0400

    fixed syntax error

commit 76c331ccfb48be7ff44a5ba9193d5bb55c2ea4c5
Merge: 61d3146 728bfa6
Author: mikecmtong <mct2159@columbia.edu>
Date:   Thu May 4 19:14:47 2017 -0400

    Merge branch 'master' into mt_point_hacky

commit 61d3146afcaff663e1da5791cf61208c833d31d8
Author: Michael Tong <mct2159@columbia.edu>
Date:   Thu May 4 19:06:31 2017 -0400

    freed a helper string

commit 6177dfb89c0650db7b919fcba48460a5dcc2ff8d
Author: Michael Tong <mct2159@columbia.edu>
Date:   Thu May 4 19:03:01 2017 -0400
```

```
    version

commit c61db32b8c47df54348aacd68d0d239e7cd67bb5
Merge: 7579250 648b37b
Author: Michael Tong <mct2159@columbia.edu>
Date:   Thu May 4 19:01:42 2017 -0400

    fix merge conflicts

commit 75792505b483eb256bff17043aed67fe9834f421
Author: Michael Tong <mct2159@columbia.edu>
Date:   Thu May 4 19:00:45 2017 -0400

    Stone * Point multiplication working.

commit 728bfa6073aa7a6afdb7f2612d53cb66d470deec
Merge: ec915d5 e2773ca
Author: zsilber <zs2266@columbia.edu>
Date:   Wed May 3 22:36:00 2017 -0400

    Merge pull request #43 from joshuazweig/rz_src

    Added crypto_lib directory

commit e2773ca914b6ac67b6c95d1f138117821753c3cd
Author: Richard Zhang <richard.zhang@columbia.edu>
Date:   Wed May 3 22:23:15 2017 -0400

    Deleted old folder and keyex from home dir

commit 77a533f68731725fa3fed0f9c24435757129cfe3
Author: Richard Zhang <richard.zhang@columbia.edu>
Date:   Wed May 3 22:21:21 2017 -0400

    Created new folder for crypto_lib

commit ed067abe938bd2ded13030a8b7bd0483368cf05e
Author: Richard Zhang <richard.zhang@columbia.edu>
Date:   Wed May 3 22:11:42 2017 -0400

    Added crypto_lib directory

commit ec915d56f3882b82a8716c46bfe49670a77a7373
Merge: 12ada73 9c23bfb
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Tue May 2 02:19:20 2017 -0400

    Merge pull request #42 from joshuazweig/jz-free

    Jz free

commit 9c23bfbd11484ca19f8cfa1823b4fcd84b994fd1
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Tue May 2 02:19:02 2017 -0400

    remove freetst
```

```
commit 648b37b359858628a03f0d47fdbaa99d8605ece3
Author: Zack Silber <zs2266@columbia.edu>
Date:   Tue May 2 01:59:25 2017 -0400

    Fix one test and refix 3.7 nonsense

commit d19c8324bf0d192d843a664aec1a0f4ded33536e
Merge: eab3716 be7c954
Author: Michael Tong <mct2159@columbia.edu>
Date:   Tue May 2 01:55:43 2017 -0400

    Merge branch 'mt_point_hacky' of https://github.com/joshuazweig/C- into
        mt_point_hacky

commit eab371672560369022ec2a82735b68aa1de6955a
Author: Michael Tong <mct2159@columbia.edu>
Date:   Tue May 2 01:55:37 2017 -0400

    update fail message

commit be7c954802576b29d4a3f17a9e8e6c15b73282e8
Author: Zack Silber <zs2266@columbia.edu>
Date:   Tue May 2 01:53:50 2017 -0400

    Mint print test fixes

commit 7b68c88d1831893567f16f16ab3f6ef27612ac3b
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Tue May 2 01:47:50 2017 -0400

    Remove test prints

commit e41cb38a14ed76bb99b42d7c474d0d6851f3855f
Merge: d435ac3 12ada73
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Tue May 2 01:41:48 2017 -0400

    Merge branch 'master' into jz-free

commit d435ac30635f757afaf489bd92edcc7a98326b02
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Tue May 2 01:41:23 2017 -0400

    finish stone memory management

commit dcd8ad9f860f91a32e9e5afc04a2fe1236c43aab
Author: Michael Tong <mct2159@columbia.edu>
Date:   Tue May 2 01:36:13 2017 -0400

    fixed syntax error

commit 6c411654225fab42418e857a59290e8d016a8650
Author: Michael Tong <mct2159@columbia.edu>
Date:   Tue May 2 01:29:55 2017 -0400

    Fixed some stuff in semant/codegen (Point to Pointer(Point))

    The case of adding additive inverses is slightly buggy with point. Can't
```

```
    figure it out but I say fuck it, let's just try and get diffie hellman
    with ECC working and call it a day.

commit b7d83cabfa7e0c0193cae010b48fb0d09114b992
Author: Michael Tong <mct2159@columbia.edu>
Date:   Tue May 2 01:09:11 2017 -0400

    added a test

commit 1ae046bbb897c82040b90cf4832e37a8fb3eea19
Author: Michael Tong <mct2159@columbia.edu>
Date:   Tue May 2 01:01:35 2017 -0400

    Points working

commit 7dc976c56a8e22856cf8889e5585d0e18344dc33
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon May 1 21:43:10 2017 -0400

    modulation not working

commit 13b137f983c10ed96c79c444bf10c015b6daa0fc
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon May 1 21:38:01 2017 -0400

    Freeing on stone add

commit b0fca5b2cbdd5976e2b31111247cc52236fc858b
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon May 1 21:31:19 2017 -0400

    Mem management is a go :)

commit 0492ddbf01f29f08bea74b7dae5610da8e0e458c
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon May 1 17:51:40 2017 -0400

    Success :)

commit ec7211a671cfc155c9498d62e118dc28a1607e1f
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon May 1 13:10:04 2017 -0400

    commit for michael

commit 12ada7347f48a7ac4147842176289c1b60bd2bf4
Merge: 3d7e887 2d82ab4
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon May 1 00:24:56 2017 -0400

    Merge pull request #41 from joshuazweig/printf_param

    Printf param

commit 2d82ab4e02c7fa70eaef87fe91b32260605406b9
Author: Michael Tong <mct2159@columbia.edu>
Date:   Sun Apr 30 23:36:19 2017 -0400
```

```
    added a test

commit 9f66c4bc25bbb7363f50b008f7316ea01c299ddf
Author: Michael Tong <mct2159@columbia.edu>
Date:  Sun Apr 30 23:33:59 2017 -0400

    semant now checks arguments of printf (just that they are valid expressions,
        not that the types match up)

commit 3d7e88755b85b5df52d6ac1fa4cb3bc5fd3dbcca
Merge: f14bb19 979fc4e
Author: zsilber <zs2266@columbia.edu>
Date:  Sun Apr 30 17:38:52 2017 -0400

    Merge pull request #40 from joshuazweig/mt_nested_scope

    Can now declare variables of the same name within nested scopes

commit 979fc4ebea44a81bb8bc41d22c9533c64071a495
Author: Michael Tong <mct2159@columbia.edu>
Date:  Sun Apr 30 16:29:54 2017 -0400

    tests

commit e26efeaa87e707ac2c87b9ccf302f0d65cbf2a73
Author: Michael Tong <mct2159@columbia.edu>
Date:  Sun Apr 30 16:26:42 2017 -0400

    added VER variable

commit 060f897424e25de9fb5674707097618c06b8fec6
Author: Michael Tong <mct2159@columbia.edu>
Date:  Sun Apr 30 16:19:45 2017 -0400

    Can now declare variables of the same name within nested scopes

commit f14bb19e70b9f3de5ce58409cd1e2f709b258f0f
Merge: 95428d4 7b292e4
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Fri Apr 28 18:14:42 2017 -0400

    Merge pull request #39 from joshuazweig/tmp_remove

    Dis why the tmp files were there

commit 7b292e40b4ab533dca21b603a14398a8eb46524a
Author: Zack Silber <zs2266@columbia.edu>
Date:  Fri Apr 28 17:20:39 2017 -0400

    Dis why the tmp files were there

commit 95428d4599bf49a6b3228bd5925d659e4d9434a7
Merge: d906d06 f8d3eef
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Fri Apr 28 17:03:12 2017 -0400

    Merge pull request #38 from joshuazweig/tmp_remove
```

```
    How did these tmp files end up here? This is almost certainly my fault

commit fdd6eeeeaf11813449b135dba524c16e038e7efb
Author: Michael Tong <mct2159@columbia.edu>
Date:   Fri Apr 28 16:49:09 2017 -0400

    added test

commit 7b24ee69156b96d0765bdd591f94700952a72d9c
Author: Michael Tong <mct2159@columbia.edu>
Date:   Fri Apr 28 16:47:15 2017 -0400

    added curve_print function, still have the same problem with segfaults

commit 2445ad5d890f70b6d7edf9b62a7ae0d74402c0f7
Author: Michael Tong <mct2159@columbia.edu>
Date:   Fri Apr 28 16:39:42 2017 -0400

    First attempt at point arithmetic. Not working.

    Segfaults due to some memory issue surrounding curves it seems. Also
    when declaring a point, the print function prints the wrong numbers

commit f8d3eeffd1974253fa3303952a2fe298815af50e
Author: Zack Silber <zs2266@columbia.edu>
Date:   Fri Apr 28 16:18:20 2017 -0400

    How did these tmp files end up here? This is almost certainly my fault

commit 7a00365f8326e56d883a1360e1648218172891b7
Author: Michael Tong <mct2159@columbia.edu>
Date:   Fri Apr 28 15:23:04 2017 -0400

    Some fixes to arithmetic logic (mint_to_stone, stone_create)

    Mod exponentiation now works properly for negative powers.
    Stones get reduced to their positive modulus, even if they begin
    negative.
    Added "VER" variable to scripts.

commit d906d06607cb2f1b353d452da568dbf806d4ec6d
Merge: 75f7b49 c5c7c0b
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Apr 28 11:46:17 2017 -0400

    Merge pull request #37 from joshuazweig/jz-free

    Jz free

commit c5c7c0bc3468d4bcb15faa738e18dc52877a4ea9
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Apr 28 11:38:03 2017 -0400

    Update func type name

commit 50b9c4f8f8003ed5b7b37662b0363ac981db8ab0
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Thu Apr 27 23:03:54 2017 -0400
```

```
    Fix mint bug

commit 93f84970d91913a8cd0d79b5f555f8b68bbd1541
Merge: cb8f77b 75f7b49
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Thu Apr 27 22:55:22 2017 -0400

    merge commit

commit cb8f77bb93c67ecdb1e2092217c2fa4a58ae0bc4
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Thu Apr 27 22:52:59 2017 -0400

    Add free function

commit 75f7b4954c343e63cebe12719057e5c27206d2fe
Merge: 125974d 2618861
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Thu Apr 27 22:48:44 2017 -0400

    Merge pull request #36 from joshuazweig/mint_testing

    Adding mint tests. Plus a tiny bit of random cleaning.

commit 26188612d8131fd439171ee83d904ec5381daa50
Author: Zack Silber <zs2266@columbia.edu>
Date:   Thu Apr 27 21:32:35 2017 -0400

    Adding mint tests. Plus a tiny bit of random cleaning.

    Squashed commit of the following:

    commit 6996c06de9c62a774bccbe8f2f04efc93256ddb5
    Author: Zack Silber <zs2266@columbia.edu>
    Date:   Thu Apr 27 21:31:28 2017 -0400

        Make sure original mint custom test is fully modulated (and more ofc)

    commit 0c6cd7430f88f581ab35cb80895232fcec1c07bb
    Author: Zack Silber <zs2266@columbia.edu>
    Date:   Thu Apr 27 21:25:30 2017 -0400

        More mint tests

    commit 15c4ca5ae1d34b2d069b82850c57308195539852
    Author: Zack Silber <zs2266@columbia.edu>
    Date:   Thu Apr 27 21:12:51 2017 -0400

        Getting some initial testing up

    commit 6dfefc4edf8606b869879cfc897371836f8eb2ec
    Author: Zack Silber <zs2266@columbia.edu>
    Date:   Thu Apr 27 17:20:33 2017 -0400

        Setting up mint testing

commit 09e049cdd1482981fd2a695185def7f6b2fd757d
```

```
Author: Zack Silber <zs2266@columbia.edu>
Date:   Thu Apr 27 17:53:41 2017 -0400

    Syntax is a go

commit 125974d74069eadf28ec260779c2ba70deee29b0
Merge: 02b4947 9afd2ef
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Thu Apr 27 15:15:29 2017 -0400

    Merge pull request #34 from joshuazweig/mt_preprocess

    Python preprocessor working.

commit 02b4947876cc57ea37b2d7fc80a2d89efe70e168
Merge: 6e76c52 928fef3
Author: mikecmtong <mct2159@columbia.edu>
Date:   Thu Apr 27 15:14:44 2017 -0400

    Merge pull request #35 from joshuazweig/jz-bring-bn

    Change so stone printing is dec and update tests

commit 9afd2efbb25c1c0f6a95eb68228785da7a98e679
Author: Zack Silber <zs2266@columbia.edu>
Date:   Thu Apr 27 15:05:32 2017 -0400

    Switched to LLVM 3.8 for master.

    Also moved stuff out of master into custom_tests for now. Have to cd into
        custom_tests directory and ../.. back to the preprocessor right now to
        make it work because of the way it searches for files. But this will
        have to be addressed anyway when we have a library directory.

commit 928fef3a28968f4ed8ad6f840f41855996feb132
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Thu Apr 27 13:44:33 2017 -0400

    Change so stone printing is dec and update tests

commit 76f00e7100d912c1148f01073c0892069ba51518
Author: Michael Tong <mct2159@columbia.edu>
Date:   Wed Apr 26 18:13:38 2017 -0400

    Python preprocessor working.

    Use # followed by...

    include "filename": runs the preprocessor on the file specified by
    filename, replacing #include statement with the result

    define SYMBOL: adds SYMBOL to preprocessor's table (cf. ifdef, ifndef).
    Note: not useful for textual substitutions (e.g. #ifdef CONSTANT 42)

    ifdef SYMBOL: if SYMBOL is in preprocessor's table, continue. else, skip
    lines until #endif is found

    ifndef SYMBOL: opposite of ifdef
```

```
    endif: only relevant in the ifdef/ifndef cases above

commit 6e76c5298d05ecc6847ae2b206771899da0f55e2
Merge: 6c1efdb 4faf5fa
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Wed Apr 26 14:45:20 2017 -0400

    Merge pull request #33 from joshuazweig/mt_cleanup

    Removed all warnings from codegen

commit 4faf5fa37fc26b1ea324ea339760e5d48a9c769a
Author: Zack Silber <zs2266@columbia.edu>
Date:   Wed Apr 26 13:10:15 2017 -0400

    Bonus cleaning

commit 28447805a6081d7688738cf76805e6954b8e290b
Author: Michael Tong <mct2159@columbia.edu>
Date:   Wed Apr 26 12:40:36 2017 -0400

    Removed all warnings from codegen

commit c2401b83234cb9a27e94d4beeca9653c0276d25e
Author: Michael Tong <mct2159@columbia.edu>
Date:   Wed Apr 26 11:52:18 2017 -0400

    Fixed syntax error

commit 08012ef92192eaf6d207909e06af00dbc1da5520
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Wed Apr 26 02:13:30 2017 -0400

    Think this should work but syntax

commit e82095e053ef0ad3acea4a5e734753264d449bdc
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Wed Apr 26 01:07:28 2017 -0400

    trying mint pointer direct

commit 517be66dd50477da3171d60be8c2c2f419e21ea1
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Tue Apr 25 17:10:19 2017 -0400

    adding test

commit 414121997061763813590 9b43de75e870f229bd2
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Tue Apr 25 17:09:42 2017 -0400

    First attempt, success doubtful

commit 6c1efdb6cd35c674275da25dfdf911a323df3f5d
Merge: 19e7325 e234a88
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Tue Apr 25 16:37:51 2017 -0400
```

```
    Merge pull request #32 from joshuazweig/travis_patch

    Travis patch

commit e234a8894bdda3ffee86861554aaede51fbdd2e6
Author: Zack Silber <zs2266@columbia.edu>
Date:   Tue Apr 25 01:32:20 2017 -0400

    Hella stone tests and also switched pow to right associative

commit 4a59e79111911373fe924e0c188256868d08d184
Author: Zack Silber <zs2266@columbia.edu>
Date:   Tue Apr 25 00:23:59 2017 -0400

    Fully integrate all valid custom tests

commit 01ee3bb21c167bfbc90b974c91501e358f78e814
Author: Zack Silber <zs2266@columbia.edu>
Date:   Tue Apr 25 00:17:40 2017 -0400

    Custom test cleaning

commit f2547c8ae0471320a5df2085eae82b5b297b111d
Author: Zack Silber <zs2266@columbia.edu>
Date:   Mon Apr 24 19:41:19 2017 -0400

    Checking stability of new travis dist

commit 19e732585ee46ff22c35d5c3be261c4faf2ad240
Merge: 98e125e 6366de0
Author: mikecmtong <mct2159@columbia.edu>
Date:   Sun Apr 23 11:22:18 2017 -0400

    Merge pull request #31 from joshuazweig/mt_grammar

    Declarations are now flexible. Integrated with both codegen and semant.

commit 6366de0471b823c02281fd89a20ac65c69de2bff
Author: Zack Silber <zs2266@columbia.edu>
Date:   Sat Apr 22 22:12:00 2017 -0400

    Whoops accidentally tracked a temporary file

commit 1d15c40e58ef067f15c5e93b7bc72b1eef32613c
Author: Zack Silber <zs2266@columbia.edu>
Date:   Sat Apr 22 22:01:02 2017 -0400

    Update grammar tests, make sure we use llc 3.8 in master

commit 0b04c5ba3c28cc849b0029d7ebb99d7fa451aa8f
Merge: 8c6a064 4fbd4bc
Author: Michael Tong <mct2159@columbia.edu>
Date:   Sat Apr 22 21:52:23 2017 -0400

    Pulled tests

commit 8c6a06406d48d21c203b50a6ce52b3d5f5c57d5d
```

```
Author: Michael Tong <mct2159@columbia.edu>
Date:   Sat Apr 22 21:49:59 2017 -0400

    Checks for Stone = StringLiteral declaration correctly

commit 4fbd4bc234a39a8bbc8549217d373f20843af2b2
Author: Zack Silber <zs2266@columbia.edu>
Date:   Sat Apr 22 19:28:00 2017 -0400

    Fix tests and move into automated folder

commit 41d81ff82e02cd5c3093d4c2505fc0abb6341ba4
Author: Zack Silber <zs2266@columbia.edu>
Date:   Sat Apr 22 19:04:32 2017 -0400

    Fix one error test with new message, remove unneccessary files, move new
        tests temporarily to protect from make clean

commit f4f55385117cd5cdf67443a598f5dc95f3a36636
Author: Michael Tong <mct2159@columbia.edu>
Date:   Fri Apr 21 15:55:46 2017 -0400

    Declarations are now flexible. Integrated with both codegen and semant.

commit 98e125e638d10a6a7966c5a81d1fdc0939457d1f
Merge: 2ef7eb6 960dc56
Author: mikecmtong <mct2159@columbia.edu>
Date:   Fri Apr 21 15:06:01 2017 -0400

    Merge pull request #30 from joshuazweig/mt_crypto_arith

    Mt crypto arith

commit 960dc565d9f118e1eaf20e4180e1ff103725babf
Author: Zack Silber <zs2266@columbia.edu>
Date:   Fri Apr 21 14:57:30 2017 -0400

    Print stone test fixes

commit 21f4c29f6d33a25e9aa059fbed2bb8c7c5e7e821
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Apr 21 14:34:15 2017 -0400

    Update custom tests and fix syntax error

commit 793444e18ae87d9acebf339e9a17933deb4e3b48
Merge: cae0784 2ef7eb6
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Apr 21 14:30:01 2017 -0400

    merge commit

commit cae07842164419ed280318748556378ad8736380
Author: Michael Tong <mct2159@columbia.edu>
Date:   Thu Apr 20 14:54:40 2017 -0400

    Changed pow from ** to ^ (caused grammar problems).
```

```
commit 2ef7eb60cb0965a21924a81f75d420706290d622
Merge: e97f256 0716396
Author: zsilber <zs2266@columbia.edu>
Date:  Thu Apr 20 11:55:15 2017 -0400

    Merge pull request #29 from joshuazweig/jz-scanf

    Fixes #22

commit 0716396d2d42fd46437ed01b0de5bc0284e7e4a6
Author: Zack Silber <zs2266@columbia.edu>
Date:  Thu Apr 20 04:23:49 2017 -0400

    Add functions to semant, clean up some testing, need to generalize malloc
        return though

commit c09401116f3f68493cbcff203e01d8cd138a0ae3
Merge: acdce14 51a3515
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Wed Apr 19 13:45:33 2017 -0400

    Merge branch 'jz-scanf' of https://github.com/joshuazweig/C- into jz-scanf

commit acdce144575ec16f4fae0301819649a09ff8bac8
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Wed Apr 19 13:45:26 2017 -0400

    Add scan test

commit 0944d315d2ce12e7d961cea480639755dba09251
Author: Michael Tong <mct2159@columbia.edu>
Date:  Wed Apr 19 13:44:30 2017 -0400

    Declared point functions, changed Stone declaration string to be more
        intuitive

commit 647d4e3fb85a7f5958f29c4c2f1b0ca91c597041
Author: Michael Tong <mct2159@columbia.edu>
Date:  Wed Apr 19 12:55:12 2017 -0400

    Finished mint/stone operators. On declarations, value of Mints are
    reduced by their modulus (e.g. <23, 5> becomes <3, 5> internally)
    WARNING: Grmamar now has reduce/reduce error, since POW was implemented
    incorrectly before.

commit 51a351502d444123823e1302532985dc58466b44
Author: Zack Silber <zs2266@columbia.edu>
Date:  Wed Apr 19 02:30:02 2017 -0400

    Re-enable semant and prevent functions from being named scanf or malloc

commit 8c1d549c21ae1dfab5c5899d6dfe6463fe6289fd
Author: Michael Tong <mct2159@columbia.edu>
Date:  Mon Apr 17 19:26:20 2017 -0400

    Mint binops pretty much finished!

commit bb76bdf449c419355f2bd268ec806b24225c69ed
```

```
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Mon Apr 17 19:10:02 2017 -0400

    scanf works

commit 2030e4b28b395707177f80b23825f34717cf256a
Author: Michael Tong <mct2159@columbia.edu>
Date:  Mon Apr 17 18:50:15 2017 -0400

    Added print_mint function for now

commit bae7ce1ac4c6a11e2ec1f93dfc8373e8f27c1d7e
Author: Michael Tong <mct2159@columbia.edu>
Date:  Mon Apr 17 18:33:50 2017 -0400

    Got rid of int in mint definition to fix bug in special_arith

commit 3b65d088bcf71d963020e5f09ddc6b8faff21e9d
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Mon Apr 17 14:28:25 2017 -0400

    Typo

commit e97f25661856c5622b847e3b62715454db89604d
Merge: 90bf0fb 396a0a8
Author: zsilber <zs2266@columbia.edu>
Date:  Mon Apr 17 11:13:41 2017 -0400

    Merge pull request #20 from joshuazweig/jz-types

    Updated testing to work through cmod.sh.

commit 396a0a8cc383b46eabb05dece31a9c0a3e0ed90b
Author: Zack Silber <zs2266@columbia.edu>
Date:  Mon Apr 17 05:07:12 2017 -0400

    Updated testing to work through cmod.sh.
    Added -v flag to support travis integration.
    Link properly to newly necessary dependencies.
    Add tests for stone printing.

    Squashed commit of the following:
    commit 90c03457d009a32c2a95058602154712bfe4ec43
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Mon Apr 17 04:59:02 2017 -0400

        Flag position fixing

    commit bee6e98e327dae04d008ea34fc5cb78cfc686758
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Mon Apr 17 04:47:54 2017 -0400

        add support for v flag that uses travis variables

    commit a7f481a9500fd87a1f70a14662d0656d06327659
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Mon Apr 17 04:14:38 2017 -0400
```

```
    omg this will work

commit d35d7debb51adee4b825851a467b77775b92b27c
Author: Zack Silber <zs2266@columbia.edu>
Date:  Mon Apr 17 04:08:19 2017 -0400

    fixing

commit d00d4ef57b62595cd6019831daf6e2bcd34e2c42
Author: Zack Silber <zs2266@columbia.edu>
Date:  Mon Apr 17 03:58:17 2017 -0400

    Finally runs, output differs tho

commit bed6509878ac0a47dabcf71e7e4594702966adb7
Author: Zack Silber <zs2266@columbia.edu>
Date:  Mon Apr 17 03:49:44 2017 -0400

    found libcrypto link

commit 0ce989d198eee18ba48d4111b8d5eb8ef89c254d
Author: Zack Silber <zs2266@columbia.edu>
Date:  Mon Apr 17 03:39:06 2017 -0400

    new print strat

commit 7ba4215b952b6ff70239c8fd85d369f686ca9f0a
Author: Zack Silber <zs2266@columbia.edu>
Date:  Mon Apr 17 03:28:25 2017 -0400

    idk man

commit d465e2b0169039f7475dd719ecbc162a1e4548e2
Author: Zack Silber <zs2266@columbia.edu>
Date:  Mon Apr 17 02:39:38 2017 -0400

    Trying another thing

commit 6ba0a4a66624b90842c1d04e02a05b6f41ba29cb
Author: Zack Silber <zs2266@columbia.edu>
Date:  Mon Apr 17 02:37:10 2017 -0400

    Attempt 2

commit 0da2bf7ca514c5e6e0e7e5de63e5ee3595ddd97d
Author: Zack Silber <zs2266@columbia.edu>
Date:  Mon Apr 17 02:33:43 2017 -0400

    installing needed dependencies on travis

commit aa48eb470941ea9cdccfc3cf08f5c6c717416978
Author: Zack Silber <zs2266@columbia.edu>
Date:  Mon Apr 17 02:26:21 2017 -0400

    I hope I have libcrypto in travis

commit 5180c529e5c93a933726758ebcf3e61eb60f05dc
Author: Zack Silber <zs2266@columbia.edu>
```

```
    Date:  Mon Apr 17 02:05:53 2017 -0400

        Omg I have not been doing make all

    commit 4842fe10ce697b42e3c39d776ef41b52fb943b25
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Mon Apr 17 01:59:42 2017 -0400

        I am hot on the travis trail

    commit f8c0e728895d8318eba22a33f03ed5c2de792d86
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Mon Apr 17 01:52:02 2017 -0400

        Travis makes me commit so many times

    commit 1026c48f3d842d3c48d18accbf5b3bffaca356ad
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Mon Apr 17 01:36:28 2017 -0400

        Playing with travis LLI path in shell script

    commit 6691ac74563a7f2a35c95e9eb01c7b6b4c0864df
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Sun Apr 16 19:13:18 2017 -0400

        Fix attempt, I hate travis

    commit 910498d0f0001a4e9af09fdd05f2ea2642afc5ae
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Sun Apr 16 18:52:12 2017 -0400

        Fixing travis LLI path

    commit ee0a86b6818b2947d30128b8775087aa10f87240
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Sun Apr 16 18:41:48 2017 -0400

        Phone is not Phony

    commit 30e86b3abbedfa609df19e8c8c5480a6d8249109
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Sun Apr 16 18:40:20 2017 -0400

        Fixing testing to use cmod.sh, added tests for stone printing

commit 73b538e120a48a59e7c6c8eb459198d35e1afab6
Author: Michael Tong <mct2159@columbia.edu>
Date:  Sun Apr 16 18:47:57 2017 -0400

    for stones, BN_new() in the C file instead of codegen. also mints segfaulting

commit 90bf0fbd549ade6a04dd51ca972a010581a6fc2f
Merge: 88001d7 01b3d57
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Sun Apr 16 16:24:03 2017 -0400

    Merge pull request #17 from joshuazweig/jz-types
```

```
    Jz types

commit 01b3d57935f489ea9cd33cfa2726b577061a1d3e
Merge: 143a8d5 c37c26f
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Sun Apr 16 16:19:09 2017 -0400

    Add print_stone

commit 143a8d50d33dfbae5704f3bfac319eef7088aab8
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Sun Apr 16 16:12:59 2017 -0400

    pulling

commit c37c26f8276dd8903ad660fa454b12cefcd48c52
Author: Michael Tong <mct2159@columbia.edu>
Date:   Sun Apr 16 16:10:50 2017 -0400

    Added more built in declarations

commit a2faa12bbab0a89aba9763308e4102aae7034c37
Author: Michael Tong <mct2159@columbia.edu>
Date:   Sun Apr 16 15:49:35 2017 -0400

    added case for assigning string literals to stones

commit a51ac5807a9ffaf570ff5a9eeba7f00e652d8a7f
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Sun Apr 16 00:28:11 2017 -0400

    stone stuff is great

commit 5ca4fb0b04a5f8173ec511dcc910eeac9d021659
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Sat Apr 15 20:49:56 2017 -0400

    Add primary stone functions

commit b297e02bd90e885e7a92ea917a48c4c032bda68e
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Sat Apr 15 03:43:03 2017 -0400

    Stones are in buisness

commit e778f21c4afeefa6dcd19b822c8aca7c8bcb526d
Merge: a306b9a 88001d7
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Sat Apr 15 01:54:18 2017 -0400

    Merge branch 'master' into jz-types

commit a306b9a1180292684b312d701e77f7091b9f79e3
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Sat Apr 15 01:43:55 2017 -0400

    Resolve stack overflow in ast, codegen
```

```
commit 88001d774b4c96987f097a8927f3a8d87eb64011
Merge: cd7a2ce c0cf48b
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Tue Apr 11 22:06:23 2017 -0400

    Merge pull request #14 from joshuazweig/jz-codegen

    Resolve semant issue

commit c0cf48bc5ac0f485df48bedfcfa91ce4870813e5
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Tue Apr 11 22:05:32 2017 -0400

    Resolve semant issue

commit cd7a2cef768061cc624b73585339c9a7c1ab1a2b
Merge: 7f673b1 d0160a0
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Tue Apr 11 20:49:41 2017 -0400

    Merge pull request #13 from joshuazweig/jz-construct2

    Jz construct2

commit d0160a062b07a4d854141711085d408bd21f0421
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Tue Apr 11 19:38:06 2017 -0400

    Update makefile and temporarily remove construct3

commit d2a529809455729bac32467cda193d998fec85bf
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Tue Apr 11 19:33:36 2017 -0400

    Add test

commit ef0ddf956d46ae138dbbe215ac09bd03f876f4a1
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Tue Apr 11 19:32:06 2017 -0400

    Appears to be bug in semant

commit db68eed5c3ce8e3cbd4f7b59dbe8305459ee6721
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Tue Apr 11 19:11:37 2017 -0400

    construct2 set up, waiting on better stones to test

commit e57625a73b5aa7bfd9171fa72a565c310ed3d93a
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Tue Apr 11 18:40:37 2017 -0400

    Trying to resolve struct issues

commit a05728b3bf9d7777d0b41a93a985b0526bf26648
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Apr 10 21:01:19 2017 -0400
```

```
    C arith function defs

commit 555f8d218ac16501f7e204fcc22ec86deedad2fe
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Apr 10 20:54:50 2017 -0400

    First run

commit 21291e6b0d65f2345c82a1af79b2b0dceb0d6d1e
Merge: b6591c4 7f673b1
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Apr 10 20:54:19 2017 -0400

    Merge branch 'master' of https://github.com/joshuazweig/C-

commit 7f673b1020ff711e4a1e20b9f57747c44795f39a
Merge: 43f8a53 16c0e38
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Apr 10 18:54:31 2017 -0400

    Merge pull request #10 from joshuazweig/codegen

    Codegen

commit 16c0e38a595f2e372824ee3f920cd8c72ea0323b
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Apr 10 18:45:15 2017 -0400

    Update Makefile

commit b6591c412bf412f1309c69e826539a17e6ae38d0
Merge: 0764842 43f8a53
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Apr 10 18:37:30 2017 -0400

    Merge branch 'master' of https://github.com/joshuazweig/C-

commit 1ae8b7f60ddb84d123c112310ef53c2a63b6a995
Merge: 8258029 43f8a53
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Apr 10 18:36:07 2017 -0400

    Merge branch 'master' into codegen

commit 82580298936ea969b1ab9f45e5c8cb286dada322
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Apr 10 16:57:54 2017 -0400

    Begin work on stone contrution

commit 61403980984176cad7641db4585a65103892ba7d
Merge: e4ef7e6 5c29a98
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Apr 10 16:04:45 2017 -0400

    merge commit links
```

```
commit 5c29a98496a3cbfae0aaae9790dbc3db8ecf971f
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Mon Apr 10 16:01:14 2017 -0400

    Got linking working woohoo

commit 43f8a53d1ed1e96dc85e93040383c96f46f0529a
Merge: 646b8e8 2d81f3c
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Sun Apr 9 23:31:40 2017 -0400

    Merge pull request #9 from joshuazweig/zack_testing_branch

    Travis Integration

commit 1eb8b2677ef78dab1faa65aadd47b345c88ded30
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Fri Apr 7 13:59:43 2017 -0400

    changing to pointer passing

commit e4ef7e67e565efb93cf7a2b427c77149cf900460
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Fri Apr 7 13:45:00 2017 -0400

    reverting...

commit 83732d67eb195ac58900baf1c7b384e22350d7c1
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Thu Apr 6 12:44:36 2017 -0400

    this is a mess

commit ce935133c1c36677320b052c166dc716a5b35914
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Thu Apr 6 02:04:59 2017 -0400

    We're close

commit c7b8d66f31f52a839781aff41771cc8b7c9fe1a8
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Wed Apr 5 23:42:05 2017 -0400

    about to redo functino

commit de09acb70c0686829a973bf17b306455d9fb2c28
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Wed Apr 5 22:28:49 2017 -0400

    its building

commit 6a3e99da4231aad4e9699f188b03cef0fa99e613
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Wed Apr 5 21:10:35 2017 -0400

    Think i have a working structure

commit 2d81f3ccf00cfedb2c0e745971df143deabb2e09
```

```
Author: Michael Tong <mct2159@columbia.edu>
Date:  Wed Apr 5 20:57:10 2017 -0400

    Added char lits

commit 9a5e92173b00b4fd0da9e99e8afa8c05c9fe3ce6
Author: Zack Silber <zs2266@columbia.edu>
Date:  Mon Apr 3 14:37:44 2017 -0400

    Squashed commit of the following:

    Finished grammar testing suite. Sanitized all relevant MicroC tests. Set up
        Travis continuous integration. Overhauled the organization of testing
        suite to be more intuitive and work with the automated testing workflow.
    commit 71264a61b54bea3504a32796c303c0dca87fa368
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Mon Apr 3 14:15:30 2017 -0400

        Add seperate flag for running test script in travis, restored all
            packages, turns out they were necessary, shocking

    commit bf2395064964935c0a35ffdb398e18260689a72c
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Mon Apr 3 13:39:21 2017 -0400

        Okay, sorry, re-adding

    commit 8f05565cbee2066e874162870acd4abb10010a8c
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Mon Apr 3 13:38:02 2017 -0400

        build did not start last commit

    commit 65afd10e871d898d174f57190286e7ad479f8f62
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Mon Apr 3 13:35:16 2017 -0400

        Works, seeing if I can remove some extra steps to speed up build

    commit e659a48c59be85b51b2d6a2e7e0dc54e790dbc9a
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Mon Apr 3 12:16:12 2017 -0400

        Okay but lowkey this one is gonna work

    commit 533fed79ab180e27d7283ac181be2fbb51b8ffaf
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Mon Apr 3 11:51:33 2017 -0400

        Debugging lli path

    commit 59969068c14e49a95e01982434679a96d3e23c8c
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Mon Apr 3 11:27:04 2017 -0400

        Change testall LLI path

    commit 6b602d74cd6fc98206ad1f7467d88be4c732e34f
```

```
Author: Zack Silber <zs2266@columbia.edu>
Date:  Mon Apr 3 05:12:04 2017 -0400

    What did I just break?

commit 304c36ff78400be7b5b4c419b16c080add553700
Author: Zack Silber <zs2266@columbia.edu>
Date:  Mon Apr 3 04:33:57 2017 -0400

    change llvm path

commit 17de3b65aede88c42372162c9a44c13b044f7462
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sun Apr 2 23:40:15 2017 -0400

    Compiler testing attempt 2

commit ad68af65dab27425b3349a04d8256e12d57bac5b
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sun Apr 2 23:29:01 2017 -0400

    Add back failing grammar tests, attempt compiler testing as well

commit b3873a3467000987993897e2da5cebe89610c343
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sun Apr 2 23:22:03 2017 -0400

    temporarily remove failing grammar test

commit aae429732bbdd40178811362ddc67892791a4542
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sun Apr 2 23:11:18 2017 -0400

    Trying to install llvm ont ravis

commit 98c32526c9104b2f78446829ed84fffb9bbaee5a
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sun Apr 2 22:50:34 2017 -0400

    Makefile changing for travis

commit 3edc245f9de334d9443959a3e748462559ed14ed
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sun Apr 2 22:38:11 2017 -0400

    travising

commit 8a7a6f1ba6cb90d6b988983859cd8f3f480ff4f0
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sun Apr 2 22:14:13 2017 -0400

    Make in travis testing

commit acb4c4113f8cf94eccb5a91dc885e8e8eae1eb47
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sun Apr 2 22:06:36 2017 -0400

    add ocaml env
```

```
commit 51ff054be984af943f3fdf807a245dc18890e4bd
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sun Apr 2 15:51:01 2017 -0400

    travis attempt 2

commit c629af71a864d1d392b1eda1df7361438b281938
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sun Apr 2 15:46:00 2017 -0400

    Travis dependency attempt

commit 292109d9b87ef7152c71b5bc6ad0ac464f431850
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sun Apr 2 15:36:03 2017 -0400

    grammar tests now ran from home directory, added travis support for
        grammar testing

commit 3c9d5d9695dbcfc1f4bcbd5dee8685bcbfcd3dad
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sun Apr 2 15:05:00 2017 -0400

    Add exit code to grammar testing script

commit c2b4e902eb18b37d1d177035bcd5eccf8f32d080
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sun Apr 2 14:40:59 2017 -0400

    Deleting old testing structure, hopefully

commit a0eb05716d8109c545f3569f0711d083507c43d0
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sun Apr 2 14:39:58 2017 -0400

    New testing file structure

commit 37920502da087573fb3319938fe976d7cdce4d31
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sat Apr 1 01:56:09 2017 -0400

    Quote changing

commit f11bf4d5fe2766f1ab3eddeae498eb414b6d3dcf
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sat Apr 1 01:49:23 2017 -0400

    All tests fixed (a few irrelevant ones removed)

commit 3948e58c02fcfdff46352d519e73c1840a33551a
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sat Apr 1 01:27:09 2017 -0400

    More test fixing

commit ccfe5144f0feec4644fd9d3a37ef3232cf38753e
Author: Zack Silber <zs2266@columbia.edu>
```

```
    Date:  Sat Apr 1 00:34:30 2017 -0400

        Putting bracket array notation on indefinite hold

    commit 7a5ddff4b4e4581721b96178da9181f7e4029431
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Fri Mar 31 23:10:55 2017 -0400

        Fix print statements

    commit 2e7d98f6c78db77c34285cd04c36c65a60e8e26b
    Author: Zack Silber <zs2266@columbia.edu>
    Date:  Fri Mar 31 05:10:13 2017 -0400

        test file extensions changed, fail tests sanitized, prettyprinter
            updated, string literal and array grammar tests added

commit 7aad6dd0c415700f27e3b5b557fd21a113582861
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Sat Apr 1 19:10:59 2017 -0400

    Add type annotations

commit 8770fa0e3d95972153c6f802b6d064d07ba20d6f
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Fri Mar 31 18:03:02 2017 -0400

    update codegen

commit 56426d0df774ae09d0fe52eec8e58141ad6852ae
Merge: 8f4dd6e 3d0c8e6
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Wed Mar 29 17:02:48 2017 -0400

    Rework type pointer structure

commit 3d0c8e6b3294741292406066787d7c91e1a57aa1
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Wed Mar 29 17:01:10 2017 -0400

    turns out pointers arent really typed in LLVM, and void pointers not
        allowed. Revamping pointers to 32 bit int type

commit 0764842dcc1fc5d7195a30469f4270238a02db01
Merge: a3f42be 646b8e8
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Wed Mar 29 16:37:44 2017 -0400

    merge commit

commit 646b8e83d0f1b995af3a2b09790f571ec13cdf60
Merge: dfdf627 a031006
Author: mlm2299 <mlm2299@columbia.edu>
Date:  Wed Mar 29 14:47:29 2017 -0400

    Merge pull request #7 from joshuazweig/hello_world

    Hello world branch !!!!
```

```
commit a031006526c607f1acbef8aabff7c8d2ac2dabed
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Tue Mar 28 10:31:57 2017 -0400

    Remove dblquote token from parser, went unused -- to be added later

commit c910daaf5cbbcfd734262c2e58cbcff892196c0b
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Tue Mar 28 10:29:06 2017 -0400

    Clean scanneprint.ml in Makefile

commit 90e17a1d60fb7b90b5cc585680996692a0cf6ca0
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Tue Mar 28 10:27:05 2017 -0400

    Remove old pretty print features from cmod.ml

commit 6ceb7c411aa8eab8d7c13a867ce2d1756766ba0e
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Mon Mar 27 19:20:12 2017 -0400

    Add .cm file

commit 745909a4efae8d69fa170b5e226e594a6604de8a
Merge: 47a41a3 dfdf627
Author: mikecmtong <mct2159@columbia.edu>
Date:  Mon Mar 27 19:16:03 2017 -0400

    Merge branch 'master' into hello_world

commit dfdf627fff2073f08aca2b7b9902a0c7d938d5ad
Merge: a62a991 d57a162
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Mon Mar 27 19:11:43 2017 -0400

    Merge pull request #6 from joshuazweig/rz_testing_suite

    Added ocamllex scannerprint.mll to Makefile

commit d57a1621ec80030ab4a9c7cfab92fda66ff23293
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Mon Mar 27 19:10:52 2017 -0400

    Update Makefile

commit 8f4dd6ea5c975c47f82ae4e1874382500b3df805
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Mon Mar 27 19:08:43 2017 -0400

    update make

commit b7317c939c5b4241e20a4b2a5a2fac5c4b898265
Author: Richard Zhang <rzh4ng@dyn-129-236-234-170.dyn.columbia.edu>
Date:  Mon Mar 27 18:56:06 2017 -0400

    Added ocamllex scannerprint.mll to Makefile
```

```
commit 47a41a3d4717b54a9a4ce1f8c4a7fe0d661926e4
Author: Michael Tong <mct2159@columbia.edu>
Date:   Mon Mar 27 18:35:55 2017 -0400

    Changed access return type

commit 6d35ea25c444ea506172c0fcc48dbd5924c0c202
Author: mikecmtong <mct2159@columbia.edu>
Date:   Mon Mar 27 18:18:12 2017 -0400

    Change char from i1_t to i8_t

commit 6f80dfe4cd6c646f0c854c3a44b571bdce32508f
Author: Michael Tong <mct2159@columbia.edu>
Date:   Mon Mar 27 15:12:06 2017 -0400

    added single-line comment capabilities

commit 0673eb6f9854729475e07e3f663d9d94e95b95c5
Author: Michael Tong <mct2159@columbia.edu>
Date:   Mon Mar 27 14:55:10 2017 -0400

    Pattern matching in expr function in semant.ml is now non-trivially
        exhaustive

commit c41903d15344a4b327002f1410ab51d8197a6777
Author: Michael Tong <mct2159@columbia.edu>
Date:   Mon Mar 27 14:19:48 2017 -0400

    Added more functionality to semant type checking (e.g. dealing with
        pointers, special crypto types, etc.)

commit a3f42be41fd33f3d6fbab0f668cb2c70a77374b8
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Mar 27 13:47:05 2017 -0400

    branch change

commit a62a9913cfa935efb3e50dc32fa509f52d45c495
Merge: f343f66 eebd289
Author: zsilber <zs2266@columbia.edu>
Date:   Mon Mar 27 13:45:00 2017 -0400

    Merge pull request #5 from joshuazweig/zack_testing_infrastructure

    Zack testing infrastructure

commit eebd289146f12c600b3c61ebe1c60acc08721265
Author: Zack Silber <zs2266@columbia.edu>
Date:   Mon Mar 27 13:42:08 2017 -0400

    Comments for scanner and scannerprint

commit 50bfd1a9246cd46f493bb2179fe55c97fb513176
Author: Michael Tong <mct2159@columbia.edu>
Date:   Mon Mar 27 13:31:21 2017 -0400
```

```
    Fixed small bug, changed main-not-defined error message

commit 19acec321d7341f3972f8455f90b03aefb93ddc9
Merge: 98e20a1 7e8a6c3
Author: mlm2299 <mlm2299@columbia.edu>
Date:   Mon Mar 27 12:40:14 2017 -0400

    Merge pull request #4 from joshuazweig/jz-types

    Add support for all types

commit 2722121e42fc0945ba1c9ecbe2cfc8511bf21f4e
Author: Zack Silber <zs2266@columbia.edu>
Date:   Mon Mar 27 00:08:55 2017 -0400

    Change file extensions.

commit 5a368c1ac7ae4c839d1822917b77a44f7c1b7cd1
Author: Zack Silber <zs2266@columbia.edu>
Date:   Sun Mar 26 23:43:13 2017 -0400

    Added type tests

commit 17aa22d5e1c39e6b41e5fcab976e117ec4a86cf0
Author: Michael Tong <mct2159@columbia.edu>
Date:   Sun Mar 26 23:40:36 2017 -0400

    Semant compiling, catching stuff it should

commit 7e8a6c3132b6b4b84de677e48f4687220f09e0e1
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Sun Mar 26 16:29:30 2017 -0400

    Add support for all types

commit 7cd52f7fff8348ad9cc176c9fbbd55b13a3f68e7
Author: Zack Silber <zs2266@columbia.edu>
Date:   Thu Mar 23 16:52:33 2017 -0400

    added pretty testing script, a few new tests, manual out for each test

commit 4bdf31ea38f056b43b2eef4383d6494a58758f52
Author: Zack Silber <zs2266@columbia.edu>
Date:   Thu Mar 23 01:08:53 2017 -0400

    Stop tracking build files

commit 6dd57053a052481da576b708de6f6e34585b6dea
Author: Zack Silber <zs2266@columbia.edu>
Date:   Thu Mar 23 01:07:13 2017 -0400

    More tests

commit 46d0326e4d7c350fefa08729ae668a745c259577
Author: Zack Silber <zs2266@columbia.edu>
Date:   Wed Mar 22 22:15:08 2017 -0400

    More testing
```

```
commit 525e85a0be58d302e34f1ac90b77b4e76b48c16a
Author: Zack Silber <zs2266@columbia.edu>
Date:  Wed Mar 22 21:10:12 2017 -0400

    Killing old print method

commit 2884583d5f19b76b2100e997be0921a08e8052c3
Author: Zack Silber <zs2266@columbia.edu>
Date:  Wed Mar 22 21:08:07 2017 -0400

    Adding new pretty printing method, new test case, setting up global git
        ignore for dsstore

commit a4a400692abb7414d370dbf3b78aa7ef00282cbe
Author: Zack Silber <zs2266@columbia.edu>
Date:  Tue Mar 21 18:11:31 2017 -0400

    Binop Tests

commit 1a198094b489d73ebf1029b6d2cc2a441d8d01ef
Author: Michael Tong <mct2159@columbia.edu>
Date:  Mon Mar 20 18:40:37 2017 -0400

    Fixed extra new lines in strings

commit ae798eba892fc8aafc93e434d3cc90c6ef730c78
Author: Zack Silber <zs2266@columbia.edu>
Date:  Mon Mar 20 18:14:24 2017 -0400

    More spacing fixes, literal handling

commit 98e20a1928318a38028a1014a22ddf3197b8c9da
Merge: 81af521 816ed16
Author: Michael Tong <mct2159@columbia.edu>
Date:  Mon Mar 20 17:58:56 2017 -0400

    Merged mt_printf and codegen

commit 816ed16ca546dd2226c1d40eebee1f8ae1de1c6f
Author: Michael Tong <mct2159@columbia.edu>
Date:  Mon Mar 20 15:50:30 2017 -0400

    Big string test works

commit 31c65acbb709d94d70cd37e2d055f0334e7bb6fe
Author: Michael Tong <mct2159@columbia.edu>
Date:  Mon Mar 20 15:47:46 2017 -0400

    String with escape characters working.
    Supported: \n \r \t \' \" \\

commit 81af521738e2601f0e260d0d8cf9d80521956302
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Mon Mar 20 14:10:40 2017 -0400

    Add shortcut to run program
```

```
commit f9400d7bddfcf33939bc1d6ced5d9d72d776b794
Author: Michael Tong <mct2159@columbia.edu>
Date:   Sun Mar 19 19:29:32 2017 -0400

    Hello world working

commit 5acfe422b88f07f2aa49e48d7278d05e70296e36
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Sun Mar 19 19:25:10 2017 -0400

    remove dsstore

commit 7e8490eea2dcde24b09459aaf2a8bc9e5b7c1a17
Merge: b441059 e2a5a32
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Sun Mar 19 19:24:06 2017 -0400

    Hello world

commit e2a5a32fc7462ed1882a25431a18db97db601614
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Sun Mar 19 19:13:16 2017 -0400

    Remove build files

commit ac86f72daa5818151cf03a530b55eca10d657466
Author: Michael Tong <mct2159@columbia.edu>
Date:   Sun Mar 19 19:11:17 2017 -0400

    strings properly implemented in parser and scanner, -t flag added

commit b441059bb93562119b57a309f814ac505448105c
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Sun Mar 19 19:11:00 2017 -0400

    working

commit 18f53558b6878a19e8d86bcdfdeebdb6d80a06ce
Author: Michael Tong <mct2159@columbia.edu>
Date:   Sun Mar 19 18:45:05 2017 -0400

    Fixed string tokens, added printf call to codegen

commit 2cd00e3364206b42a682a2325b49502cf6346365
Author: Zack Silber <zs2266@columbia.edu>
Date:   Sun Mar 19 18:38:08 2017 -0400

    Unused variable cleanup

commit 003287c5615102b6a0d697bbbafae03d133f9fe4
Author: Zack Silber <zs2266@columbia.edu>
Date:   Sun Mar 19 18:35:28 2017 -0400

    Quick fixes

commit 6886a72b6926e1ace4b2caa9d9050412067619fa
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Sun Mar 19 17:13:18 2017 -0400
```

```
    printf working

commit c9ea8ae940560d534faa16e24f27311928dbb773
Author: Zack Silber <zs2266@columbia.edu>
Date:   Sun Mar 19 16:53:11 2017 -0400

    Too many empty lines

commit 612ebfe81a7aab093d519651cbfced5dfd49eb1a
Author: Zack Silber <zs2266@columbia.edu>
Date:   Sun Mar 19 16:50:11 2017 -0400

    Fixed pattern matching

commit 4d60d4093c80eebaac2ba223f27cc238453a27a6
Author: Zack Silber <zs2266@columbia.edu>
Date:   Sun Mar 19 16:34:53 2017 -0400

    Cleaned up unused variables

commit 794da2a690faae4594d106309bfd822f6a1d7327
Author: Zack Silber <zs2266@columbia.edu>
Date:   Sun Mar 19 16:26:30 2017 -0400

    Ironed out some spacing issues with tokens

commit 1a33dda3888f277fe1261fdddf25aeaa1c3b92af
Author: Zack Silber <zs2266@columbia.edu>
Date:   Sun Mar 19 16:09:38 2017 -0400

    Token print mainly finished, a few TODO comments are left for things that
        might become problems as we test out the printer

commit 8f3e33f5408b4ce6b7a1a3e72c498bbb74420ab0
Author: Zack Silber <zs2266@columbia.edu>
Date:   Sun Mar 19 15:56:17 2017 -0400

    Token printing functions now all call each other

commit d92accfd0adb8eccbee4c37a069282f8f5be7bbf
Author: Zack Silber <zs2266@columbia.edu>
Date:   Sun Mar 19 15:51:55 2017 -0400

    Added token printing strings for types and operators, not sure how to handle
        pointers right now

commit 52f32c0445e42cec3091d0febbeb55547ec84308
Author: Zack Silber <zs2266@columbia.edu>
Date:   Sun Mar 19 15:45:39 2017 -0400

    Added t flag for tokenized string, started building token print function

commit f343f66d591a6dcc574f8971d42651a53270bf87
Merge: 28ffb18 727ade3
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Sun Mar 19 00:19:52 2017 -0400
```

```
    Merge pull request #2 from joshuazweig/rz_scanner

    Current grammar, pre pretty print testing

commit 727ade3f640246e842c1b2e12f7041a40176662d
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Wed Mar 8 21:15:37 2017 -0500

    Resolve all unused in parser

commit 71ce44b72a4cdf042d2c48326a65e004694f3c58
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Wed Mar 8 21:08:40 2017 -0500

    Looks good but a few unused tokens

commit d465596c8a2263178bdddc72e4308f206edc0795
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Wed Mar 8 21:01:13 2017 -0500

    remove unused nonassoc

commit a4f4aa45d9b387e295631f5c976cee70a39fb290
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Wed Mar 8 20:36:43 2017 -0500

    Fix typo

commit 39891a5bb6063f63ecf5e3622f2d9f0eb0e722aa
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Wed Mar 8 20:26:18 2017 -0500

    Change built in type construction in grammar

commit a26ebcdb33be22d187cbfaba4e76402ea93f6894
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Wed Mar 8 13:26:05 2017 -0500

    \Remove lval comments

commit 3ed4382a665ffea3af423d6c9064c2dd990f068f
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Mar 6 21:17:56 2017 -0500

    Revise stmtlist production

commit 46c34ee0913f79491ee89d5207a3bee6a7f1ff4a
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Mar 6 20:53:17 2017 -0500

    updated scanner typo

commit 63862365797467230c27e6af68c4b4be20277f48
Author: Michael Tong <mct2159@columbia.edu>
Date:   Mon Mar 6 19:57:43 2017 -0500

    added all phony to makefile
```

```
commit 83408e48dcfb1b6d752300d5027883fe6e1ba8a7
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Mar 6 19:44:17 2017 -0500

    Revise AST

commit b59aab3345d2497ae58eff163a3ef51c526b94b7
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Mar 6 19:41:46 2017 -0500

    Finished revising grammar

commit 3c5cf5a1b255895c46eaf52f2bd175fabba8bc05
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Mar 6 19:32:06 2017 -0500

    Revise scanner and change tokens STAR DEREF, etc

commit 6b987e172740f6d90eee0c73596688ede0c311c1
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Mar 6 19:07:56 2017 -0500

    Remove open AST and ignore unused var warnings

commit ed620974e385cd4e6f83db7b615273772a8e8ea7
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Mon Mar 6 18:55:13 2017 -0500

    Remove pointer warning

commit 1cd60ffd6d24e36262a868909508a5a2484a120e
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 23:24:52 2017 -0500

    Update scanner

commit aff9e46c4f3d0ad38a6e39f72a76f6719511460a
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 23:18:10 2017 -0500

    Adding Makefile, cmod.ml and sentamt.ml to allow for testing

commit bde59c48eb245db655a29e4afefa520f2d1886c5
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 23:12:18 2017 -0500

    Changed parser def of assign,modassign

commit 1d112d473432bc96677807511aa87e39750d430a
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 23:10:26 2017 -0500

    Removed lvals

commit 335911f5f7f57531b0256cb2ce43ab9457bfb6a5
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 23:08:51 2017 -0500
```

```
    Changed char to be interpeted to ocaml stirng

commit 9a594edfdd304e9fccd5eb651e16b5c1901db9e1
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 23:07:20 2017 -0500

    changing char to string, but might be a better way to go anyway

commit 829e7147d8c917d6bd623def3f2b80b255f92705
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 23:02:25 2017 -0500

    Banging out inconsistency

commit 7a055eba1ac34c047fe7085c1313b791c7a178c0
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 22:58:30 2017 -0500

    Update exp op -> pow

commit e5dcfb7a9728d8f6c064f3a208c1b6aefff4d318
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 22:55:58 2017 -0500

    Add pointer to AST

commit aa12f3ef7735abb76bd42b08fef8317b7795f378
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 22:39:40 2017 -0500

    removing bool, false,true from scanner

commit 96a9abe92131b7b7f52ea0c82b368881cab0d4d2
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 22:39:33 2017 -0500

    removing bool, false,true from scanner

commit caf8d9513c4270aa63826035ab33aa0eada67ae6
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 22:38:16 2017 -0500

    revise inf in ast

commit b92a823416e239f0b8fdbfc9866d3beee388685b
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 22:37:47 2017 -0500

    revise inf in ast

commit f67fe5a0bde0af0e4b38b99811d4014f1e255493
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 22:28:49 2017 -0500

    Added DEREF in addition to star, but not sure if need prec in 123

commit 78d0d089eda47e45b508bbfc525b11efb522ee32
Author: Joshua Zweig <jmz2135@columbia.edu>
```

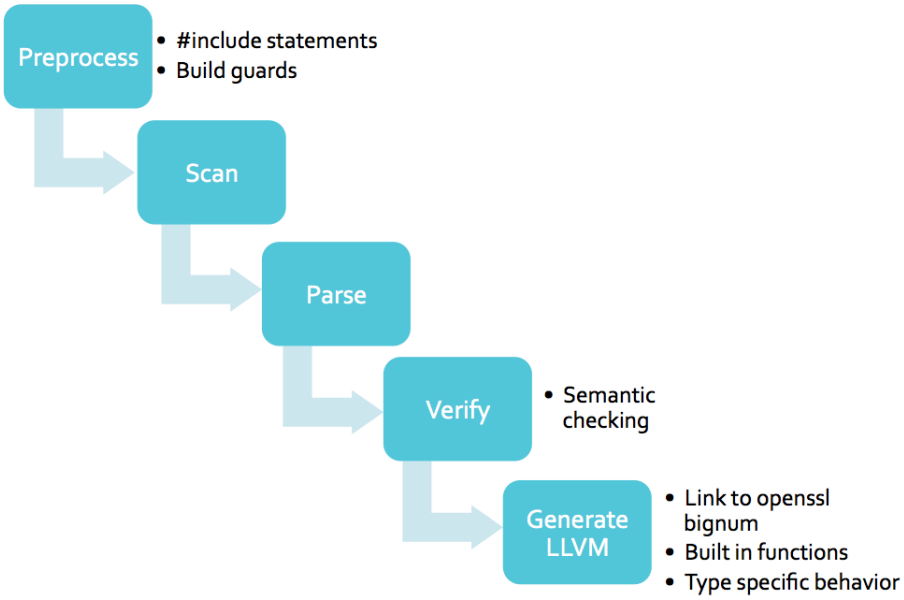
```
Date:   Fri Mar 3 22:24:55 2017 -0500

    Changing modassign back

commit d21448765e269fe7f13a4a12032d794c696a4a1e
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 22:21:25 2017 -0500

    Rename dash -> minus in scanenr

commit a252b1b23c6905e158cd87a18572850f83b18e5f
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 22:20:32 2017 -0500

    Remove mod assign and adjust corresponding production

commit 906baa55d28462daf1db787ecd2e1f6d2f031485
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:   Fri Mar 3 22:18:31 2017 -0500

    Add inf to ast.ml

commit ccc2c5c8655bb299c71dda5dfa9b8fc408c215cf
Author: Richard Zhang <rzh4ng@dyn-129-236-234-140.dyn.columbia.edu>
Date:   Fri Mar 3 19:55:41 2017 -0500

    Preliminary changes to parser.mly, changes to ast.ml are
    nothing (says there are changes, but are none).

commit eb474980532f76884a6078ff37e3908b58ac6e18
Author: Richard Zhang <rzh4ng@dyn-129-236-234-140.dyn.columbia.edu>
Date:   Fri Mar 3 19:52:53 2017 -0500

    Created scanner.mll, preliminary changes to parser.mly to account
    for scanner changes.

commit 28ffb182d79ef940d7a2cdb1071fe19e17e435e9
Merge: 41c5779 f88551d
Author: mlm2299 <mlm2299@columbia.edu>
Date:   Fri Mar 3 19:34:41 2017 -0500

    Merge pull request #1 from joshuazweig/mm-ast

    AST updated and now compiles

commit f88551dc597f24846003f27488159add5fca9f39
Author: Maggie <mlm2299@columbia.edu>
Date:   Fri Mar 3 19:29:13 2017 -0500

    AST updated and now compiles

commit 41c57790e4aca6e9ff9ca7f19ca195329e6e7b53
Author: Maggie <mlm2299@columbia.edu>
Date:   Fri Mar 3 19:13:54 2017 -0500

    Updated ast to match grammar, first full try

commit bead446aabb9f16c522e535a4fe5b3cf9df92407
```

```
Author: Maggie <mlm2299@columbia.edu>
Date:  Fri Mar 3 18:35:47 2017 -0500

    Updated ast for expr

commit 7123deb0b7a7054b523eaf44c5b7826f5042d08a
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Mon Feb 27 18:51:31 2017 -0500

    Remove most lval productions in favor of handling during semantic checking

commit fc184be9db2d76cf59559a42e50486d2d1a58472
Author: Zack Silber <zs2266@columbia.edu>
Date:  Mon Feb 27 18:45:04 2017 -0500

    Statement lists can not be empty anymore, must write null statement
        (semicolon) instead

commit 045fca6f34b3991eed6663183753afc47b7944f7
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Mon Feb 27 18:17:11 2017 -0500

    Remove duplicate pow production

commit 981e5e04b3ff0dc71fb654e5159be6934f751055
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Mon Feb 27 13:59:40 2017 -0500

    It compiles, ofc w conflicts

commit e150c4273c61f8d6c64566d1a06b57e6481c6908
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Mon Feb 27 13:56:59 2017 -0500

    update pointer typ mapping

commit 0415f7dfd172e30613c47dc199653c7c658dc23c
Author: Zack Silber <zs2266@columbia.edu>
Date:  Sat Feb 25 17:57:43 2017 -0500

    Adding AST

commit 8e88ba11e7d6b75a837448c7f7bb09c118c2b998
Author: Maggie <mlm2299@columbia.edu>
Date:  Sat Feb 25 17:42:22 2017 -0500

    First go at the ocamlyacc CFG

commit a251ed1a581a53b65410882ace03c9aaf35cfbb8
Author: Joshua Zweig <jmz2135@columbia.edu>
Date:  Sat Feb 25 15:32:00 2017 -0500

    First commit
```

Chapter 5

System Architecture and Design

5.1 The Compiler



5.1.1 Scanner

The scanner reads in a stream of characters and turns them into tokens. Our scanner supports both C-style block comments with `/* */` and line comments with . Our scanner furthermore allows for declarations of string literals with `""` as well as char literals with ’’, with both constructs supporting various escape characters, namely

```
\\, \", \', \n, \r, \t, \0
```

5.1.2 Parser

The parser takes the stream of tokens returned by the scanner and translates it into a tree. On the top level, the parser sees functions and global variables. Within each function, the parser sees a list of variable declarations and a list of statements, where a statement may be a block in its own right. Indeed, our parser is a little quirky: instead of having a single stream of statements, we have two streams which separate out the variable declarations from other types of statements. The same logic applies when one is in a block nested inside

of a function. This is simply an artifact from the MicroC architecture which only allowed the user to declare a list of variable declarations followed by a list of typical statements. The resulting effects of this implementation are non-trivial: the parser cannot differentiate the relative order between variable declaration statements and other statements. As a concrete example, the parser cannot differentiate between the intuitively reasonable code

```
int main() {
    int x;
    int y;
    x = y = 3;
    return 0;
}
```

and the traditionally absurd

```
int main() {
   x = y = 3;
   int x;
   int y;
    return 0;
}
```

Of course, there is nothing *inherently* contradictory about the second block of code – it just defies the expectation that code is read in order, top to bottom. Indeed, as will be discussed in the Code Generator section, the statements *are* indeed read top to bottom, with the caveat that all variables in that block are declared before anything else. This distinction ends up not being very harmful, since only in the most pathological cases would one want to specifically delay the declaration of a variable if it is eventually going to be declared in that scope anyway.

5.1.3 Semantic Checker

The semantic checker walks through the OCaml AST returned by the parser, checking that the expressions and statements present make sense. Indeed, it walks through the AST in a depth-first fashion which passes a symbol table which gets updated as local variables fall in and out of scope.

The high level structure of the checker is as follows: it adds the global variables to the symbol table and then checks each function present individually. In each check, it walks through each statement one at a time to verify that they make sense. This can mean anything from checking that types match up (e.g., an integer is being added to an integer and not to, say, a function name) to checking that the variables being used actually exist in that scope. If something goes wrong, it prints a helpful error message that is specific to the context that the error is found. For example, one of our tests called `fail-func8.cm`:

```
void foo(int a, int b)
{
}

void bar()
{
}

int main()
{
   foo(42, 1);
   foo(42, bar()); /* int and void, not int and int */
}
```

gives the error

```
Fatal error: exception Failure("illegal actual argument found void
expected int in foo(42, bar())")
```

5.1.4 Preprocessor

The preprocessor is very basic and is only meant to support file inclusion, including build guards. It is written in python. The script just checks for a `#` at the beginning of each line, rigidly expecting to match one of the following patterns

```
#include "file_name"
#define VAR_NAME
#ifndef VAR_NAME
#ifdef VAR_NAME
#endif
```

With the effects being just like that in C. In particular, the file given by `file_name` must be present in the same directory of execution (our compiler has no `-I` flag to check other directories). Also, notice that the `#define` keyword is not used for textual substitution of constants (e.g. `#define BLOCK_SIZE 4096` is not supported), but only so that the build guards can function. As an example, a typical header file `myadd.h` could be something like

```
#ifndef __MYADD_H__
#define __MYADD_H__

int myadd(int a, int b) {
    return a + b;
}

#endif
```

and thus a main file defined as so

```
#include "myadd.h"

int main() {
    int x;
    x = myadd(5, 7);
    printf("%d\n", x);
    return 0;
}
```

will worked as expected when interpreted as a C% program.

5.1.5 Code Generator

The code generator takes in the OCaml AST and translates it to an LLVM AST. This is accomplished by mapping between expressions, control flow blocks and other aspects of the C% AST to an LLVM AST using the LLVM OCaml module. The mappings are the subject of each subsection to follow.

Definitions and Declarations

Our language supports both global and local variables. However, function prototypes such as

```
int do_something_later(char *x, void *y, int z);
```

is not valid. Thus, the only way to declare a function is to also implement it.

Expressions

The expressions component of C% is at the heart of what makes the language unique and valuable. The generation of expressions is what allows the language user to write `x + y` over two `int` types or `s + t` for two arguments of type `stone` and have the correct operation performed. OCaml's `match` statements were leveraged heavily to translate the expressions to match on types to make decisions on which function to call for a given expression.

Furthermore, the values of many expressions were not computable by the functions in the standard LLVM OCaml module. For this reason, we wrapped many C functions that were called to compute the values of expressions for operations on types `stone`, `mint`, `curve`, `point`, which will be the subject of Section 5.2.2. Most of these features were implemented by Josh, with `curve` and `point` functionality implemented by Michael.

Scoping

The scoping in our language is static. In particular, our language supports a subset of scoping that is allowed in C: it supports local variables in any block scope as well as global variables. However, it does not support `static` or `extern` variables in the sense of C. Also, due to the quirk of our parser (cf. the example code given in that section), a variable can only exist for the entire duration of a block, since all variables in a block are declared at once. As a concrete example, we have:

```
int main() {
    //x, y visible
    int x;
    for (x = 0; x < 5; x = x + 1) {
        //x, y, z visiible
        int z;
    }
    //x, y visible
    int y;
}
```

Control Flow

The control flow in our language is exactly how one would expect from C. The main thing our language is missing is `break` and `continue`.

5.2 Supplementary Code

5.2.1 Cryptography Library

Our languages main function in cryptography so we included programs writting in C% for our users. We include examples that allow the user to securely exchange keys, translate strings via a simple cipher, and generate primes. These programs serve to demonstrate the capabilities of our language, and give our users base programs to modify for their needs.

These functions can all be compiled as any other C% program or included in programs that our users write themselves. Our python preprocessor can then integrate any code that they want to use from our cryptography library.

5.2.2 Big-Num Integration and Memory Management

These components were primarily implemented by Josh with support from Michael.

Big-Num Integration

Underlaying all of the types and operations in C% that depend on arbitrary size arithmetic (a key feature needed for any serious cryptographic application) is the openSSL `bn` (for big number) library. The openssl suite and especially its API is quite uniformly regarded as terrible [**?**], and the `bn` library is certainly no exception. However, it is used quite commonly as it is better than most of the alternatives out there. Thus, allowing programmers to write C-like code while being able to apply traditional operators to arbitrarily large numbers is a major contribution of C%.

Below is an excerpt from `bn.h`

```
int BN_add(BIGNUM *r, const BIGNUM *a, const BIGNUM *b);
int BN_mul(BIGNUM *r, const BIGNUM *a, const BIGNUM *b, BN_CTX *ctx);
```

Any useful operation in this library requires the programmer to pass in a pointer to store the result of the operation in, as the operation is not performed in place. To leverage the `bn` library required to further layers of indirection. First, the library function were wrapped in a C function. For example, we include here the C wrapper for adding two values of type `stone`:

```
void* stone_add_func(void *a, void *b)
{
  BIGNUM *r = BN_new();

  BN_add(r, a, b);
  return r;
}
```

All of these such functions appear in `special_arith.c`. Notice here we much create *on the heap* a place for the result to be stored and returned. We must do this since we allow the user to simply add two stones with the addition operator, and thus they have no ability to pass in a location to store the result. Taking care of the allocation for the result is easy, but we must also automatically free these results as well when they are intermediate results and the user does not have a pointer to them, which is harder and will be the subject of the discussion in Section 5.2.2.

We then define our functions in `special_arith.c` functions in our code generator so that it can call them as needed. Carrying on with the example the definition in `codegen.ml` for the function to add arguments of type `stone` is included here:

```
let stone_add_func_t = L.function_type obj_pointer [| obj_pointer ; obj_pointer
    |] in
let stone_add_func = L.declare_function "stone_add_func" stone_add_func_t
    the_module in
```

Through nested type matching on the types of the arguments then on the operator being applied we determine which of the many of such functions to call. Further, for this all to come together, we must link the `.s` file generated with the `special_arith.o` as well as `libcrypto`, which is where the arbitrary size arithmetic functions live.

To date, we have implemented the following support for operations on data of type `stone`, `mint`, `point` and `curve`. Implementing more is trivial, and we intend to, but these are what time would allow.

```
  (* Add two mints *)
```

```
let mint_add_func = L.declare_function "mint_add_func" mint_add_func_t
    the_module

(* Subtract two mints *)
let mint_sub_func = L.declare_function "mint_sub_func" mint_sub_func_t
    the_module

(* Multiply two mints *)
let mint_mult_func = L.declare_function "mint_mult_func" mint_mult_func_t
    the_module

(* Raise a mint to a mint *)
let mint_pow_func = L.declare_function "mint_pow_func" mint_pow_func_t
    the_module

(* Raise a mint to a stone *)
let mint_to_stone_func = L.declare_function "mint_to_stone_func"
    mint_to_stone_func_t the_module

(* Add two stones *)
let stone_add_func = L.declare_function "stone_add_func" stone_add_func_t
    the_module

(* Subtract two stones *)
let stone_sub_func = L.declare_function "stone_sub_func" stone_sub_func_t
    the_module

(* Multiply two stones *)
let stone_mult_func = L.declare_function "stone_mult_func" stone_mult_func_t
    the_module

(* Divide two stones *)
let stone_div_func = L.declare_function "stone_div_func" stone_div_func_t
    the_module

(* Raise a stone to a stone *)
let stone_pow_func = L.declare_function "stone_pow_func" stone_pow_func_t
    the_module

(* Compute stone mod a stone *)
let stone_mod_func = L.declare_function "stone_mod_func" stone_mod_func_t
    the_module

(* Add two points *)
let point_add_func = L.declare_function "point_add_func" point_add_func_t
    the_module

(* Subtract two points *)
let point_sub_func = L.declare_function "point_sub_func" point_sub_func_t
    the_module

(* Multiply two points *)
let point_mult_func = L.declare_function "point_mult_func" point_mult_func_t
    the_module

(* Create a stone *)
let stone_create_func = L.declare_function "stone_create_func"
    stone_create_func_t the_module
```

```
(* Free a stone *)
let stone_free_func = L.declare_function "stone_free_func" stone_free_t
    the_module
```

Big-Num Memory Management

As mentioned in the previous section, allocating memory for the user on the heap is easy. Knowing when to free it is much harder. For example, consider the program below.

```
stone x;
stone y;
stone z;

x = "10";
y = "15";
z = "20";

z = x + y + z;
```

The computation of `z` here happens in the following order.

1. Compute `x + y`. Store result in `temp1`. (Note the user has no way to reach temp1).
2. Compute `temp1 + z`. Store result in `temp2`. (Note the user will have access to `temp2` because it should become the new value of `z`).

The fact that there must be a heap allocation for each of these intermediate results in this library is one of its major hindrances. Indeed, the library "fixes" this issue by making the user always pass in the pointer to the result as an argument, and then the function stores its result in this allocated memory. However, we want the user to be able to say something intuitive like `x+y+z`, not `BN_add(z, y, z); BN_add(z, x, z);`, and so we had to be clever to make sure our programs didn't leak large amounts of memory due to intermediate computations.

Our memory management strategy here leverages the computation tree conception of an arithmetic expression. In the tree, we know that the user must have a way to identify each leaf (explicit operand), but will have no way to access the internal nodes of the tree, with the exception of the root. To implement this strategy, we relay on the same method we do for type annotating the AST. Specifically, each expression is represented internally as a tuple of (*expression value*, *(type, isID)*). Thus, the first element stores the value of the expression and the second is a tuple of information, such as whether or not we have a handle to that expression (which in the case of an intermediate result we do not).

Every time an operation is done, we check if the program has a symbol for the expression in this way. When it does not, we conduct the necessary operation using the intermediate result and free it immediately after. This paradigm allows us to free the result openSSL forces us to store as soon as its value is no longer needed.

5.2.3 Built-in Functions

In the same way that Big-Num functions were linked to implicitly for the user, C% also provides several built in functions for users to leverage in writing programs. The implementation follows very much the same structure as described in the previous section, the only difference being that some of the functions are not wrapped in C functions and are instead linked directly to (i.e. those that are built into the OS). The built in functions fall into two categories, access, memory management and I/O.

The access functions, allowing the user access to the `stone`s underlying `mint`s, `curve`s, and `points`, were written in C and linked in similarly to the operator functions described above. Though originally intended to be in the language as a unary operator, we ultimately decided to create a separate sequence of functions to provide index by index access.

For memory management, we provide the user with malloc and free to manage the lifetime of stack variables. These functions are provided so the user can explicitly call `malloc()` and `free()` in their program. These will never have to be used for our special built in types as per the previous section, as we manage the memory automatically.

We additionally provide a robustly featured `printf` and `scanf`. In addition to their traditional uses, they can be levered in C%, as they are in our Diffie Helman Key Exchange for networked programs to pass communications in a cryptographic handshake between one and other. We additionally provide the function `print_stone()` for this application.

Chapter 6

Test Plan

6.1 Testing Phases

6.1.1 Grammar Testing

In the beginning stages of the project we needed to make sure that our grammar was functioning as intended. We wrote a program called scannerprint.mll that takes in source code written in our language and outputs the list of tokens that corresponds to the source code. This lexical analysis also removes comments, tabs, and whitespace and is demonstrated below.

```
// multPrec.cm
int main() {
    int x;

    x = 1 * 2 + 3 * 4;
}
```

```
INT ID LPAREN RPAREN LBRACE INT ID SEMI ID ASSIGN LITERAL STAR LITERAL PLUS
    LITERAL STAR LITERAL SEMI RBRACE EOF
```

Once we had a list of tokens we were able to pipe that into menhir with options set to show us the entire CST based off of our parser.mly. Menhir would produce a CST formatted as below. Once we had this CST we ran a diff between Menhir's CST and the CST that we expected and went over by hand. If there were any differences then the test would fail and we knew that something was wrong with our grammar.

```
ACCEPT
[program:
  [decls:
    [decls:]
    [fdecl:
      [typ: INT]
      ID
      LPAREN
      [formals_opt:]
      RPAREN
      LBRACE
      [combined_list:
        [combined_list: [combined_list:] [vdecl: [typ: INT] ID SEMI]]
        [stmt:
          [expr:
            ID
```

```
          ASSIGN
          [expr:
            [expr: [expr: LITERAL] STAR [expr: LITERAL]]
            PLUS
            [expr: [expr: LITERAL] STAR [expr: LITERAL]]
          ]
        ]
        SEMI
      ]
    ]
    RBRACE
  ]
 ]
 EOF
]
```

All of our grammar tests are located in one folder along with their expected outputs. There is a make command, test_grammar that will execute a python script, testAllPretty.py, written to generate the CST for every test and compare them to their corresponding .out files. If there are any differences then the python script will print them all out and return a fail value which was essential for our continuous integration that will be explained later.

After the grammar was ironed out this type of testing became less important. However, there were multiple times during the development of our project that changes would affect the grammar and the continuous execution of these tests prevented us from pushing changes that would have made our grammar function differently than intended. These tests also allowed us to preserve the stability of our grammar when adding new tokens.

Integration Testing

The rest of our tests would all be considered integration tests. Unit tests were not used because we were focused on adding new features and unit tests would not be reliable to see if the entire feature was functioning as expected. We would often look through certain layers of our compiler like making sure that semant would catch the right errors during development, but every test in our test suite checked the ultimate functionality of each feature.

We extended the testing workflow from MicroC. There were two types of tests, passing and failing tests. Passing tests were written in C% and had a corresponding expected output file. Failing tests, similarly, had to have source code along with a corresponding expected error file. These types of tests allowed us to not only make sure that our features exhibited expected behavior, but also to make sure that the right errors were being thrown when they were not used as intended.

We execute the source code of all of our tests and compare their outputs (via printf) or errors to the expected ones that we also have in the same testing folder. The expected workflow when building new features is to write some programs that would work (or fail properly) if the features were implemented correctly. Initially, these tests would not pass. However, once the feature is completely ironed out, all of the tests should pass and the feature can be considered working. Having all of these tests then becomes important moving forward because if any future updates break the feature then the tests will start failing again. Every pull request to master was expected to have passing tests for everything that was implemented.

Testing cases were chosen to fully flesh out every new feature that we added. Every use case that we could think of was added and every function had to have tests for it. Additionally, because our language was built on top of MicroC we started off our test suite

by slightly modifying all of their tests to work in our language.

Test suite

We put a lot of commands in our Makefile to allow for easy testing. Once you run 'make all' you could then run 'make test_grammar' or 'make test_compiler_travis'. To run the integration tests you simply had to input './testall.sh' , but when TravisCI ran the script there were additional flags that had to be set which made it more convenient to build a specific Makefile command.

We had two main testing scripts, a python script that ran all of the grammar testing (testAllPretty.py) and a shell script (testall.sh) that was modified from the original MicroC file that ran all of our compiler integration tests. testall.sh was ran as described above and output a list of every test along with whether they passed or failed.

```
-n test-add1...
OK
-n test-arith1...
OK
-n test-arith2...
OK
-n test-arith3...
OK
-n test-fib...
OK
-n test-for1...
OK
-n test-for2...
OK
//Continues for every test in testing folder
```

The shell script returns a passing value only if every single test passes. It is important to note that there is a small handful of tests that this script does not run. There are some tests (tests/custom_tests) that are dependent on manual input (File I/O) and some that have more specific instructions to run (Preprocessing, import statements). These tests could be integrated into the main testing suite, but would require further modification of the shell script which time did not allow. For now these tests are periodically run by hand in order to ensure their correct functionality.

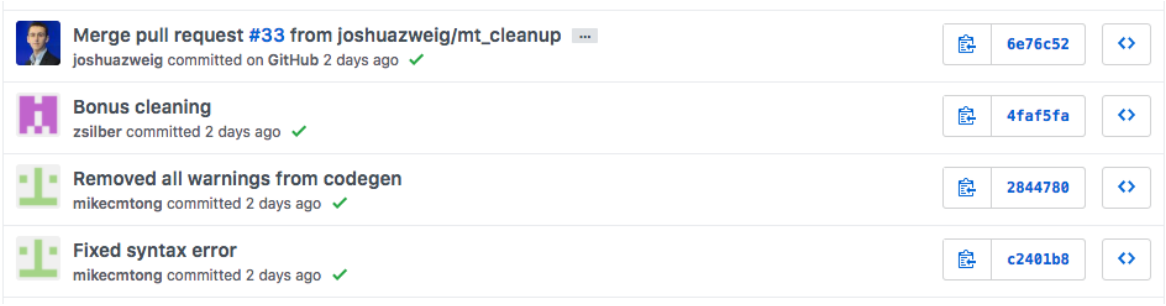
Automation/Continuous Integration

We used TravisCI for automated testing integration. Every time we either push to any of our branches or submit a pull request travis will run our entire testing suite and make sure that everything passes. We made our github repository public and gave TravisCI access to it. We also included a .travis.yml file in our source code in order to make sure that Travis knows how to run our tests.

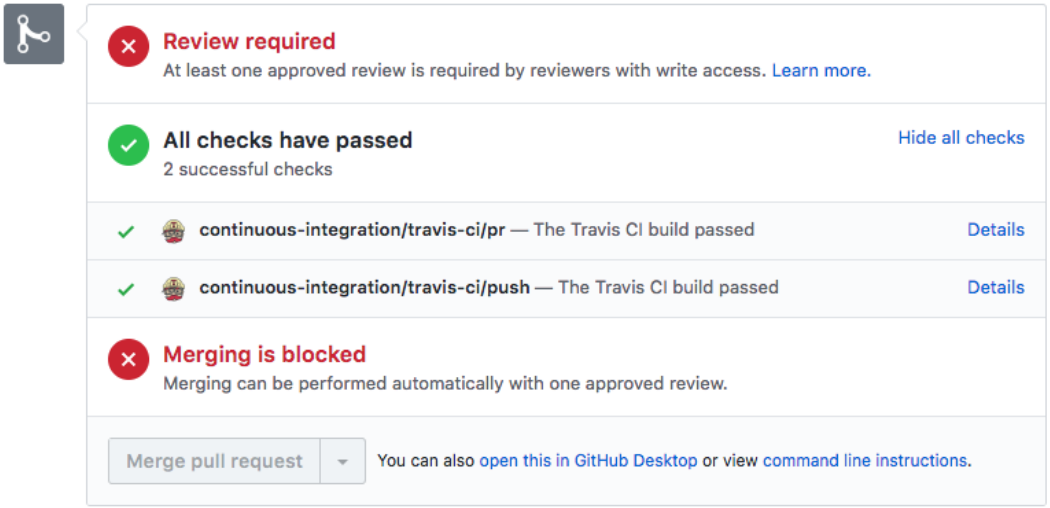
Every time we push or PR Travis spins up a virtual machine running Ubuntu and checks our .travis.yml file for instructions. The virtual machine proceeds to download all of the required dependencies for our program (opam, llvm, etc.). Then it uses our source code and runs makefile commands to check our entire testing suite. We can go to their website at any time to check the status of every build we have ever ran and see exactly which parts of our testing suite passed or failed.

```
724 test-mint-arith4...OK
725 test-mint-arith5...OK
726 test-mint-arith6...FAILED
727   ./cmod.sh failed on ./cmod.sh -v tests/compiler_tests/test-mint-arith6.cm > test-mint-arith6.out
728 test-mint-decl1...OK
729 test-mint-decl2...OK
```

Another very important aspect of Travis is that it reports whether our testing passed or failed directly on our github repository. Every commit has a symbol on it that indicates whether it passed our testing.



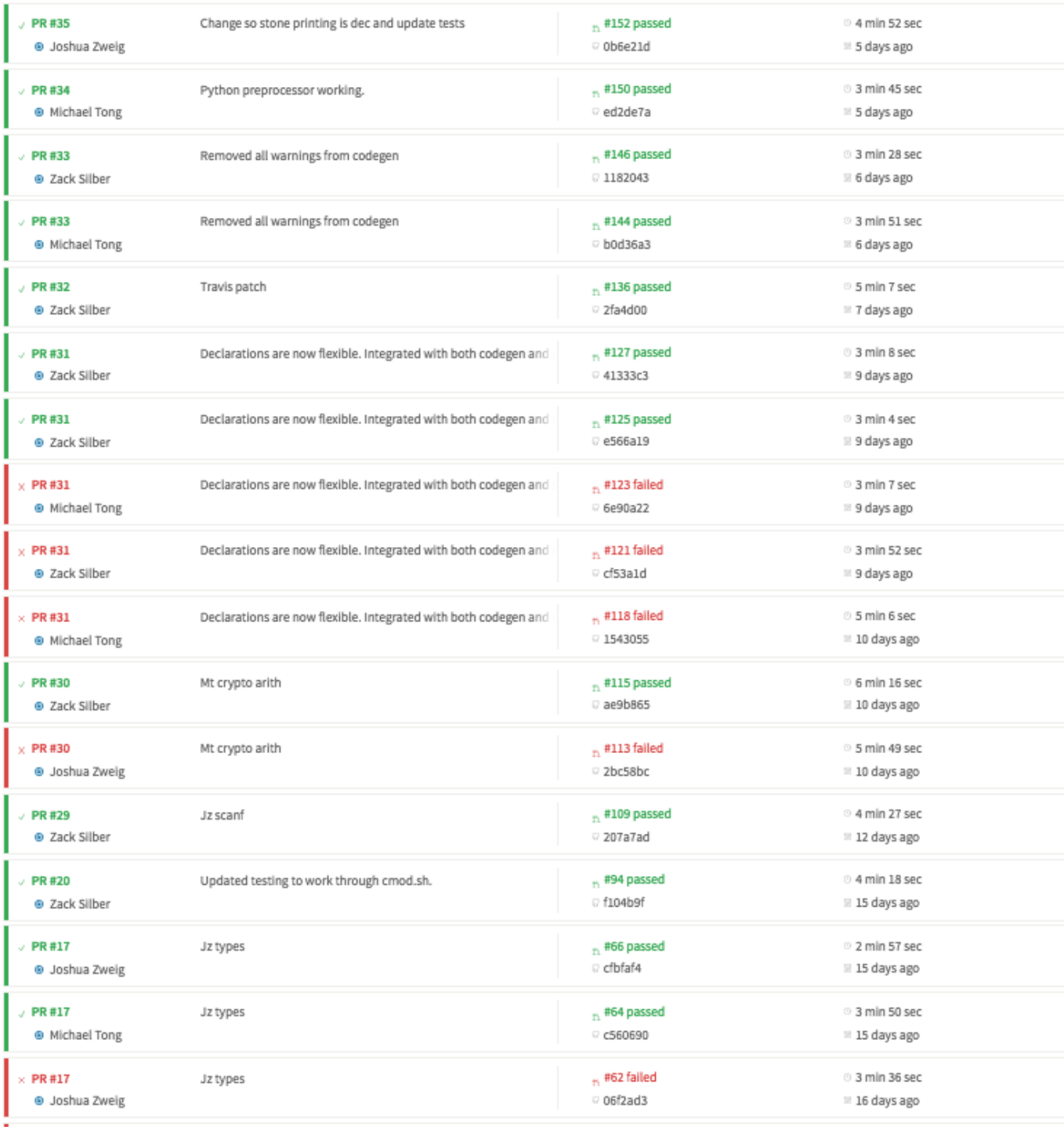
Lastly, it became very important in our development process that no broken branches were ever merged with master. Travis would not allow us to pull request any branches that would have made our test cases fail which allowed us to maintain a very stable master branch.



6.1.2 TravisCI Performance

Our builds run in on average 3-5 minutes. This varies drastically with the regular 9-5 hours of the workday because of our dependance on Travis' servers. There are opportunities to speed this up, we could have utilized Travis to cache some of our dependencies. However, this build time worked well enough for us.

It is hard to get exact data on how often faulty pull requests were fixed because of catches made by Travis. However, after going through the pull request build log it seems that roughly one third of PR's fail at least one test when they are initially uploaded. Furthermore, many errors are caught when viewing the Travis build status of git pushes which leads them to be fixed before the PR is submitted.



✓ PR #35 Joshua Zweig	Change so stone printing is dec and update tests	#152 passed 0b6e21d	4 min 52 sec 5 days ago
✓ PR #34 Michael Tong	Python preprocessor working.	#150 passed ed2de7a	3 min 45 sec 5 days ago
✓ PR #33 Zack Silber	Removed all warnings from codegen	#146 passed 1182043	3 min 28 sec 6 days ago
✓ PR #33 Michael Tong	Removed all warnings from codegen	#144 passed b0d36a3	3 min 51 sec 6 days ago
✓ PR #32 Zack Silber	Travis patch	#136 passed 2fa4d00	5 min 7 sec 7 days ago
✓ PR #31 Zack Silber	Declarations are now flexible. Integrated with both codegen and	#127 passed 41333c3	3 min 8 sec 9 days ago
✓ PR #31 Zack Silber	Declarations are now flexible. Integrated with both codegen and	#125 passed e566a19	3 min 4 sec 9 days ago
✗ PR #31 Michael Tong	Declarations are now flexible. Integrated with both codegen and	#123 failed 6e90a22	3 min 7 sec 9 days ago
✗ PR #31 Zack Silber	Declarations are now flexible. Integrated with both codegen and	#121 failed cf53a1d	3 min 52 sec 9 days ago
✗ PR #31 Michael Tong	Declarations are now flexible. Integrated with both codegen and	#118 failed 1543055	5 min 6 sec 10 days ago
✓ PR #30 Zack Silber	Mt crypto arith	#115 passed ae9b865	6 min 16 sec 10 days ago
✗ PR #30 Joshua Zweig	Mt crypto arith	#113 failed 2bc58bc	5 min 49 sec 10 days ago
✓ PR #29 Zack Silber	Jz scanf	#109 passed 207a7ad	4 min 27 sec 12 days ago
✓ PR #20 Zack Silber	Updated testing to work through cmod.sh.	#94 passed f104b9f	4 min 18 sec 15 days ago
✓ PR #17 Joshua Zweig	Jz types	#66 passed cfbfaf4	2 min 57 sec 15 days ago
✓ PR #17 Michael Tong	Jz types	#64 passed c560690	3 min 50 sec 15 days ago
✗ PR #17 Joshua Zweig	Jz types	#62 failed 06f2ad3	3 min 36 sec 16 days ago

6.2 C% to LLVM IR

6.2.1 Example 1

C%

```
int main() {
    printf("%s", "Hello World!\n");
    return 0;
}
```

LLVM

```
; ModuleID = 'Cmod'

@fmt2 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmts = private unnamed_addr constant [14 x i8] c"Hello World!\0A\00"
@fmts.1 = private unnamed_addr constant [3 x i8] c"%s\00"

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

declare i32 @scanf(i8*, i8*)

declare i8* @malloc(i32)

declare void @free(i8*)

declare { i8*, i8* } @mint_add_func({ i8*, i8* }*, { i8*, i8* }*)

declare { i8*, i8* } @mint_sub_func({ i8*, i8* }*, { i8*, i8* }*)

declare { i8*, i8* } @mint_mult_func({ i8*, i8* }*, { i8*, i8* }*)

declare { i8*, i8* } @mint_pow_func({ i8*, i8* }*, { i8*, i8* }*)

declare { i8*, i8* } @mint_to_stone_func({ i8*, i8* }*, i8*)

declare i8* @stone_add_func(i8*, i8*)

declare i8* @stone_sub_func(i8*, i8*)

declare i8* @stone_mult_func(i8*, i8*)

declare i8* @stone_div_func(i8*, i8*)

declare i8* @stone_pow_func(i8*, i8*)

declare i8* @stone_mod_func(i8*, i8*)

declare i32 @stone_print_func(i8*)

declare i32 @mint_print_func({ i8*, i8* })

declare i8* @point_add_func(i8*, i8*)

declare i8* @point_sub_func(i8*, i8*)

declare i8* @point_mult_func(i8*, i8*)

declare i8* @stone_create_func(i8*)

declare i32 @stone_free_func(i8*)

define i32 @main() {
entry:
  %0 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x
      i8]* @fmts.1, i32 0, i32 0), i8* getelementptr inbounds ([14 x i8], [14 x
      i8]* @fmts, i32 0, i32 0))
  ret i32 0
```

```
}
```

6.2.2 Example 2

C%

```
int main()
{
  stone x;
  stone p;

  mint m;

  x = "15352395";
  p = "65537";

  m = <x + x, p>;

  print_mint(m);

  return 0;
}
```

LLVM

```
; ModuleID = 'Cmod'

@fmt2 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmts = private unnamed_addr constant [9 x i8] c"15352395\00"
@fmts.1 = private unnamed_addr constant [6 x i8] c"65537\00"

// built in function declarations...

define i32 @main() {
entry:
  %x = alloca i8*
  %p = alloca i8*
  %m = alloca { i8*, i8* }
  %stone_create_func = call i8* @stone_create_func(i8* getelementptr inbounds
      ([9 x i8], [9 x i8]* @fmts, i32 0, i32 0))
  store i8* %stone_create_func, i8** %x
  %stone_create_func1 = call i8* @stone_create_func(i8* getelementptr inbounds
      ([6 x i8], [6 x i8]* @fmts.1, i32 0, i32 0))
  store i8* %stone_create_func1, i8** %p
  %x2 = load i8*, i8** %x
  %x3 = load i8*, i8** %x
  %stone_add_res = call i8* @stone_add_func(i8* %x2, i8* %x3)
  %p4 = load i8*, i8** %p
  %stone_mod_res = call i8* @stone_mod_func(i8* %stone_add_res, i8* %p4)
  %sm = insertvalue { i8*, i8* } undef, i8* %stone_mod_res, 0
  %sm2 = insertvalue { i8*, i8* } %sm, i8* %p4, 1
  store { i8*, i8* } %sm2, { i8*, i8* }* %m
  %m5 = load { i8*, i8* }, { i8*, i8* }* %m
  %mint_print_func = call i32 @mint_print_func({ i8*, i8* } %m5)
  ret i32 0
}
```

6.2.3 Example 3

C%

```
int add(int a, int b)
{
  return a + b;
}

int main()
{
  int a;
  a = add(39, 3);
  printf("%d", a);
  return 0;
}
```

LLVM

```
; ModuleID = 'Cmod'

@fmt2 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmts = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt2.1 = private unnamed_addr constant [3 x i8] c"%s\00"

//built in function declarations...

define i32 @main() {
entry:
  %a = alloca i32
  %add_result = call i32 @add(i32 39, i32 3)
  store i32 %add_result, i32* %a
  %a1 = load i32, i32* %a
  %0 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x
      i8]* @fmts, i32 0, i32 0), i32 %a1)
  ret i32 0
}

define i32 @add(i32 %a, i32 %b) {
entry:
  %a1 = alloca i32
  store i32 %a, i32* %a1
  %b2 = alloca i32
  store i32 %b, i32* %b2
  %a3 = load i32, i32* %a1
  %b4 = load i32, i32* %b2
  %tmp = add i32 %a3, %b4
  ret i32 %tmp
}
```

Chapter 7

Lessons Learned

7.0.1 Zack

Since my main job was testing and general QA, I learned a lot about how to maintain a large project and keep it stable under the revisions of multiple contributors. The value of continuous integration was really proven to me. However, I definitely wish that I had made sure that my team was more on the same page in regards to the structure of our files and how to run and write tests. When I looked at the development work-flow of my teammates I realized that they weren't taken advantage of a lot of the testing features I had implemented, especially in the beginning. The responsibility to on-board them and teach them how to use everything definitely should have fallen on me. I learned that I should have been in closer collaboration with everyone's work-flow and made sure that they were taking advantage of everything instead of assuming that they would figure it out themselves. This also would have gone a long way in increasing my own efficiency because I would often tidy up PR's in order to maintain order.

More generally, I really learned about the value of overestimating how long certain tasks will take and starting everything early. It was really amazing how much work our team could produce while functioning as a unit, but the project itself is huge and definitely expects that level of output.

7.0.2 Michael

I think one of the biggest lessons I learned from a software engineering perspective was the importance of truly understanding legacy code before going ahead and trying to make any changes. Indeed, since our language was C-like we based a lot of our initial changes off of the MicroC architecture provided by Professor Edwards, which meant that there were several hundred lines of OCaml code that we needed to understand before even beginning to add elements of our language. This was no easy task, and even the most trivial of changes early on (e.g. incorporating the built-in function "printf" for the Hello World demo) seemed difficult since we just didn't know what one had to do to get it working. That said, by the end of the project, after having implemented many new features into the semantic checker and code generator, whenever we needed to do something new, we were so in tune with the overall structure of our compiler that it wasn't a problem. In fact, this led to many great conversations about what the best way to implement certain features of our language were, which was definitely some of the most rewarding parts of this project for me.

I also learned the importance of testing (thanks Zack!) – I remember having made a large change to the structure of our language and thinking "how can I know this works?" The presence of Travis, as well as a local script which checked it for you, was immensely helpful for my sanity in these cases.

Though I wasn't the manager, I also learned some lessons in team management. When we started getting into the meat of the project, we assigned roles rather arbitrarily on a sort of "whatever you want" basis. While this worked out for some – Zack turned out to be a great tester and got the Travis integration working very quickly – others ended up struggling with their roles. It would have been much better for us to have reassigned roles once our strengths and weaknesses were clear so that everyone was being as efficient as possible. Indeed, by the end of the project, whenever something needed to be done, there was a clear point person who would get it done three times faster than anyone else would, and if we had done this throughout the course of the project we could have cut down on a lot of unproductive hours. In a similar vein, it would have been better to have recognized earlier when certain members of the team were falling behind and getting them up to speed so that we had more hands available and ready to work on the main parts of the compiler.

7.0.3 Josh

Over the course of the semester, I learned a lot technical things, a lot about project management and a lot about the intersection of the two.

From the technical side, I became much better at being able to dive into some API or chunk of code and really understand what it does, how it works and what its flaws are. A lot of the code we relayed on (e.g. LLVM OCaml module, openssl/bn) are not nearly as well documented as some of the APIs, modules, etc that I have used in the past. It felt it infinitely more satisfying to dive into and solve errors and bugs on my own than it does to seek answers on StackOverflow at every turn. The grit and skill I learned at this is something I will certainly carry with me into the future, and I think it will serve my wonderfully.

I also gained a thorough appreciation for the importance of testing and emphasizing it early and often. Zack did an awesome job setting up continuous integration, and it saved us on more than one occasion. It is quite humbling to see two random tests fail when you push changes that you thought you had nothing to do with. Our robust and automated testing suite allowed us to be confident whenever we merged a new PR. When I find myself in similar situations in the future, I will be sure to be even more aggressive about writing good tests early then we were in this project. They are, and proved to be, a really invaluable resource.

From the team management perspective, I learned a lot about some of the classic leadership challenges in software development. One thing we did a great job at was setting reasonably sized goals and tasks for each team member each week. Being disciplined about meeting these goals was at the foundation of our team's success. Not once did we pull an all nighter or find ourselves totally scrambling to meet a deadline, and of that I am extremely proud. We only really hit our stride with respect to productivity in the last month of the project and there were certainly some growing pains getting there. Looking back, there are a few things that stand out between how our team operates not, at high efficiency, and how it did before we goth here. One key thing is setting deliverables. At the beginning, we did not set clear deliverables for the next week, which resulted in team members coming to the meeting with something 'almost finished,' but was not quite so. After all, in software development there is the first 90% and the second 90%. To counter this, we started requiring PRs to be up before our weekly Monday meetings. Even so, some members of the team were not keeping pace, which was a big challenge. This was difficult to manage, but setting well defined and narrower aspects of the project for the week seemed to have some positive impact. Allocating human capital is hard. It will certainly remain hard, but my amazing experience completing an awesome project with this team has taught me an incredible amount.

7.0.4 Maggie

This project taught me an incredible amount about software engineering from a team perspective. It was the first sizable group programming project I've worked on, and it was initially quite tough for me to wrap my head around working with version control for multiple contributors. By now, I feel incredibly more comfortable understanding how to follow the growth of a project and keep track of who's changing what, etc., and I'm sure this will prove immensely useful to me going forward. The most important thing I learned there was probably to be more proactive in considering what everyone else was working on at any given time, as there were definitely times when I would implement some functionality only to realize that it had already been done or that my work depended on an outdated version of another piece of the project.

More than just the multiple contributors, this project had a lot of moving parts, and it was at times difficult to understand what needed to get done and when in order to stay on track. In the end, it was incredibly gratifying to gain understanding of how all the pieces fit together, and to be able to easily navigate the multiple files fitting together to isolate where an error was coming from. I think this slowed our group down a bit in the beginning, but as we worked on more pieces of the project we really hit our stride. We became a lot more productive once we began to set more concrete to-dos and began to see our project taking shape. We also, initially, had a lot of really big PRs, and I think we were much more productive once we began to push smaller and more specific changes, solving problems piece by piece instead of waiting for an entire section to be finished. Even so, I learned a lot about managing tasks between multiple people. There were times when I'd get stuck on something, and I'd let myself just stay stuck there, when I could've sought more help from teammates who were likely facing the same difficulties. Moreover, I think I realized late in the game how much control we had in the decisions we made (what to implement, what not to, etc.), and getting stuck on one thing should not have prevented me from working on other aspects of the project, even if it wasn't a pre-existing priority or if other members of my group were already working on them. I think Michael, for example, did a great job of identifying parts of the project that were missing and stepping up to implement them, as he did with scoping issues and declaration flexibility.

Finally, I learned a lot about the importance of testing. Integrated testing was a new concept for me entirely, and seeing it in action was very cool. It was hard enough to understand how all the parts of the project worked together, much less understanding how some small change would impact every other aspect of it. Putting up a PR and seeing the Travis build fail after a small, seemingly unrelated change was not only fascinating but also made clear how important integrated testing is to a large software project, allowing you to clearly isolate which changes introduce issues. Altogether, I had a fantastic group and learned an immense amount.

7.0.5 Richard

In this project, I learned the most about working on a software project in a group environment. Prior to this project, I hadn't worked with Github, and I was sometimes lost about how to parallelize my work alongside others. Funny enough, for the first few things I wrote, I would email them to Zack or Josh and that's how I worked for the first few weeks, which wasn't optimal whatsoever. I also was afraid to speak up sometimes and ask for help. There was a time this semester when I was stuck on integrating break and continue into our codegen for a few weeks, and every week I would claim that I was getting closer, but that I didn't need help. Because of this, there was a lot of wasted time and effort over implementing a fairly trivial portion of our code, which never got implemented in the end. It was humbling and it showed me that communication was necessary to working

effectively in a group. Over the course of this semester, I've learned to use group version control and to be a more communicative member of the team. I think I also learned a ton about the role of testing during this project. I think testing and QA are usually viewed as a role that's less desirable compared to roles that work on the projects directly. However, this project has shown me that testing is incredibly important, and is almost the backbone that the project relies on to make sure nothing breaks. Throughout the project, Zack emphasized for us to write our own tests as we developed code, and it made integration so much easier in the long run. Working with Travis was so helpful and it made everyone's lives so much easier.

Chapter 8

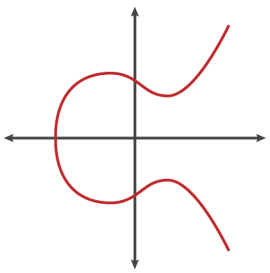
Appendix on Elliptic Curve Cryptography

8.1 Background and definitions

A typical elliptic curve is an equation

$$y^2 = x^3 + ax + b$$

where the variables in this equation take values in the real numbers $\mathbb{R}$. So one can visualize this as a set of points (x, y) in the plane for which this equation is true. Here is an example:



The curve $y^2 = x^3 - x + 1$

The elliptic curve has the following remarkable property: if a line passes through two points on the curve, then it passes through three points. Well, not exactly. The points of intersection are counted *with multiplicity*, which practically means that if a line is tangent to a curve at a point, then this intersection counts as two. Also, one might notice that in the picture, a vertical line only intersects the curve in two points. Vertical lines are said to intersect the curve *at infinity* – this sounds like cheating, but there is a (rather technical) way to have this make perfect sense (and our implementation deals with this behind the scenes). Furthermore, notice that if a point $P = (x, y)$ is on an elliptic curve then $(x, -y)$ is also on it, and we call this point $-P$.

Using these two facts, we have the following binary operation, called addition and denoted $+$: it takes two points P and Q as input. It finds the unique line passing through both points (if the points are the same, it takes the tangent line) and finds the third point on the line R. The operation then returns $-R$. In short, $P + Q = -R$.

It takes some work to check, but it turns out that this addition $+$ satisfies the properties stated above in §9.2 so that the points on an elliptic curve form a group.

8.2 Addition formula

The following formula describes the addition law mentioned above. Let E be an elliptic curve given by $y^2 = x^3 + ax + b$. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on E. The point $P_1 + P_2 = P_3 = (x_3, y_3)$ is defined as follows:

$$\begin{aligned} x_3 &= m^2 - x_1 - x_2 \\ y_3 &= m(x_1 - x_3) - y_1 \end{aligned}$$

where

$$m = \begin{cases} (y_2 - y_1)/(x_2 - x_1) & \text{if } P_1 \neq P_2 \\ (3x_1^2 + a)/(2y_1) & \textit{if } P_1 = P_2. \end{cases}$$

and if the denominator of the relevant m is zero, then the sum is the point infinity and we write $P_3 = \infty$.

8.3 Translation to cryptography

Now, in the above exposition we took elliptic curves defined over the real numbers, but it turns out that this can be defined over any algebraic object known as a *field* (simplified, this is a set with *two* binary operations $+$ and $\times$ where both operations enjoy the nice properties of a group and both operations distribute over each other, as they do in $\mathbb{R}$).

Furthermore, the set of integers when taken with a prime modulus p, equipped with the typical modular operations $+$ and $\times$, is a field and is typically denoted $\mathbb{F}_p$. Typically, elliptic curves in cryptography are defined over this object $\mathbb{F}_p$ instead of $\mathbb{R}$, so that a point a curve E is really an equation

$$y^2 \equiv x^3 + ax + b \pmod{p}$$

and the points on it are pairs of (modular) integers (x, y) for which this equation holds. Indeed, even the addition law is the same, except the formula for m "changes" (we put "changes" in quotation marks since this is really just a generalization of the above) to

$$m = \begin{cases} (y_2 - y_1)(x_2 - x_1)^{-1} & \text{if } P_1 \neq P_2 \\ (3x_1^2 + a)(2y_1)^{-1} & \textit{if } P_1 = P_2. \end{cases}$$

where x^{-1} denotes the *multiplicative inverse* of x in whatever field it is defined in. For example, the multiplicative inverse of 5 with respect to the modulus 7 is 3, since $5 \cdot 3 \equiv 1 \pmod{7}$, so we write $5^{-1} \equiv 3$. As above, the sum of two points is ∞ if the "denominator", i.e. the term being inverted, of the relevant m is equal to zero.

8.4 Comparison with modular arithmetic

The discussion in §9.2 motivates the analogy between the two cryptosystems. Indeed, while in modular arithmetic we use integers mod p as building blocks and modular multiplication as an operation relating these elements, in elliptic curve cryptography we use the points on an elliptic curve as the elements and use the aforementioned addition law to relate these elements together. With this analogy, one can translate many cryptographic protocols over the modular integers into an exact protocol over an elliptic curve.

Indeed, recall the discrete log problem from §9.1. The elliptic curve analogy is this: given a curve E and two points on it P and Q, can you find an integer k for which $kP = Q$? (Here kP denotes P added to itself k times). The analogy should be becoming clear.

Chapter 9

Code Listing

9.1 Compiler Source

9.1.1 Primary

Listing 9.1: Makefile

```
CC = gcc

.PHONY: cmod.native
cmod.native:
   ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis -cflags -w,+a-4 \
      cmod.native

.PHONY: test_grammar
test_grammar:
   ocamllex scannerprint.mll
   python tests/grammar_tests/testAllPretty.py

.PHONY: test_compiler_travis
test_compiler_travis:
   export LLI="/usr/lib/llvm-3.8/bin/lli"
   ./testall.sh -v

special_arith.o: special_arith.c
   clang -I/usr/local/opt/openssl/include -c special_arith.c

access.o: access.c
   clang -I/usr/local/opt/openssl/include -c access.c

access: access.o
   clang access.o -lcrypto -o access

cmc:
   mkdir bin
   cp cmc.sh ./bin/cmc
   chmod +x ./bin/cmc

.PHONY: clean
clean :
   ocamlbuild -clean
   rm -rf testall.log *.diff cmod scanner.ml parser.ml parser.mli
   rm -rf *.cmx *.cmi *.cmo *.cmx *.o
   rm -rf *.err *.ll *.diff *.out
```

```
   -rm -f scannerprint.ml *.tmp
   rm -f *.exe *.s
   rm -rf bin

.PHONY : all
all : clean cmod.native special_arith.o access.o cmc
```

Listing 9.2: cmod.ml

```
(* Top-level of the MicroC compiler: scan & parse the input,
   check the resulting AST, generate LLVM IR, and dump the module *)

type action = Ast | LLVM_IR | Compile

let _ =
  let action = if Array.length Sys.argv > 1 then
    List.assoc Sys.argv.(1) [ ("-a", Ast); (* Print the AST only *)
            ("-l", LLVM_IR); (* Generate LLVM, don't check *)
            ("-c", Compile) ] (* Generate, check LLVM IR *)
  else Compile in
  let lexbuf = Lexing.from_channel stdin in
  let ast = Parser.program Scanner.token lexbuf in
  Semant.check ast;
  match action with
    Ast -> print_string (Ast.string_of_program ast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate ast))
  | Compile -> let m = Codegen.translate ast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)
```

Listing 9.3: scanner.mll

```
  (* Ocamllex scanner for MicroC *)
{ open Parser
    module B = Buffer }

(* why are some string and some chars LT, GT eg *)

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"    { comment lexbuf }           (* Comments *)
| "//"    { comment2 lexbuf }
| '('     { LPAREN }
| ')'     { RPAREN }
| '{'     { LBRACE }
| '}'     { RBRACE }
| '['     { LSQUARE }
| ']'     { RSQUARE }
| ';'     { SEMI }
| ','     { COMMA }
| '+'     { PLUS }
| '-'     { MINUS }
| '*'     { STAR }
| '^'     { POW }
| '/'     { DIVIDE }
| '%'     { MOD }
| '&'     { ADDRESSOF }
```

```
| '='      { ASSIGN }
| '^'      { POW }
| "%="     { MODASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| '>'      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| '!'      { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "do"     { DO }
| "break"  { BREAK }
| "continue" { CONTINUE }
| "return" { RETURN }
| "int"    { INT }
| "void"   { VOID }
| "char"   { CHAR }
| "NULL"   { NULL }
| "stone"  { STONE }
| "mint"   { MINT }
| "point"  { POINT }
| "curve"  { CURVE }
| '~'      { INF }
| "access" { ACCESS }
| "'"      { CHARLIT (read_char lexbuf) }
| '"'      { STRING (build_str (B.create 100) lexbuf) }
| ['0'-'9']+ as lxm { LITERAL(int_of_string lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment = parse
  "*/" { token lexbuf }
| _    { comment lexbuf }

and comment2 = parse
  '\n' { token lexbuf }
| _    { comment2 lexbuf }

and build_str sb = parse
 | '"'      { B.contents sb }
 | '\\''\\' { B.add_char sb '\\'; build_str sb lexbuf }
 | '\\''"' { B.add_char sb '"'; build_str sb lexbuf }
 | '\\''\'' { B.add_char sb '\''; build_str sb lexbuf }
 | '\\''n' { B.add_char sb '\n'; build_str sb lexbuf }
 | '\\''r' { B.add_char sb '\r'; build_str sb lexbuf }
 | '\\''t' { B.add_char sb '\t'; build_str sb lexbuf }
 | _ as t  { B.add_char sb t;   build_str sb lexbuf }

 and read_char = parse
 | '\\''\\' { check_length '\'' lexbuf }
 | '\\''"' { check_length '"' lexbuf }
```

```
 | '\\''\'' { check_length '\'' lexbuf }
 | '\\''n' { check_length '\n' lexbuf }
 | '\\''r' { check_length '\r' lexbuf }
 | '\\''t' { check_length '\t' lexbuf }
 | '\\''0' { check_length (char_of_int 0) lexbuf } (* '\0' char in C *)
 (* this should only match characters of length 1, as expected *)
 | (_ as t) { check_length t lexbuf }


and check_length buf = parse
 | '\'' { buf }
 | _ as t { raise( Failure ("illegal char literal " ^ Char.escaped t)) }
```

Listing 9.4: parser.mly

```
/* Ocamlyacc parser for C%, after that for MicroC */

%{
open Ast
%}

%token SEMI COMMA LPAREN RPAREN LBRACE RBRACE LSQUARE RSQUARE
%token PLUS MINUS STAR DIVIDE MOD ASSIGN NOT POW ADDRESSOF /*NEG*/ /* minus is
    neg, star is times */
%token MODASSIGN /* star is deref*/
%token EQ NEQ LT LEQ GT GEQ AND OR
%token RETURN IF ELSE FOR WHILE DO BREAK CONTINUE
%token INT CHAR VOID NULL
%token STONE MINT CURVE POINT INF ACCESS
%token <int> LITERAL //need string literals
%token <string> ID
%token <string> STRING
%token <char> CHARLIT
%token EOF

//COMMA?
%nonassoc NOELSE
%nonassoc ELSE
%right MODASSIGN ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%right ACCESS
%left STAR DIVIDE MOD //star is times
%right POW
%right NOT NEG ADDRESSOF DEREF /* minus is neg, mod is addof, star is deref */

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

decls:
```

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

fdecl:
   typ ID LPAREN formals_opt RPAREN LBRACE combined_list RBRACE
     { { typ = $1;
     fname = $2;
     formals = $4;
     locals = List.rev (fst $7);
     body = List.rev (snd $7) } }

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

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

typ:
    INT { Int }
  | CHAR { Char }
  | VOID { Void }
  | STONE { Stone }
  | MINT { Mint }
  | CURVE { Curve }
  | POINT { Point }
  | typ STAR { Pointer($1) } // unclear if this is a proper declaration

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

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

stmt:
    expr SEMI { Expr $1 } /*expr_opt here instead of nullstmt maybe*/
  | RETURN SEMI { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
  | LBRACE combined_list RBRACE { Block(List.rev (fst $2), List.rev (snd $2)) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([], [])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr_opt SEMI expr_opt RPAREN stmt /* made expr2
      optional */
     { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }
  | DO stmt WHILE LPAREN expr RPAREN { DoWhile($2, $5) } /* ADDED */
  | BREAK SEMI { Break } /* ADDED */
  | CONTINUE SEMI { Continue } /* added */
  | SEMI { NullStmt } /* ADDED - unclear if could be Noexpr */

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

```
expr:
    LITERAL          { Literal($1) }
  | ID               { Id($1) }
  | INF              { Inf }
  | expr PLUS  expr { Binop($1, Add, $3) }
  | expr MINUS expr { Binop($1, Sub, $3) }
  | expr STAR  expr { Binop($1, Mult, $3) } //star is times
  | expr DIVIDE expr { Binop($1, Div, $3) }
  | expr POW   expr { Binop($1, Pow, $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 OR    expr { Binop($1, Or,  $3) }
  | MINUS expr %prec NEG { Unop(Neg, $2) } /* second minus is neg */
  | NOT expr         { Unop(Not, $2) }
  | ID ASSIGN expr { Assign($1, $3) } //changed ID to lval
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN { $2 }
  | NULL { Null } /* Added all after this line in expr */
  | STAR expr %prec DEREF  { Unop(Deref, $2) } // star is deref
  | ADDRESSOF expr { Unop(AddrOf, $2) } /* must be an lvalue, changed back to
      unop */
  | expr MOD expr { Binop($1, Mod, $3) }
  | ID MODASSIGN expr { ModAssign($1, $3) }
  | STRING { String($1) } /* string literal */
  | CHARLIT { Ch($1) } /* char literal */
  | LT expr COMMA expr GT { Construct2($2, $4) }
  | LT expr COMMA expr COMMA expr GT { Construct3($2, $4, $6) }
  | ID LSQUARE expr RSQUARE { Subscript($1, $3) }
  | ACCESS expr { Unop(Access, $2) }

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

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

Listing 9.5: ast.ml

```
(* Abstract Syntax Tree and functions for printing it *)

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

type uop = Neg | Not | Deref | AddrOf | Access

type typ = Int | Char | Stone | Mint | Curve | Point | Void | Pointer of typ

type bind = typ * string

type expr =
    Literal of int
```

```
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Construct2 of expr * expr
  | Construct3 of expr * expr * expr
  | Assign of string * expr
  | Call of string * expr list
  | Noexpr   (* not Null? *)
  | Null
  | ModAssign of string * expr
  | String of string
  | Ch of char (* Maybe change back to char *)
  | Subscript of string * expr
  | Inf

type stmt =
    Block of (bind list * stmt list)
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt (* need to account for optional exprs? *)
  | While of expr * stmt
  | DoWhile of stmt * expr
  | Break
  | Continue
  | NullStmt

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

type program = bind list * func_decl list

(* Pretty-printing functions *)

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

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

```
  | Deref -> "*"
  | AddrOf -> "&"
  | Access -> "access"

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""
  | Null -> "NULL" (* pointer to zero *)
  | Inf -> "Inf"
  | ModAssign(v, e) -> v ^ " %= " ^ string_of_expr e
  | String(s) -> s
  | Ch (c) -> String.make 1 c
  | Subscript(s, e) -> s ^ "[" ^ string_of_expr e ^ "]"
  | Construct2(e1, e2) -> "{" ^ string_of_expr e1 ^ ", " ^ string_of_expr e2 ^
      "}"
  | Construct3(e1, e2, e3) ->
      "{" ^ string_of_expr e1 ^ ", " ^ string_of_expr e2 ^ ", " ^ string_of_expr
          e3 ^ "}"

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

let rec string_of_typ = function
    Int -> "int"
  | Char -> "char"
  | Stone -> "stone"
  | Mint -> "mint"
  | Curve -> "curve"
  | Point -> "point"
  | Void -> "void"
  | Pointer (_ as t) -> "pointer " ^ string_of_typ(t)

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

let string_of_fdecl fdecl =
```

```
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

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

Listing 9.6: semant.ml

```
(* Semantic checking for the MicroC compiler *)

open Ast

module StringMap = Map.Make(String)

(* Semantic checking of a program. Returns void if successful,
   throws an exception if something is wrong.

   Check each global variable, then check each function *)


let check (globals, functions) =

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

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

  (* Raise an exception of the given rvalue type cannot be assigned to
     the given lvalue type *)
  let check_assign lvaluet rvaluet err =
     if lvaluet = rvaluet then lvaluet else raise err
  in

  (**** Checking Global Variables ****)

  List.iter (check_not_void (fun n -> "illegal void global " ^ n)) globals;

  report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd globals);

  (**** Checking Functions ****)

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

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

if List.mem "access_curve" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function access_curve may not be defined")) else ();

if List.mem "access_point" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function access_point may not be defined")) else ();

if List.mem "scanf" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function scanf may not be defined")) else ();

if List.mem "malloc" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function malloc may not be defined")) else ();

if List.mem "free" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function free may not be defined")) else ();

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

(* Function declaration for a named function *)
let built_in_decls = List.fold_left (fun map (name, attr) -> StringMap.add
name attr map) StringMap.empty [
     ("printf", { typ = Void; fname = "printf"; formals = [];
     (* change formals to be variadic? Right now, this is fixed by just not
     comparing formals and actuals list if the name of the function is printf *)
     locals = []; body = [] });
     ("atoi", { typ = Int; fname = "atoi"; formals = [(Pointer(Char),
     "x")]; locals = []; body = [] });
     ("print_stone", { typ = Int; fname = "print_stone"; formals = [(Stone,
     "x")]; locals = []; body = [] });

     ("access_mint", {typ = Stone; fname = "access_mint"; formals = [(Mint,
         "m"); (Int, "i")];
      locals = []; body = []});
     ("access_curve", {typ = Stone; fname = "access_curve"; formals =
         [(Pointer(Curve), "c"); (Int, "i")];
      locals = []; body = []});
     ("access_point", {typ = Stone; fname = "access_point"; formals =
         [(Pointer(Point), "p"); (Int, "i")];
      locals = []; body = []});

     ("print_mint", { typ = Int; fname = "print_mint"; formals = [(Mint,
     "x")]; locals = []; body = [] });
     ("print_div", { typ = Int; fname = "print_div"; formals = [(Mint,
     "x")]; locals = []; body = [] });
     ("print_point", { typ = Int; fname = "print_point"; formals =
         [(Pointer(Point), "P")]; locals = []; body = [] });
     ("print_point_sep", { typ = Int; fname = "print_point_sep"; formals =
         [(Pointer(Point), "P")]; locals = []; body = [] });
     ("print_curve", { typ = Int; fname = "print_curve"; formals =
         [(Pointer(Curve),
     "E")]; locals = []; body = [] });
     ("scanf", { typ = Void; fname = "scanf"; formals = [(Pointer(Char), "x")];
     locals = []; body = [] });
     ("malloc", { typ = Pointer(Char); fname = "malloc"; formals = [(Int, "x")];
```

```
    locals = []; body = [] });
    ("free", { typ = Void; fname = "free"; formals = [(Pointer(Char), "x")];
    locals = []; body = [] })
    ]
    (* Can only malloc char pointers, best way to generalize? *)

 in

let function_decls = List.fold_left (fun m fd -> StringMap.add fd.fname fd m)
                     built_in_decls functions
in

let function_decl s = try StringMap.find s function_decls
     with Not_found -> if s = "main" then raise (Failure ("main function must
         be defined"))
     else raise (Failure ("unrecognized function " ^ s))
in

let _ = function_decl "main" in (* Ensure "main" is defined *)
(* Note: This prints a weird error message in the case main isn't defined.
 * Maybe change it? (This is edwards' code) *)

let check_function func =

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

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

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

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


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

  let type_of_pointer t ex = match t with
      Pointer(_ as x) -> x;
    | _ -> raise (Failure ("non-pointer expression " ^ string_of_expr ex ^
    " is being used as a pointer"))
  in

  (* Return the type of an expression or throw an exception *)
  let rec expr table = function
      Inf -> Point
    | Null -> Pointer(Void)
    | Literal _ -> Int
    | Id s -> type_of_identifier s table
    | Ch _ -> Char
    | String _ -> Pointer(Char)
    | Subscript(a, i) as e -> if (expr table i) = Int then (type_of_pointer
    (type_of_identifier a table) e) else raise (Failure ("use of non-integer
```

```
        type as index in " ^
      string_of_expr e))
    | Binop(e1, op, e2) as e -> let t1 = expr table e1 and t2 = expr table e2
        in
 (match op with
       Add | Sub when t1 = Pointer(Point) && t2 = Pointer(Point) ->
          Pointer(Point)
     | Mult when t1 = Stone && t2 = Pointer(Point) -> Pointer(Point)
     | Add | Sub | Mult | Div | Pow when t1 = Int && t2 = Int -> Int
     | Add | Sub | Mult | Div | Pow when t1 = Stone && t2 = Stone -> Stone
     | Add | Sub | Mult | Pow when t1 = Mint && t2 = Mint -> Mint
     | Pow when t1 = Mint && t2 = Stone -> Mint
 | Equal | Neq when t1 = t2 -> Int (* might want to extend this to allow
     e.g., t1 and t2 both integer types so one can do stone=int *)
 | Less | Leq | Greater | Geq when t1 = Int && t2 = Int -> Int
| Equal | Neq | Less | Leq | Greater | Geq when t1 = Stone && t2 = Stone -> Int
     | _ -> raise (Failure ("illegal binary operator " ^
           string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
           string_of_typ t2 ^ " in " ^ string_of_expr e))
     )
    | Unop(op, e) as ex -> let t = expr table e in
  (match op with
    Neg when t = Int -> Int
      | Neg when t = Stone -> Stone
      | Neg when t = Mint -> Mint
      | Neg when t = Pointer(Point) -> Pointer(Point)
      | Neg when t = Char -> Char
      | Not when t = Int -> Int
      | Deref -> type_of_pointer t e
      | AddrOf -> Pointer(t)
      | Access when t = Mint || t = Point || t = Curve -> Stone
      | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
        string_of_typ t ^ " in " ^ string_of_expr ex)))
    | Construct2(e1, e2) -> let t1 = expr table e1 and t2 = expr table e2 in
      (match (t1, t2) with
        (Stone, Stone) -> Mint
      | (Mint, Mint) -> Pointer(Curve)
      | _ -> raise (Failure ("illegal constructor type pair (" ^ string_of_typ
          t1
      ^ "," ^ string_of_typ t2 ^ ")")))
    | Construct3(e1, e2, e3) -> let t1 = expr table e1 and t2 = expr table e2
and t3 = expr table e3 in
      (match (t1, t2, t3) with
      | (Pointer(Curve), Stone, Stone) -> Pointer(Point)
      | _ -> raise (Failure ("illegal constructor type pair (" ^ string_of_typ
          t1
      ^ "," ^ string_of_typ t2 ^ "," ^ string_of_typ t3 ^ ")")))
    | Noexpr -> Void

    (* Definitely need to change this to support things which return lvalues,
     * e.g. dereferencing *)
    | Assign(var, e) as ex -> let lt = type_of_identifier var table
                          and rt = expr table e in
      if (lt, rt) = (Stone, Pointer(Char)) then Stone else
      check_assign lt rt (Failure ("illegal assignment " ^ string_of_typ lt ^
              " = " ^ string_of_typ rt ^ " in " ^
              string_of_expr ex))
    | ModAssign(var, e) as ex -> let lt = type_of_identifier var table
```

```
                              and rt = expr table e in
    (match (lt, rt) with
      ((Int|Stone) as t, (Int|Stone)) -> t
    | _ -> raise (Failure ("illegal use of %= with types " ^ string_of_typ
    lt ^ " and " ^ string_of_typ rt ^ " in " ^ string_of_expr ex)))
  | Call(fname, actuals) as call -> let fd = function_decl fname in
     if fname = "printf"
       then
          let _ = List.iter (fun e -> ignore(expr table e)) actuals in Void
     else
       if List.length actuals != List.length fd.formals
         then
         raise (Failure ("expecting " ^ string_of_int
           (List.length fd.formals) ^ " arguments in " ^ string_of_expr call))
       else
          let _ = List.iter2 (fun (ft, _) e -> let et = expr table e in
           ignore (check_assign ft et
           (Failure ("illegal actual argument found " ^ string_of_typ et ^
           " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr call))))
         fd.formals actuals
         in
       fd.typ
in

let check_int_expr table e = if expr table e != Int
 then raise (Failure ("expected integer expression in " ^ string_of_expr e))
 else () in

(* Verify a statement or throw an exception *)
let rec stmt table in_loop = function
    Block (vl, sl) -> let rec check_block block_table = function
       [Return _ as s] -> stmt block_table in_loop s
     | Return _ :: _ -> raise (Failure "nothing may follow a return")
     | (Block (_, _) as b) :: ss -> stmt block_table in_loop b; check_block
     block_table ss
     | s :: ss -> stmt block_table in_loop s ; check_block block_table ss
     | [] -> ()
    in
      List.iter (check_not_void (fun n -> "illegal void local " ^ n ^
        " in " ^ func.fname)) vl;
        (* check for void type *)

      report_duplicate (fun n -> "duplicate local " ^ n ^ " in " ^ func.fname)
        ((List.map snd vl) );
        (* check for duplicate names in that scope *)

      let new_table = List.fold_left (fun m (t, n) -> StringMap.add n t
        m) table vl in

      check_block new_table sl
        (* check the block with new lookup table *)

  | Expr e -> ignore (expr table e)
  | Return e -> let t = expr table e in if t = func.typ then () else
     raise (Failure ("return gives " ^ string_of_typ t ^ " expected " ^
                    string_of_typ func.typ ^ " in " ^ string_of_expr e))

  | If(p, b1, b2) -> check_int_expr table p; stmt table false b1; stmt table
```

```
        false b2
    | For(e1, e2, e3, st) -> ignore (expr table e1); check_int_expr table e2;
                            ignore (expr table e3); stmt table true st
    | While(p, s) -> check_int_expr table p; stmt table true s
    | DoWhile(s, p) -> stmt table true s; check_int_expr table p
    | Break -> if in_loop then () else
        raise (Failure ("break statement found outside of a loop context"))
    | Continue -> if in_loop then () else
        raise (Failure ("continue statement found outside of a loop context"))
    | NullStmt -> ()
  in

  (* Type of each variable (global, formal, or local *)
  let table = List.fold_left (fun m (t, n) -> StringMap.add n t m)
    StringMap.empty (globals @ func.formals @ func.locals) in

  stmt table false (Block ([], func.body))

 in
 List.iter check_function functions
```

Listing 9.7: codegen.ml

```
(* Code generation: translate takes a semantically checked AST and
produces LLVM IR

LLVM tutorial: Make sure to read the OCaml version of the tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)

module L = Llvm
module A = Ast

module StringMap = Map.Make(String)

let translate (globals, functions) =
  let context = L.global_context () in
  let the_module = L.create_module context "Cmod"
  (*and i64_t = L.i64_type context *)
  and i32_t = L.i32_type context
  and i8_t  = L.i8_type  context
  and void_t = L.void_type context in
  let obj_pointer = L.pointer_type (L.i8_type context) in (* void pointer, 8
      bytes *)
  let mint_type = L.struct_type context [| obj_pointer ; obj_pointer |] in (*
      struct of two void pointers *)
  let curve_type = L.struct_type context [| mint_type ; mint_type |] in (* cruve
      defined by two modints *)
  let point_type = L.struct_type context [| curve_type ; obj_pointer ;
  obj_pointer; i8_t |] in(* curve + two stones *)
  let point_ptr = L.pointer_type point_type in
```

```
let curve_ptr = L.pointer_type curve_type in
let mint_pointer = L.pointer_type mint_type in
(* Must consider best way to implement points wrt Inf *)
(* maybe define diff points for inf and normal to enforce that
it has to be one or two, not arb length array *)

let rec ltype_of_typ = function
    A.Int -> i32_t
  | A.Char -> i8_t (* chars are 1 byte ints *)
  | A.Void -> void_t
  | A.Stone -> obj_pointer (* Pointer to arb prec list for C lib *)
  | A.Mint -> mint_type
  | A.Curve -> curve_type
  | A.Point -> point_type
  | A.Pointer x -> L.pointer_type (ltype_of_typ x) in
  (* Cant define pointer w normal form bc need type at time *)

(* Declare each global variable; remember its value in a map *)
let global_vars =
  let global_var m (t, n) =
    let init = L.const_int (ltype_of_typ t) 0
    in StringMap.add n ((L.define_global n init the_module), (t, 0)) m in
  List.fold_left global_var StringMap.empty globals in

(* Declare printf(), which the print built-in function will call *)
let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
let printf_func = L.declare_function "printf" printf_t the_module in


let read_t = L.function_type i32_t [| L.pointer_type i8_t ; L.pointer_type
    i8_t |] in
let read_func = L.declare_function "scanf" read_t the_module in

let malloc_t = L.function_type (L.pointer_type i8_t) [| i32_t |] in
let malloc_func = L.declare_function "malloc" malloc_t the_module in

let free_t = L.function_type void_t [| L.pointer_type i8_t |] in
let free_func = L.declare_function "free" free_t the_module in


(* Declare other linked to / "built in" functions *)
(* Function returns an 8 byte pointer, taking in two 8 byte pointers as
    arguments *)
let mint_add_func_t = L.function_type mint_type [| mint_pointer ; mint_pointer
    |] in
let mint_add_func = L.declare_function "mint_add_func" mint_add_func_t
    the_module in

let mint_sub_func_t = L.function_type mint_type [| mint_pointer ; mint_pointer
    |] in
let mint_sub_func = L.declare_function "mint_sub_func" mint_sub_func_t
    the_module in

let mint_mult_func_t = L.function_type mint_type [| mint_pointer ;
    mint_pointer |] in
let mint_mult_func = L.declare_function "mint_mult_func" mint_mult_func_t
    the_module in
```

```
let mint_pow_func_t = L.function_type mint_type [| mint_pointer ; mint_pointer
    |] in
let mint_pow_func = L.declare_function "mint_pow_func" mint_pow_func_t
    the_module in

let mint_to_stone_func_t = L.function_type mint_type [| mint_pointer ;
    obj_pointer |] in
let mint_to_stone_func = L.declare_function "mint_to_stone_func"
    mint_to_stone_func_t the_module in

let stone_add_func_t = L.function_type obj_pointer [| obj_pointer ;
    obj_pointer |] in
let stone_add_func = L.declare_function "stone_add_func" stone_add_func_t
    the_module in

let stone_sub_func_t = L.function_type obj_pointer [| obj_pointer ;
    obj_pointer |] in
let stone_sub_func = L.declare_function "stone_sub_func" stone_sub_func_t
    the_module in

let stone_mult_func_t = L.function_type obj_pointer [| obj_pointer ;
    obj_pointer |] in
let stone_mult_func = L.declare_function "stone_mult_func" stone_mult_func_t
    the_module in

let stone_div_func_t = L.function_type obj_pointer [| obj_pointer ;
    obj_pointer |] in
let stone_div_func = L.declare_function "stone_div_func" stone_div_func_t
    the_module in

let stone_pow_func_t = L.function_type obj_pointer [| obj_pointer ;
    obj_pointer |] in
let stone_pow_func = L.declare_function "stone_pow_func" stone_pow_func_t
    the_module in

let stone_mod_func_t = L.function_type obj_pointer [| obj_pointer ;
    obj_pointer |] in
let stone_mod_func = L.declare_function "stone_mod_func" stone_mod_func_t
    the_module in

let stone_eq_func_t = L.function_type i32_t [| obj_pointer ; obj_pointer |] in
let stone_eq_func = L.declare_function "stone_eq_func" stone_eq_func_t
    the_module in

let stone_neq_func_t = L.function_type i32_t [| obj_pointer ; obj_pointer |] in
let stone_neq_func = L.declare_function "stone_neq_func" stone_neq_func_t
    the_module in

let stone_leq_func_t = L.function_type i32_t [| obj_pointer ; obj_pointer |] in
let stone_leq_func = L.declare_function "stone_leq_func" stone_leq_func_t
    the_module in

let stone_geq_func_t = L.function_type i32_t [| obj_pointer ; obj_pointer |] in
let stone_geq_func = L.declare_function "stone_geq_func" stone_geq_func_t
    the_module in

let stone_less_func_t = L.function_type i32_t [| obj_pointer ; obj_pointer |]
    in
```

```
let stone_less_func = L.declare_function "stone_less_func" stone_less_func_t
    the_module in

let stone_greater_func_t = L.function_type i32_t [| obj_pointer ; obj_pointer
    |] in
let stone_greater_func = L.declare_function "stone_greater_func"
    stone_greater_func_t the_module in

let stone_print_func_t = L.function_type i32_t [| obj_pointer |] in
let stone_print_func = L.declare_function "stone_print_func"
    stone_print_func_t the_module in

let mint_print_func_t = L.function_type i32_t [| mint_type |] in
let mint_print_func = L.declare_function "mint_print_func" mint_print_func_t
    the_module in

let div_print_func_t = L.function_type i32_t [| mint_type |] in
let div_print_func = L.declare_function "div_print_func" div_print_func_t
    the_module in

let point_print_func_t = L.function_type i32_t [| point_ptr |] in
let point_print_func = L.declare_function "point_print_func"
    point_print_func_t the_module in

let point_print_sep_func_t = L.function_type i32_t [| point_ptr |] in
let point_print_sep_func = L.declare_function "point_print_sep_func"
point_print_sep_func_t the_module in

let curve_print_func_t = L.function_type i32_t [| curve_ptr |] in
let curve_print_func = L.declare_function "curve_print_func"
    curve_print_func_t the_module in

let point_add_func_t = L.function_type point_ptr [| point_ptr ; point_ptr |] in
let point_add_func = L.declare_function "point_add_func" point_add_func_t
    the_module in

let point_sub_func_t = L.function_type point_ptr [| point_ptr ; point_ptr |] in
let point_sub_func = L.declare_function "point_sub_func" point_sub_func_t
    the_module in

let atoi_func_t = L.function_type i32_t [| L.pointer_type i8_t |] in
let atoi_func = L.declare_function "atoi" atoi_func_t the_module in

(* stone * point, i.e. add point to itself stone many times *)
let point_mult_func_t = L.function_type point_ptr [| obj_pointer ; point_ptr
    |] in
let point_mult_func = L.declare_function "point_mult_func" point_mult_func_t
    the_module in

let stone_create_func_t = L.function_type obj_pointer [| L.pointer_type i8_t
    |] in
let stone_create_func = L.declare_function "stone_create_func"
    stone_create_func_t the_module in

let curve_create_func_t = L.function_type curve_ptr [| mint_type ; mint_type
    |] in
let curve_create_func = L.declare_function "curve_create_func"
    curve_create_func_t the_module in
```

```
 let point_create_func_t = L.function_type point_ptr
   [| curve_ptr ; obj_pointer ; obj_pointer |] in
 let point_create_func = L.declare_function "point_create_func"
   point_create_func_t the_module in

 let stone_free_t = L.function_type i32_t [| L.pointer_type i8_t |] in (* bn
     free func *)
 let stone_free_func = L.declare_function "stone_free_func" stone_free_t
     the_module in

(* let mint_free_t = L.function_type i32_t [| mint_pointer |] in
 let mint_free_func = L.declare_function "mint_free_func" mint_free_t
     the_module in *)

 let access_mint_t = L.function_type obj_pointer [| mint_type ; i32_t |] in
   let access_mint = L.declare_function "access_mint" access_mint_t the_module
       in

 let access_curve_t = L.function_type obj_pointer [| curve_ptr ; i32_t |] in
   let access_curve = L.declare_function "access_curve" access_curve_t
       the_module in

 let access_point_t = L.function_type obj_pointer [| point_ptr ; i32_t |] in
   let access_point = L.declare_function "access_point" access_point_t
       the_module in

 (* let invert_point_func_t = L.function_type point_type [| point_type |] in
   let invert_point_func = L.declare_function "invert_point_func"
       invert_point_func_t the_module in *)


 (* Define each function (arguments and return type) so we can call it *)
 let function_decls =
   let function_decl m fdecl =
     let name = fdecl.A.fname
     and formal_types =
  Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.A.formals)
     in let ftype = L.function_type (ltype_of_typ fdecl.A.typ) formal_types in
     StringMap.add name (L.define_function name ftype the_module, fdecl) m in
   List.fold_left function_decl StringMap.empty functions in

 (* Fill in the body of the given function *)
 let build_function_body fdecl =
   let (the_function, _) = StringMap.find fdecl.A.fname function_decls in
   let builder = L.builder_at_end context (L.entry_block the_function) in

   let char_format_str = L.build_global_stringptr "%s" "fmt2" builder in

   (* Construct the function's "locals": formal arguments and locally
      declared variables. Allocate each on the stack, initialize their
      value, if appropriate, and remember their values in the "locals" map *)

   (* Return the value for a variable or formal argument *)
   let lookup n table = try StringMap.find n table
                with Not_found -> StringMap.find n global_vars
   in
```

```
let manage l1 l2 ex ex2 =
     let _ = if (l1 = 0) then
         ignore(L.build_call stone_free_func [| ex |] "res" builder)
     else () in
     if (l2 = 0) then
          ignore(L.build_call stone_free_func [| ex2 |] "res" builder)
     else ()
in

(*let manage_mint l1 l2 ex ex2 =
     let _ = if (l1 = 0) then
         ignore(L.build_call mint_free_func [| ex |] "res" builder)
     else () in
     if (l2 = 0) then
          ignore(L.build_call mint_free_func [| ex2 |] "res" builder)
     else ()
in    *)

(* Construct code for an expression; return its value *)
let rec expr table builder = function
    A.Literal i -> (L.const_int i32_t i, (A.Int, 0))
    (*we dont want too big of int in here, maybe declare stone literals as
        strings*)
  | A.String s -> (L.build_global_stringptr s "fmts" builder,
      (A.Pointer(A.Char), 0))
  | A.Noexpr -> (L.const_int i32_t 0, (A.Void, 0))
  | A.Id s ->
    let binding = lookup s table in
      (L.build_load (fst binding) s builder, (fst (snd binding), 1))
  | A.Construct2 (e1, e2) ->
    let (e1', (t1, _)) = expr table builder e1
    and (e2', (t2, _)) = expr table builder e2 in
    (match (t1, t2) with
      (A.Stone, A.Stone) ->
       let struct_m = L.undef mint_type in
         let reduced_val = L.build_call stone_mod_func [| e1' ; e2' |]
            "stone_mod_res" builder in
           let struct_m2 = L.build_insertvalue struct_m (reduced_val) 0 "sm"
               builder in
       (L.build_insertvalue struct_m2 e2' 1 "sm2" builder, (A.Mint, 1)) (*1
           right?*)
     | (A.Mint, A.Mint) ->
       (L.build_call curve_create_func [| e1' ; e2' |] "curve_create_res"
       builder, (A.Pointer(A.Curve), 1))
     | _ -> raise(Failure("wrong types in construct2")))
     (* impossible; semant will check this *)

  | A.Construct3 (e1, e2, e3) ->
    let (e1', (t1, _)) = expr table builder e1
    and (e2', (t2, _)) = expr table builder e2
    and (e3', (t3, _)) = expr table builder e3 in
    (match (t1, t2, t3) with
      (A.Pointer(A.Curve), A.Stone, A.Stone) -> (*only construct 3?*)
          (L.build_call point_create_func [| e1' ; e2' ; e3' |]
            "point_create_res" builder, (A.Pointer(A.Point), 1))
     | _ -> raise(Failure("wrong types in construct3")))
     (* this last match is impossible; semant will check this
      * correct solution is to make a "polymorphic variant"; no one has
```

```
      * time for that *)
  | A.Binop (e1, op, e2) ->
    let (e1', (t1, leaf1)) = expr table builder e1
    and (e2', (t2, leaf2)) = expr table builder e2 in
    (match (t1, t2) with
       (A.Int, A.Int) ->
         ((match op with
           A.Add     -> L.build_add
         | A.Sub     -> L.build_sub
         | A.Mult    -> L.build_mul
         | A.Div     -> L.build_sdiv
         | A.And     -> L.build_and
         | A.Or      -> L.build_or
         | A.Equal   -> L.build_icmp L.Icmp.Eq
         | A.Neq     -> L.build_icmp L.Icmp.Ne
         | A.Less    -> L.build_icmp L.Icmp.Slt
         | A.Leq     -> L.build_icmp L.Icmp.Sle
         | A.Greater -> L.build_icmp L.Icmp.Sgt
         | A.Geq     -> L.build_icmp L.Icmp.Sge
         | _ as o -> raise(Failure("Illegal operator " ^ A.string_of_op o
         ^ " in int * int binop"))
         ) e1' e2' "tmp" builder, (A.Int, 0))
      | (A.Mint, A.Mint) ->
         let ptr1 = L.build_alloca mint_type "e1" builder and
         ptr2 = L.build_alloca mint_type "e2" builder in
         let _ = L.build_store e1' ptr1 builder and
         _ = L.build_store e2' ptr2 builder in
         ((match op with
             A.Add ->
               L.build_call mint_add_func [| ptr1 ; ptr2 |] "mint_add_res"
                   builder (*?? can i just this?*)
           | A.Sub ->
               L.build_call mint_sub_func [| ptr1 ; ptr2 |] "mint_sub_res"
                   builder
           | A.Mult ->
               L.build_call mint_mult_func [| ptr1 ; ptr2 |] "mint_mult_res"
                   builder
           | A.Pow ->
               L.build_call mint_pow_func [| ptr1 ; ptr2 |] "mint_pow_res"
                   builder
           | _ as o -> raise(Failure("Illegal operator " ^ A.string_of_op o
            ^ " in mint * mint binop"))
         ), (A.Mint, 0))

        (*Raise mint to stone*)
      | (A.Mint, A.Stone) ->
         ((match op with
           (* In semant, check that this is only op possible *)
         A.Pow ->
             let ptr = L.build_alloca mint_type "e1" builder in
             let _ = L.build_store e1' ptr builder in

             L.build_call mint_to_stone_func [| ptr ; e2' |]
             "mint_to_stone_res" builder
       | _ as o -> raise(Failure("Illegal operator " ^ A.string_of_op o
         ^ " in mint * stone binop"))

         ), (A.Mint, 0))
```

```
| (A.Stone, A.Stone) ->
    ((match op with
      A.Add ->
      let call = L.build_call stone_add_func [| e1' ; e2' |]
          "stone_add_res" builder in
       let _ = manage leaf1 leaf2 e1' e2' in
      call
      (*L.build_call stone_add_func [| e1' ; e2' |] "stone_add_res"
          builder*)
    | A.Sub ->
      let call = L.build_call stone_sub_func [| e1' ; e2' |]
          "stone_sub_res" builder in
        let _ = manage leaf1 leaf2 e1' e2' in
      call
    | A.Mult ->
      let call = L.build_call stone_mult_func [| e1' ; e2' |]
          "stone_mult_res" builder in
        let _ = manage leaf1 leaf2 e1' e2' in
      call
    | A.Div ->
      let call = L.build_call stone_div_func [| e1' ; e2' |]
          "stone_div_res" builder in
        let _ = manage leaf1 leaf2 e1' e2' in
      call
    | A.Pow ->
      let call = L.build_call stone_pow_func [| e1' ; e2' |]
          "stone_pow_res" builder in
        let _ = manage leaf1 leaf2 e1' e2' in
      call
    | A.Mod ->
      let call = L.build_call stone_mod_func [| e1' ; e2' |]
          "stone_mod_res" builder in
        let _ = manage leaf1 leaf2 e1' e2' in
      call
    | A.Equal ->
      let call = L.build_call stone_eq_func [| e1' ; e2' |]
          "stone_eq_res" builder in
        let _ = manage leaf1 leaf2 e1' e2' in
      call
    | A.Neq ->
      let call = L.build_call stone_neq_func [| e1' ; e2' |]
          "stone_neq_res" builder in
        let _ = manage leaf1 leaf2 e1' e2' in
      call
    | A.Less ->
      let call = L.build_call stone_less_func [| e1' ; e2' |]
          "stone_less_res" builder in
        let _ = manage leaf1 leaf2 e1' e2' in
      call
    | A.Leq ->
      let call = L.build_call stone_leq_func [| e1' ; e2' |]
          "stone_leq_res" builder in
        let _ = manage leaf1 leaf2 e1' e2' in
      call
    | A.Greater ->
      let call = L.build_call stone_greater_func [| e1' ; e2' |]
          "stone_greater_res" builder in
```

```
            let _ = manage leaf1 leaf2 e1' e2' in
          call
        | A.Geq ->
          let call = L.build_call stone_geq_func [| e1' ; e2' |]
              "stone_geq_res" builder in
            let _ = manage leaf1 leaf2 e1' e2' in
          call
        | _ as o -> raise(Failure("Illegal operator " ^ A.string_of_op o
        ^ " in stone * stone binop"))
        ), (A.Stone, 0))
     | (A.Pointer(A.Point), A.Pointer(A.Point)) ->
        ((match op with
        A.Add ->
          L.build_call point_add_func [| e1' ; e2' |] "point_add_res"
              builder
      | A.Sub ->
          L.build_call point_sub_func [| e1' ; e2' |] "point_sub_res"
              builder
      | _ as o -> raise(Failure("Illegal operator " ^ A.string_of_op o
        ^ " in point* * point* binop"))
        ), (A.Pointer(A.Point), 0))
    | (A.Stone, A.Pointer(A.Point)) ->
        ((match op with
        A.Mult ->
            L.build_call point_mult_func [| e1' ; e2' |] "point_mult_res"
            builder
        | _ as o -> raise(Failure("Illegal operator " ^ A.string_of_op o
        ^ " in stone * point binop"))
        ), (A.Pointer(A.Point), 0))
    | _ ->
          raise(Failure("illegal binop type " ^ A.string_of_typ t1 ^
          A.string_of_op op ^ A.string_of_typ t2))
   )


  | A.Unop(op, e) ->
      let e', (t, _) = expr table builder e in
      ((match op with
       A.Neg    -> (match t with
           A.Int -> L.build_neg e' "tmp" builder
         (* | A.Point -> L.build_call invert_point_func [| e' |]
             "invert_point_func" builder *) ) (* Point inversion *)
      | A.Not    -> L.build_icmp L.Icmp.Eq (L.const_null (ltype_of_typ t))
          e' "tmp" builder (* Still need to test on Pointer types *)
      | _ -> raise(Failure("not implemented yet")) e' "tmp" builder
    (* | A.Deref  -> L.build_load e' "tmp" builder *) (* load object pointed
        to *)
    (* | A.AddrOf -> fst(lookup e') *)(*L.build_store e' builder*)
      ), (match op with
        A.Neg -> (t, 0)
        | A.Not -> (t, 0)
        | _ -> (t, 0)
        (* | A.Deref -> (match t with
            A.Pointer x -> x)
        | A.AddrOf -> A.Pointer t *)))
   | A.Assign (s, e) -> let (e', (t, _)) = expr table builder e and
                        (* if t string, otherwise is behavior normal?*)
                      (*snd lookup is type of thing*)
```

```
                    ltype = (fst (snd (lookup s table))) in (match (ltype,
                        t) with
                    | (A.Stone, A.Pointer(A.Char)) ->
                       let ptr =
                        L.build_call stone_create_func [|e' |]
                            "stone_create_func" builder in
                         (*let res =
                          L.build_call stone_char_func [| e' ; ptr |]
                          "stone_char_func" builder in *)
                          ignore(L.build_store ptr (fst (lookup s table))
                              builder); (ptr, (t, 0))


                    | _ -> ignore (L.build_store e' (fst (lookup s table))
                        builder); (e', (t, 0)) )



| A.Call("access_mint", [e; i]) -> let (e', (t, _)) = expr table builder e
    and (i', (t', _)) = expr table builder i in
    (L.build_call access_mint [| e' ; i' |] "access_mint" builder,
        (A.Stone, 0));
| A.Call("access_curve", [e; i]) -> let (e', (t, _)) = expr table builder
    e and (i', (t', _)) = expr table builder i in
    (L.build_call access_curve [| e' ; i' |] "access_curve" builder,
        (A.Stone, 0));
| A.Call("access_point", [e; i]) -> let (e', (t, _)) = expr table builder
    e and (i', (t', _)) = expr table builder i in
    (L.build_call access_point [| e' ; i' |] "access_point" builder,
        (A.Stone, 0));
| A.Call ("printf", act) ->
    let actuals, _ = List.split (List.rev (List.map (expr table builder)
    (List.rev act))) in
    let result = "" in (* printf is void function *)
    (L.build_call printf_func (Array.of_list actuals) result builder,
      (A.Pointer(A.Char), 0))
| A.Call("print_point", [e]) -> let (e', (t, _)) = expr table builder e in
    (L.build_call point_print_func [| e' |] "point_print_res" builder, (t,
        0));
| A.Call("print_point_sep", [e]) -> let (e', (t, _)) = expr table builder
    e in
    (L.build_call point_print_sep_func [| e' |] "point_print_res" builder,
        (t, 0));
| A.Call("print_curve", [e]) -> let (e', (t, _)) = expr table builder e in
    (L.build_call curve_print_func [| e' |] "curve_print_res" builder, (t,
        0));
| A.Call("print_stone", [e]) -> let (e', (t, _)) = expr table builder e in
    (L.build_call stone_print_func [| e' |] "stone_print_func" builder, (t,
        0));
| A.Call("print_mint", [e]) -> let (e', (t, _)) = expr table builder e in
    (L.build_call mint_print_func [| e' |] "mint_print_func" builder, (t,
        0));
| A.Call("print_div", [e]) -> let (e', (t, _)) = expr table builder e in
    (L.build_call div_print_func [| e' |] "div_print_func" builder, (t, 0));
| A.Call("scanf", [e]) ->
    let (e', (t, _)) = expr table builder e in
      ignore(L.build_call read_func [| char_format_str ; e' |] "scanf"
          builder );
```

```
        (e' , (t, 0))
  | A.Call("malloc", [e]) ->
     let (e', (t, _)) = expr table builder e in
     (L.build_call malloc_func [| e' |] "malloc" builder, (t, 0))
  | A.Call("free", [e]) ->
     let (e', (t, _)) = expr table builder e in
     (L.build_free e' builder, (A.Void, 0)) (*void correct?*)
  | A.Call("atoi", [e]) ->
     let (e', (t, _)) = expr table builder e in
     (L.build_call atoi_func [| e' |] "atoi_res" builder, (t, 0));
  | A.Call (f, act) ->
    let (fdef, fdecl) = StringMap.find f function_decls in
      let actuals, _ = List.split (List.rev (List.map (expr table builder)
          (List.rev act))) in
      let result = (match fdecl.A.typ with A.Void -> ""
                                    | _ -> f ^ "_result") in
     (L.build_call fdef (Array.of_list actuals) result builder,
        (fdecl.A.typ, 0))
  | _ -> raise(Failure("illegal expression"))

in

(* Invoke "f builder" if the current block doesn't already
   have a terminal (e.g., a branch). *)
let add_terminal builder f =
  match L.block_terminator (L.insertion_block builder) with
Some _ -> ()
  | None -> ignore (f builder) in

(* Build the code for the given statement; return the builder for
   the statement's successor *)

let rec stmt table builder = function
    A.Block (vl, sl) ->
    let new_table =
     let add_local m (t, n) =
      let local_var = L.build_alloca (ltype_of_typ t) n builder
       in StringMap.add n (local_var, (t, 0)) m
     in
     List.fold_left add_local table vl
   in
   List.fold_left (stmt new_table) builder sl

  | A.Expr e -> ignore (expr table builder e); builder
  | A.Return e -> ignore (match fdecl.A.typ with
                A.Void -> L.build_ret_void builder
   | _ -> L.build_ret (fst (expr table builder e)) builder); builder
  | A.If (predicate, then_stmt, else_stmt) ->
     let bool_val = fst (expr table builder predicate) in
let merge_bb = L.append_block context "merge" the_function in

let then_bb = L.append_block context "then" the_function in
add_terminal (stmt table (L.builder_at_end context then_bb) then_stmt)
  (L.build_br merge_bb);

let else_bb = L.append_block context "else" the_function in
add_terminal (stmt table (L.builder_at_end context else_bb) else_stmt)
  (L.build_br merge_bb);
```

```
        ignore (L.build_cond_br bool_val then_bb else_bb builder);
        L.builder_at_end context merge_bb

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

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

        let pred_builder = L.builder_at_end context pred_bb in
        let bool_val = fst (expr table pred_builder predicate) in

        let merge_bb = L.append_block context "merge" the_function in
        ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
        L.builder_at_end context merge_bb


    | A.For (e1, e2, e3, body) -> stmt table builder
    ( A.Block ([], [A.Expr e1 ; A.While (e2, A.Block ([], [body ; A.Expr e3]))
        ] ))
    | _ -> raise(Failure("illegal statement"))

  in

  let local_vars =
    let add_formal m (t, n) p = L.set_value_name n p;
       let local = L.build_alloca (ltype_of_typ t) n builder in
       ignore (L.build_store p local builder);
       StringMap.add n (local, (t, 0)) m in (* local, t to add type info to map
           as well *)

    let add_local m (t, n) =
       let local_var = L.build_alloca (ltype_of_typ t) n builder
       in StringMap.add n (local_var, (t, 0)) m in (* BSURE this might be it *)

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

  (* Build the code for each statement in the function *)
  let builder = stmt local_vars builder (A.Block ([], fdecl.A.body)) in

  (* Add a return if the last block falls off the end *)
  add_terminal builder (match fdecl.A.typ with
      A.Void -> L.build_ret_void
    | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
in

List.iter build_function_body functions;
the_module
```

Listing 9.8: scannerprint.mll

```
(* Prints program tokens *)
```

```
{ open Printf }

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"    { comment lexbuf }        (* Comments *)
| "//"    { comment2 lexbuf }
| '('     { print_string "LPAREN " }
| ')'     { print_string "RPAREN " }
| '{'     { print_string "LBRACE " }
| '}'     { print_string "RBRACE " }
| '['     { print_string "LSQUARE " }
| ']'     { print_string "RSQUARE " }
| ';'     { print_string "SEMI " }
| ','     { print_string "COMMA " }
| '+'     { print_string "PLUS " }
| '-'     { print_string "MINUS " }
| '*'     { print_string "STAR " }
| '/'     { print_string "DIVIDE " }
| '%'     { print_string "MOD " }
| '&'     { print_string "ADDRESSOF " }
| '='     { print_string "ASSIGN " }
| '^'     { print_string "POW " }
| "%="    { print_string "MODASSIGN " }
| "=="    { print_string "EQ " }
| "!="    { print_string "NEQ " }
| '<'     { print_string "LT " }
| "<="    { print_string "LEQ " }
| '>'     { print_string "GT " }
| ">="    { print_string "GEQ " }
| "&&"    { print_string "AND " }
| "||"    { print_string "OR " }
| '!'     { print_string "NOT " }
| "if"    { print_string "IF " }
| "else"  { print_string "ELSE " }
| "for"   { print_string "FOR " }
| "while" { print_string "WHILE " }
| "do"    { print_string "DO " }
| "break" { print_string "BREAK " }
| "continue" { print_string "CONTINUE " }
| "return" { print_string "RETURN " }
| "int"   { print_string "INT " }
| "void"  { print_string "VOID " }
| "char"  { print_string "CHAR " }
| "NULL"  { print_string "NULL " }
| "stone" { print_string "STONE " }
| "mint"  { print_string "MINT " }
| "point" { print_string "POINT " }
| "curve" { print_string "CURVE " }
| '~'     { print_string "INF " }
| "access" { print_string "ACCESS " }
| ['\'']['  '-'~']*['\'']  { print_string "CHARLIT " }
| ['"'][' '-'~']*['"'] { print_string "STRING " }
| ['0'-'9']+ { print_string "LITERAL " }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* { print_string "ID " }

and comment = parse
  "*/" { token lexbuf }
```

```
| _    { comment lexbuf }

and comment2 = parse
  '\n' { token lexbuf }
| _    { comment2 lexbuf }

{
  let main () =
    let lexbuf = Lexing.from_channel stdin in
    try
        while true do
            ignore (token lexbuf)
        done
    with _ -> print_string "EOF\n"
  let _ = Printexc.print main ()

}
```

9.1.2 Preprocessing

Listing 9.9: prescanner.mll

```
  (* Ocamllex scanner for MicroC *)
{   open Parser
    module B = Buffer }

(* why are some string and some chars LT, GT eg *)

rule prepro m = parse
  '#'       { handle_pound m lexbuf }
| _ as t    { print_char t; prepro m lexbuf }

and

handle_pound m = parse
  "include" { handle_file m lexbuf }
| "define"  { handle_define m lexbuf }
| "ifdef"   { handle_ifdef m lexbuf }
| "endif"   { raise(Failure("unexpected #endif")) }
| _ as t    { raise(Failure("unexpected word " ^ t ^ " found after #")) }

and

handle_file m = parse
  "\""(_ as file)"\""   { let in_stream = open_in file in prepro m in_stream }

and

handle_define m = parse
  (_ as k)" "(_ as v) { StringMap.add k v m }

and

handle_ifdef m = parse
" "(_ as t)" "     { try StringMap.find t m w; find_endif lexbuf with
                     Not_found -> prepro m lexbuf }
```

Listing 9.10: preprocessor.py

```
import sys

def add_preprocess_name(in_file_name):
    temp = in_file_name.split('.')
    temp[-2] += "_preprocess"
    return '.'.join(temp)

def preprocess_file(in_file_name, out_file, table):
    in_file = open(in_file_name, 'r')
    wait_endif = 0
    for line in in_file:
        sys.stdout.write("%50s %s\n"%(line.rstrip('\n'), in_file_name))
        if wait_endif:
            try:
                if line.split()[0] == "#endif":
                    wait_endif = 0
            except IndexError:
                pass
        elif line.strip() == '':
            out_file.write(line)
        elif line.strip()[0] == "#":
            line = line[1:]
            tokens = line.split()
            if tokens[0] == "include":
                file_name = tokens[1].lstrip(' "').rstrip(' "')
                preprocess_file(file_name, out_file, table)
            elif tokens[0] == "define": # just for build guards
                                        # not good for textual substitutions
                table[tokens[1]] = 1
                # change this to tokens[2] for textual substitutions,
                # when implemented
            elif tokens[0] == "ifdef":
                if tokens[1] not in table:
                  wait_endif = 1
            elif tokens[0] == "ifndef":
                print table
                if tokens[1] in table:
                  wait_endif = 1
            elif tokens[0] == "endif":
                pass
            else:
                raise Exception("illegal token " + tokens[0] + " found after #")
        else:
            out_file.write(line)

def main():
    in_file_name = sys.argv[1]
    out_file_name = add_preprocess_name(in_file_name)
    out_file = open(out_file_name, 'w')
    symbol_table = {}
    preprocess_file(in_file_name, out_file, symbol_table)
    return 0

if __name__ == '__main__':
    main()
```

9.1.3 C Wrappers

Listing 9.11: access.c

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <openssl/bn.h>
#include "types.h"


// takes a mint and an int (0 or 1),
// returning val for idx 0 and mod for idx 1
void *access_mint(struct mint m, int index) {

   if(index == 0) { return m.val; }
   return m.mod;
}

// takes a curve pointer and an index (0-3)
// 0-1 corresponding to mint1 stones, 2-3 to mint2 stones
void *access_curve(struct curve* c, int index) {

   if(index < 2) {
      return access_mint(c->a, index);
   }
   else {
      return access_mint(c->b, index-2);
   }

}

// takes a point pointer and an index (0-5)
// 0-3 correspond to curve stones, 4, 5 correspond to x, y coordinates
void *access_point(struct point* p, int index) {

   if(index < 4) {
      return access_curve(&(p->E), index);
   }
   else {
      if(index == 4) {
         return p->x;
      }
   }
   return p->y;
}
```

Listing 9.12: special_arith.c

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <openssl/bn.h>
#include "types.h"

/*
* Stone
* @Michael We will need to implement these based on the bignum
```

```
* library. Implement all others assuming you have these
* functions. Ill make some headway on this once I
* come to a conclusion on a library
*/

int stone_print_func(void *a)
{
  printf("%s\n", BN_bn2dec(a));

  return 0;
}
//construct

void *stone_create_func(char *str) {
    BIGNUM *r = BN_new();
    BN_dec2bn(&r, str);
    //fprintf(stderr, "Creating %p\n", r);

    return r;
}

struct curve *curve_create_func(struct mint a, struct mint b) {
    struct curve *E;
    E = (struct curve *)malloc(sizeof(struct curve));
    E->a = a;
    E->b = b;
    return E;
}

struct point *point_create_func(struct curve *E, void *a, void *b) {
    struct point *R;
    R = (struct point *)malloc(sizeof(struct point));
    R->E = *E;
    R->x = a;
    R->y = b;
    R->inf = 0;
    return R;
}


int stone_free_func(void *a){
  //fprintf(stderr, "Freeing %p\n", a);

  BN_free(a);
  return 0;
}

//Add
void* stone_add_func(void *a, void *b)
{
  BIGNUM *r = BN_new();
  //fprintf(stderr, "a: %p\nb: %p\n", a, b);
  //fprintf(stderr, "Creating to add %p\n", r);

  BN_add(r, a, b);
  return r;
}
```

```
//Subtract
void* stone_sub_func(void *a, void *b)
{
  BIGNUM *r = BN_new();
  BN_sub(r, a, b);
  return r;
}

//Multiply
void* stone_mult_func(void *a, void *b)
{
  BIGNUM *r = BN_new();
  BN_CTX* ctx = BN_CTX_new();
  BN_mul(r, a, b, ctx);
  BN_CTX_free(ctx);

  return r;
}

//Divide
void* stone_div_func(void *a, void *b)
{
  BIGNUM *r = BN_new();
  BN_CTX *ctx = BN_CTX_new();
  BN_div(r, NULL, a, b, ctx);
  BN_CTX_free(ctx);

  return r;
}

//Mod
void* stone_mod_func(void *a, void *b)
{
  BIGNUM *r = BN_new();
  BN_CTX* ctx = BN_CTX_new();
  BN_mod(r, a, b, ctx);
  if (BN_is_negative(r)) {
      BN_add(r, r, b);
  }
  BN_CTX_free(ctx);
  return r;
}

//Exponent
void* stone_pow_func(void *a, void *p)
{
  BIGNUM *r = BN_new();
  BN_CTX* ctx = BN_CTX_new();
  BN_exp(r, a, p, ctx);
  BN_CTX_free(ctx);

  return r;
}

//Comparators

//0 if true, else false
int stone_eq_func(void *a, void *b)
```

```
{
  return BN_cmp(a, b);
}

//0 if true, else false
int stone_neq_func(void *a, void *b)
{
  return !BN_cmp(a, b);
}

int stone_less_func(void *a, void *b)
{
  if (BN_cmp(a, b) == -1)
    return 0;
  return 1;
}

int stone_leq_func(void *a, void *b)
{
  if (BN_cmp(a, b) <= 0)
    return 0;
  return 1;

}

int stone_greater_func(void *a, void *b)
{
  if (BN_cmp(a, b) == 1)
      return 0;
    return 1;
}

int stone_geq_func(void *a, void *b)
{
  if (BN_cmp(a, b) >= 0)
    return 0;
  return 1;
}

/* for point mult */

char *hex_to_bin_help(char *hx) {
    size_t len = strlen(hx);
    char *x = (char *)malloc(len * 4 + 1);
    char *buf;
    for (size_t j = 0; j < len; j = j + 1) {
        switch (*hx) {
            case '0':
                buf = "0000";
                break;
            case '1':
                buf = "0001";
                break;
            case '2':
                buf = "0010";
                break;
            case '3':
                buf = "0011";
```

```
                break;
            case '4':
                buf = "0100";
                break;
            case '5':
                buf = "0101";
                break;
            case '6':
                buf = "0110";
                break;
            case '7':
                buf = "0111";
                break;
            case '8':
                buf = "1000";
                break;
            case '9':
                buf = "1001";
                break;
            case 'A':
                buf = "1010";
                break;
            case 'B':
                buf = "1011";
                break;
            case 'C':
                buf = "1100";
                break;
            case 'D':
                buf = "1101";
                break;
            case 'E':
                buf = "1110";
                break;
            case 'F':
                buf = "1111";
                break;
        }
        for (int i = 0; i < 4; i++) {
            x[4*j+i] = buf[i];
        }
        hx++;
    }
    x[4*len] = '\0';
    return x;
}


void point_add_func_help(struct point *R, struct point *P, struct point *Q) {
    R->E = P->E;
    if (P->inf) {
        R->x = Q->x;
        R->y = Q->y;
        R->inf = Q->inf;
    } else if (Q->inf) {
        R->x = P->x;
        R->y = P->y;
        R->inf = P->inf;
```

```
} else { /* neither points are inf */
    BIGNUM *xval = BN_new();
    BIGNUM *yval = BN_new();
    BN_CTX *ctx = BN_CTX_new();

    BIGNUM *lambda = BN_new();
    BIGNUM *t1 = BN_new();
    BIGNUM *t2 = BN_new();

    // calculate lambda
    BN_sub(t1, Q->y, P->y);
    BN_sub(t2, Q->x, P->x);
    if (BN_is_zero(t2)) {
        if (BN_is_zero(t1)) {
            /* same point, double it
             * calculate lambda this way */
            BN_mod_sqr(t1, P->x, P->E.a.mod, ctx);
            BN_mod_add(t2, t1, t1, P->E.a.mod, ctx); /* t2 = 2 t1 */
            BN_mod_add(t2, t1, t1, P->E.a.mod, ctx); /* t1 = t1 + t2 = 3t1 */
            BN_mod_add(t1, t1, t2, P->E.a.mod, ctx);
            BN_mod_add(t1, t1, P->E.a.val, P->E.a.mod, ctx);

            BN_mod_add(t2, P->y, P->y, P->E.a.mod, ctx); /* t2 = 2 P.y */
            BN_mod_inverse(t2, t2, P->E.a.mod, ctx);

            BN_mod_mul(lambda, t1, t2, P->E.a.mod, ctx);

        } else {
            /* additive inverses, return inf
             * Fill coords with junk values from P */
            R->x = P->x;
            R->y = P->y;
            R->inf = 1;
            BN_free(t1);
            BN_free(t2);
            BN_CTX_free(ctx);
            return;
        }
    } else {
        // finish calculating lambda for "normal" case
        BN_mod_inverse(t2, t2, P->E.a.mod, ctx);
        BN_mod_mul(lambda, t1, t2, P->E.a.mod, ctx);
    }

    //calculate xval
    BN_mod_sqr(t1, lambda, P->E.a.mod, ctx);
    BN_mod_sub(t1, t1, P->x, P->E.a.mod, ctx);
    BN_mod_sub(xval, t1, Q->x, P->E.a.mod, ctx);

    //calculate yval
    BN_mod_sub(t1, P->x, xval, P->E.a.mod, ctx);
    BN_mod_mul(t1, lambda, t1, P->E.a.mod, ctx);
    BN_mod_sub(yval, t1, P->y, P->E.a.mod, ctx);

    //put in values
    R->x = xval;
    R->y = yval;
    R->inf = P->inf;
```

```
        BN_free(t1);
        BN_free(t2);
        BN_CTX_free(ctx);
    }
}

struct point *point_add_func(struct point *P, struct point *Q) {
    struct point *R;
    R = (struct point *)malloc(sizeof(struct point));
    point_add_func_help(R, P, Q);
    return R;
}

struct point *point_sub_func(struct point *P, struct point *Q) {
    ((BIGNUM *) Q->y)->neg = !((BIGNUM *) Q->y)->neg;
    struct point *R;
    R = point_add_func(P, Q);
    /* restore neg value of Q */
    ((BIGNUM *) Q->y)->neg = !((BIGNUM *) Q->y)->neg;
    return R;
}

struct point *point_mult_func(void *k, struct point *P) {
    char *x;
    char *z;
    BIGNUM *y;
    y = stone_create_func("26");
    z = BN_bn2hex((BIGNUM *) k);
    x = hex_to_bin_help(z);
    z = x; // free this at the end
    struct point *R;
    R = (struct point *)malloc(sizeof(struct point));
    R->E = P->E;
    R->x = P->x;
    R->y = P->y;
    R->inf = (*x) == '0' ? 1 : P->inf;
    // if first bit is 0, then return infinity.
    // this fixes leading zeroes in the binary string
    // else, set result equal to P
    while (*x != '\0') {
        // if bit is 1, R = 2R + P
        // if bit is 0, R = 2R
        point_add_func_help(R, R, R);
        if (*x++ == '1') {
            point_add_func_help(R, R, P);
        }
    }
    free(z);
    return R;
}

/*
* Mint
*/

//Add
struct mint mint_add_func(struct mint* a, struct mint* b) {
```

```
    BIGNUM *val = BN_new();
    BN_CTX *ctx = BN_CTX_new();

    //BN_mod_add_quick(val, v1, v2, v3);
    BN_mod_add(val, a->val, b->val, a->mod, ctx);
    BN_CTX_free(ctx);
    struct mint r;
    r.val = val;
    r.mod = a->mod; /* use a's modulus */
    return r;
}

struct mint mint_sub_func(struct mint* a, struct mint* b) {
    BIGNUM *val = BN_new();
    BN_CTX *ctx = BN_CTX_new();

    BN_mod_sub(val, a->val, b->val, a->mod, ctx);
    BN_CTX_free(ctx);
    struct mint r;
    r.val = val;
    r.mod = a->mod; /* use a's modulus */
    return r;
}

struct mint mint_mult_func(struct mint* a, struct mint* b) {
    BIGNUM *val = BN_new();
    BN_CTX *ctx = BN_CTX_new();

    BN_mod_mul(val, a->val, b->val, a->mod, ctx);
    BN_CTX_free(ctx);
    struct mint r;
    r.val = val;
    r.mod = a->mod; /* use a's modulus */
    return r;
}

struct mint mint_to_stone_func(struct mint *a, void *b) {
    BIGNUM *val = BN_new();
    BN_CTX *ctx = BN_CTX_new();
    if (BN_is_negative((BIGNUM *)b)) {
        BN_mod_inverse(a->val, a->val, a->mod, ctx);
    }
    BN_mod_exp(val, a->val, b, a->mod, ctx);
    /* BN_mod_exp takes the absolute value of b.
     * This is why this works */
    BN_CTX_free(ctx);
    struct mint r;
    r.val = val;
    r.mod = a->mod;
    return r;
}

struct mint mint_pow_func(struct mint* a, struct mint* b) {
    return mint_to_stone_func(a, b->val);
}

/* testing function */
```

```
int div_print_func(struct mint a) {
    printf("%s\n", BN_bn2dec(a.val));
    return 0;
}

int mint_print_func(struct mint a) {
    printf("<%s, %s>\n", BN_bn2dec(a.val), BN_bn2dec(a.mod));
    return 0;
}

int point_print_func(struct point *P) {
    //mint_print_func(P.E.a);
    //mint_print_func(P.E.b);
    if (P->inf) {
        printf("inf\n");
    } else {
        printf("<%s, %s>\n", BN_bn2dec(P->x), BN_bn2dec(P->y));
    }
    //stone_print_func(P.x);
    //stone_print_func(P.y);
    return 0;
}

int point_print_sep_func(struct point *P) {
    printf("%s\n%s\n", BN_bn2dec(P->x), BN_bn2dec(P->y));
    return 0;
}

int curve_print_func(struct curve *E) {
    printf("a: %s\nb: %s\np: %s\n", BN_bn2dec(E->a.val), BN_bn2dec(E->b.val),
           BN_bn2dec(E->a.mod));
    return 0;
}

//Equality and Inequality ops ofr mints are in LRM,
//but we can hold off on implemenitng

/*
* @Michael other stuff that is left is point/curve ops
* thats your expertise so ill leave it to you to
* define the headers and functions in the same way as above
*/
```

Listing 9.13: types.h

```
// Defines all CMod Types

struct mint {
    void *val;
    void *mod; //should be immutable
};

struct curve {
    struct mint a;
    struct mint b;
};

struct point {
```

```
    struct curve E;
    void *x;
    void *y;
    char inf;
};
```

9.2 Compiler Interface

Listing 9.14: cmc.sh

```
#!/bin/sh

#Requires you have LLI variable set (I reccomend in your bash profile) to your
    LLI
#may need to chmod this script to 755

VER="3.8"
LLC="/usr/local/opt/llvm@$VER/bin/llc-$VER"
CRYPTO="/usr/lib/libcrypto.0.9.8.dylib"
TEST="$2"

usage() { echo "Usage: $0 [-h help] [-t token] [-a ast] [-l llvm] [-c ll-file]
    [-s s-file] [-e exe-file] <file-name>.cm" 1>&2; exit 1; }

help() { echo "\n Welcome to the C% compiler CMC!
   \n USAGE: $0 [-h help] [-t token] [-a ast] [-l llvm] [-c ll-file] [-s s-file]
       [-e exe-file] <file-name>.cm\n
   \n OPTIONS:
   -h   help       This option prints this message!\n
   -t token   This option prints the tokenized program to stdout.\n
   -a   ast     This option prints the abstract syntax tree of the program to
       stdout.\n
   -l   llvm    Compiles <file-name>.cm to llvm and prints the result to
       stdout.\n
   -c   ll-file    Compiles <file-name>.cm to llvm and puts the result in
       <file-name>.ll. This is the default option.\n
   -s   assembly Compiles <file-name>.cm to llvm, translates to assembly, and
       puts the result in <file-name>.s
              (leaves <file-name>.ll in directory as well)\n
   -e   executable Creates the executable version of <file-name>.cm, simply
       called <file-name> to be run ./<file-name>
              (leaves behind the corresponding .ll and .s files as well)\n"
                  1>&2; exit 1; }

if getopts "h:t:a:l:c:s:e:" c; then
   basename=‘echo "$TEST" | sed 's/.*\\///
                           s/.cm//’‘

    case $c in
        h) # help
         help
         ;;
      t) # print tokenized program
         ocamllex scannerprint.mll
         ocaml scannerprint.ml < "$TEST"
         ;;
      a) # print the AST to stdout
         ocamllex scannerprint.mll
```

```
                    ocaml scannerprint.ml < "$TEST" | menhir --interpret
                        --interpret-show-cst parser.mly
        # ./cmod.native -a < "$TEST"
        ;;
     l) # compile to llvm, print to stdout
        ./cmod.native < "$TEST"
        ;;
     c) # compile to llvm, put in .ll file
        ./cmod.native < "$TEST" > ${basename}.ll
        ;;
     s) # translate to .s file
        ./cmod.native < "$TEST" > ${basename}.ll
        "$LLC" ${basename}.ll > ${basename}.s
        ;;
     e) # create executable
        ./cmod.native < "$TEST" > ${basename}.ll
        "$LLC" ${basename}.ll > ${basename}.s
        cc -o ${basename} ${basename}.s special_arith.o access.o "$CRYPTO"
        ;;
     *) # everything else
        usage
        ;;
   esac
else
   # DEFAULT
   TEST="$1"
   basename=`echo "$TEST" | sed 's/.*\\///
                            s/.cm//'`
   ./cmod.native < "$TEST" > ${basename}.ll
fi
```

Listing 9.15: cmod.sh

```
#!/bin/sh

VER="3.8"
LLC="/usr/local/opt/llvm@$VER/bin/llc-$VER"
CRYPTO="/usr/lib/libcrypto.0.9.8.dylib"
TEST="$1"

while getopts "v:" c; do
    case $c in
        v) # Use Travis Paths
            LLC="/usr/lib/llvm-3.8/bin/llc"
            CRYPTO="/usr/lib/x86_64-linux-gnu/libcrypto.so.0.9.8"
            TEST="$2"
            ;;
    esac
done

#Requires you have LLI variable set (I reccomend in your bash profile) to your
    LLI
#may need to chmod this script to 755
basename=`echo "$TEST" | sed 's/.*\\///
                          s/.cm//'`
./cmod.native < "$TEST" > ${basename}.ll
```

```
"$LLC" ${basename}.ll > ${basename}.s

cc -o ${basename}.exe ${basename}.s special_arith.o access.o "$CRYPTO"
./${basename}.exe
```

9.3 Testing

Listing 9.16: testall.sh

```
#!/bin/sh

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

# Path to the LLVM interpreter
# LLI="lli"

VER="3.8"
LLI="/usr/local/opt/llvm@$VER/bin/lli-$VER"

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

RUNSHELL="./cmod.sh"

FLAG=""

# Set time limit for all operations
ulimit -t 30

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

keep=0

Usage() {
    echo "Usage: testall.sh [options] [.cm files]"
    echo "-k   Keep intermediate files"
    echo "-h   Print this help"
    echo "-v   Use alternate Travis build LLI path"
    exit 1
}

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

# Compare <outfile> <reffile> <difffile>
```

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

# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    eval $* || {
   SignalError "$1 failed on $*"
   return 1
    }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
    echo $* 1>&2
    #stderr redirects to stdout, can no longe detect this
    #Still will always check for an error being thrown
    eval $* && {
   #SignalError "failed: $* did not report an error"
   #return 1
       return 0
    }
    return 0
}

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

    echo -n "$basename..."

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

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.out" &&
    Run "$RUNSHELL" "$FLAG" $1 ">" "${basename}.out" &&
    Compare ${basename}.out ${reffile}.out ${basename}.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
   if [ $keep -eq 0 ] ; then
       rm -f $generatedfiles
   fi
```

```
   echo "OK"
   echo "###### SUCCESS" 1>&2
    else
   echo "###### FAILED" 1>&2
   globalerror=$error
    fi

    rm -f *.s *.ll *.exe
}

CheckFail() {
    error=0
    basename=‘echo $1 | sed ’s/.*\\///
                            s/.cm//’‘
    reffile=‘echo $1 | sed ’s/.cm$//’‘
    basedir="‘echo $1 | sed ’s/\/[^\/]*$//’‘/."

    echo -n "$basename..."

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

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
    RunFail "$RUNSHELL" "$FLAG" "2>&1" $1 "|" "head" "-1" "|" "tee"
        "${basename}.err" ">>" $globallog &&
    Compare ${basename}.err ${reffile}.err ${basename}.diff

    # Report the status and clean up the generated files

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

    rm -f *.s *.ll
}

while getopts kdpshv c; do
    case $c in
   k) # Keep intermediate files
       keep=1
       ;;
   h) # Help
       Usage
       ;;
        v) # Test flag for Travis-CI
       LLI="/usr/lib/llvm-3.8/bin/lli"
            FLAG="-v"
       ;;
    esac
```

```
done

shift `expr $OPTIND - 1`

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

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


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

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

exit $globalerror
```

Listing 9.17: Functionality Pass and Fail Tests

```
int main()
{
  int i;
  char c;

  i = 42;
  i = 10;
  c = 'a';
  c = 'b';
  i = 'x';
}
void myvoid()
{
  return;
}

int main()
```

```
{
  int i;

  i = myvoid(); /* Fail: assigning a void to an integer */
}
int main()
{
  int i;

  i = 15;
  return i;
  i = 32; /* Error: code after a return */
}
int main()
{
  int i;

  {
    i = 15;
    return i;
  }
  i = 32; /* Error: code after a return */
}
int main()
{
  int i;
  for ( ; 1 ; ) {} /* OK: Forever */

  for (i = 0 ; i < 10 ; i = i + 1) {
    if (i == 3) return 42;
  }

  for (j = 0; i < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}
int main()
{
  int i;

  for (i = 0; j < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}
int main()
{
  int i;

  for (i = 0; i < 10 ; i = j + 1) {} /* j undefined */

  return 0;
}
int main()
{
  int i;

  for (i = 0; i < 10 ; i = i + 1) {
    foo(); /* Error: no function foo */
```

```
  }

  return 0;
}
int foo() {}

int bar() {}

int baz() {}

void bar() {} /* Error: duplicate function bar */

int main()
{
  return 0;
}
int foo(int a, int c) { }

void bar(int a, int a) {} /* Error: duplicate formal a in bar */

int main()
{
  return 0;
}
int foo(int a, int c) { }

void bar(int a, void b, int c) {} /* Error: illegal void formal b */

int main()
{
  return 0;
}
int foo() {}

void bar() {}

int printf() {} /* Should not be able to define printf */

void baz() {}

int main()
{
  return 0;
}
int foo() {}

int bar() {
  int a;
  void b; /* Error: illegal void local b */

  return 0;
}

int main()
{
  return 0;
}
void foo(int a, int b)
```

```
{
}

int main()
{
  foo(42, 1);
  foo(42); /* Wrong number of arguments */
}
void foo(int a, int b)
{
}

int main()
{
  foo(42, 1);
  foo(42, 1, 0); /* Wrong number of arguments */
}
void foo(int a, int b)
{
}

void bar()
{
}

int main()
{
  foo(42, 1);
  foo(42, bar()); /* int and void, not int and int */
}
int c;
void a; /* global variables should not be void */


int main()
{
  return 0;
}
int b;
int a;
int b; /* Duplicate global variable */

int main()
{
  return 0;
}
int main()
{
  if (1) {}
  if (0) {} else {}
  if ('c') {} /* Error: non-bool predicate */
}
int main()
{
  if (1) {
    foo; /* Error: undeclared variable */
  }
}
```

```
int main()
{
  if (1) {
    42;
  } else {
    bar; /* Error: undeclared variable */
  }
}
int main()
{
  stone x;
  stone p;

  mint m;
  mint n;

  x = "7";
  p = "21";

  m = <x, p>;
  n = m + 2;

  return 0;
}
int main()
{
  mint m;

  m = <5, 11>;

  return 0;
}
int main()
{
  mint m;

  m = <"5", "11">;

  return 0;
}
int foo() {
    return 0;
}
int main() {
    int x;
    int y;
    y = *x;
    return 0;
}
int main() {
    printf("%d\n", x, f);
    return 0;
}
void foo()
{
  if (1) return 42; /* Should return void */
  else return;
}
```

```
int main()
{
  return 42;
}
int main() {
    int i;
    for (i = 0; i < 5; i=i+1) {
        int x;
        int x; // should be error due to duplicate local
    }
}
int main() {
    int i;
    for (i = 0; i < 5; i=i+1) {
        int x;
        x = 4;
    }
    x = 3; //this should give an error, since x doesn't exist anymore
    return 0;
}
int main()
{
  int i;

  while (1) {
    i = i + 1;
  }

  while ('c') { /* Should be boolean */
    i = i + 1;
  }

}
int main()
{
  int i;

  while (1) {
    i = i + 1;
  }

  while (1) {
    foo(); /* foo undefined */
  }

}
int main()
{

   stone a;
   stone b;
   stone c;
   stone d;
   mint m;
   a = "3";
   b = "5";
   m = <a, b>;
```

```
   printf("Calling print_stone on a (=3): ");
   print_stone(a);

   printf("Calling print_stone on b (=5): ");
   print_stone(b);

   c = access_mint(m, 0);
   printf("Printing result of access_mint(m, 0), should = a: ");
   print_stone(c);

   d = access_mint(m, 1);
   printf("Printing result of access_mint(m, 1), should = b: ");
   print_stone(d);

   return 0;
}
int main()
{

   stone a;
   stone b;
   stone c;
   stone d;

   stone a1;
   stone b1;
   stone c1;
   stone d1;

   mint m;
   mint n;

   //curve cc;

   a = "3";
   b = "5";
   c = "37";
   d = "101";

   m = <a, b>;
   n = <c, d>;

   //cc = <m, n>;

   printf("Calling print_stone on a (=3): ");
   print_stone(a);

   printf("Calling print_stone on b (=5): ");
   print_stone(b);

   printf("Calling print_stone on c (=37): ");
   print_stone(c);

   printf("Calling print_stone on d (=101): ");
   print_stone(d);

   a1 = access_curve(<m, n>, 0);
```

```
   printf("Printing result of access_curve(cc, 0), should = a: ");
   print_stone(a1);

   b1 = access_curve(<m, n>, 1);
   printf("Printing result of access_curve(cc, 1), should = b: ");
   print_stone(b1);

   c1 = access_curve(<m, n>, 2);
   printf("Printing result of access_curve(cc, 2), should = c: ");
   print_stone(c1);

   d1 = access_curve(<m, n>, 3);
   printf("Printing result of access_curve(cc, 3), should = d: ");
   print_stone(d1);


   return 0;
}
int main()
{

   stone a;
   stone b;
   stone c;
   stone d;
   stone e;
   stone f;

   stone a1;
   stone b1;
   stone c1;
   stone d1;
   stone e1;
   stone f1;

   mint m;
   mint n;

   //curve cc;
   //point p;

   a = "3";
   b = "5";
   c = "37";
   d = "101";
   e = "103";
   f = "107";

   m = <a, b>;
   n = <c, d>;

   //cc = <m, n>;

   printf("Calling print_stone on a (=3): ");
   print_stone(a);

   printf("Calling print_stone on b (=5): ");
   print_stone(b);
```

```
   printf("Calling print_stone on c (=37): ");
   print_stone(c);

   printf("Calling print_stone on d (=101): ");
   print_stone(d);

   printf("Calling print_stone on d (=101): ");
   print_stone(e);

   printf("Calling print_stone on d (=101): ");
   print_stone(f);

   // p = <cc, e, f> = <<m, n>, e, f>;

   a1 = access_point(<<m, n>, e, f>, 0);
   printf("Printing result of access_point(p, 0), should = a: ");
   print_stone(a1);

   b1 = access_point(<<m, n>, e, f>, 1);
   printf("Printing result of access_point(p, 1), should = b: ");
   print_stone(b1);

   c1 = access_point(<<m, n>, e, f>, 2);
   printf("Printing result of access_point(p, 2), should = c: ");
   print_stone(c1);

   d1 = access_point(<<m, n>, e, f>, 3);
   printf("Printing result of access_point(p, 3), should = d: ");
   print_stone(d1);

   e1 = access_point(<<m, n>, e, f>, 4);
   printf("Printing result of access_point(p, 4), should = e: ");
   print_stone(e1);

   f1 = access_point(<<m, n>, e, f>, 5);
   printf("Printing result of access_point(p, 5), should = f: ");
   print_stone(f1);


   return 0;
}
int add(int x, int y)
{
  return x + y;
}

int main()
{
  printf("%d", add(17, 25));
  return 0;
}
int main()
{
  printf("%d", 39 + 3);
  return 0;
}
int main()
```

```
{
  printf("%d", 1 + 2 * 3 + 4);
  return 0;
}
int foo(int a)
{
  return a;
}

int main()
{
  int a;
  a = 42;
  a = a + 5;
  printf("%d", a);
  return 0;
}
int main() {
    stone a;
    stone b;
    stone p;

    a = "7";
    b = "26";
    p = "61";

    mint A;
    mint B;
    curve *E;

    A = <a, p>;
    B = <b, p>;
    E = <A, B>;

    print_curve(E);
    return 0;
}
int fib(int x)
{
  if (x < 2) return 1;
  return fib(x-1) + fib(x-2);
}

int main()
{
  printf("%d\n", fib(0));
  printf("%d\n", fib(1));
  printf("%d\n", fib(2));
  printf("%d\n", fib(3));
  printf("%d\n", fib(4));
  printf("%d\n", fib(5));
  printf("Done!\n");
  return 0;
}
int main()
{
  int i;
  for (i = 0 ; i < 5 ; i = i + 1) {
```

```
    printf("%d\n", i);
  }
  printf("%d\n", 42);
  return 0;
}
int main()
{
  int i;
  i = 0;
  for ( ; i < 5; ) {
    printf("%d\n", i);
    i = i + 1;
  }
  printf("%d\n", 42);
  return 0;
}
int add(int a, int b)
{
  return a + b;
}

int main()
{
  int a;
  a = add(39, 3);
  printf("%d", a);
  return 0;
}
/* Bug noticed by Pin-Chin Huang */

int fun(int x, int y)
{
  return 0;
}

int main()
{
  int i;
  i = 1;

  fun(i = 2, i = i+1);

  printf("%d", i);
  return 0;
}

void printem(int a, int b, int c, int d)
{
  printf("%d\n", a);
  printf("%d\n", b);
  printf("%d\n", c);
  printf("%d\n", d);
}

int main()
{
  printem(42,17,192,8);
  return 0;
```

```
}
int add(int a, int b)
{
  int c;
  c = a + b;
  return c;
}

int main()
{
  int d;
  d = add(52, 10);
  printf("%d", d);
  return 0;
}
int foo(int a)
{
  return a;
}

int main()
{
  return 0;
}
int a;

void foo(int c)
{
  a = c + 42;
}

int main()
{
  foo(73);
  printf("%d", a);
  return 0;
}
void foo(int a)
{
  printf("%d", a + 3);
}

int main()
{
  foo(40);
  return 0;
}
int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}

int main()
{
  printf("%d\n", gcd(2,14));
```

```
  printf("%d\n", gcd(3,15));
  printf("%d\n", gcd(99,121));
  return 0;
}
int gcd(int a, int b) {
  while (a != b)
    if (a > b) a = a - b;
    else b = b - a;
  return a;
}

int main()
{
  printf("%d\n", gcd(14,21));
  printf("%d\n", gcd(8,36));
  printf("%d\n", gcd(99,121));
  return 0;
}
int a;
int b;

void printa()
{
  printf("%d\n", a);
}

void printb()
{
  printf("%d\n", b);
}

void incab()
{
  a = a + 1;
  b = b + 1;
}

int main()
{
  a = 42;
  b = 21;
  printa();
  printb();
  incab();
  printa();
  printb();
  return 0;
}
int main() {
    printf("%s", "Hello World!\n");
    return 0;
}
int main()
{
  printf("%d\n", 42);
  printf("%d\n", 71);
  printf("%d\n", 1);
  return 0;
```

```
}
int main() {
    printf("abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefgh
    return 0;
}
int main()
{
  stone x;
  stone y;
  stone p;

  mint m;
  mint n;

  x = "15352395";
  y = "11";
  p = "65537";

  m = <x, p>;
  n = <y, p>;

  print_mint(m);
  print_mint(n);

  return 0;
}
int main()
{
  stone x;
  stone p;

  mint m;

  x = "15352395";
  p = "65537";

  m = <x + x, p>;

  print_mint(m);

  return 0;
}
int main()
{
  stone x;
  stone p;

  mint m;
  mint n;

  x = "15352395";
  p = "65537";

  m = <x, p>;
  n = m - m;
  print_mint(n);

  n = n - m;
```

```
  print_mint(n);

  return 0;
}
int main()
{
  stone x;
  stone p;

  mint m;
  mint n;

  x = "5";
  p = "11";

  m = <x, p>;
  n = m * m;
  print_mint(n);

  n = n * m;
  print_mint(n);

  return 0;
}
int main()
{
  stone x;
  stone p;

  mint m;
  mint n;

  x = "2";
  p = "11";

  m = <x, p>;
  n = m ^ m;
  print_mint(n);

  n = n ^ m;
  print_mint(n);

  return 0;
}
int main()
{
  stone x;
  stone p;

  mint m;
  mint n;

  x = "7";
  p = "21";

  m = <x, p>;
  n = m * m + m * m;
  print_mint(n);
```

```
  return 0;
}
int main()
{
  stone x;
  stone p;

  mint m;
  mint n;

  x = "3";
  p = "5";

  m = <x, p>;
  n = m ^ x;
  print_mint(n);

  return 0;
}
int main()
{
  stone x;
  stone y;
  stone p;

  mint m;
  mint n;

  x = "15352395";
  y = "11";
  p = "65537";

  m = <x, p>;
  n = <y, p>;

  print_stone(x);
  print_stone(y);
  print_stone(p);

  print_mint(m);
  print_mint(n);

  return 0;
}
int main()
{
  stone x;
  stone p;

  mint m;
  mint n;

  x = "15352395";
  p = "65537";

  m = <x, p>;
  n = m;
```

```
  print_mint(m);
  print_mint(n);

  return 0;
}
int main()
{
  printf("%d\n", 1 + 2);
  return 0;
}
int main()
{
  printf("%d\n", 100/2);
  return 0;
}
int main()
{
  printf("%d\n", 1 == 2);
  return 0;
}
int main()
{
  printf("%d\n", 1 == 1);
  return 0;
}
int main()
{
  printf("%d\n", 1 >= 2);
  return 0;
}
int main()
{
  printf("%d\n", 1 >= 1);
  return 0;
}
int main()
{
  printf("%d\n", 2 >= 1);
  return 0;
}
int main()
{
  printf("%d\n", 1 > 2);
  return 0;
}
int main()
{
  printf("%d\n", 2 > 1);
  return 0;
}
int main()
{
  printf("%d\n", 1 <= 2);
  return 0;
}
int main()
{
```

```
  printf("%d\n", 1 <= 1);
  return 0;
}
int main()
{
  printf("%d\n", 2 <= 1);
  return 0;
}
int main()
{
  printf("%d\n", 99);
  return 0;
}
int main()
{
  printf("%d\n", 1 < 2);
  return 0;
}
int main()
{
  printf("%d\n", 2 < 1);
  return 0;
}
int main()
{
  printf("%d\n", 1 * 2);
  return 0;
}
int main()
{
  printf("%d\n", 1 != 2);
  return 0;
}
int main()
{
  printf("%d\n", 1 != 1);
  return 0;
}
int main()
{
  printf("%d\n", 1 - 2);
  return 0;
}
int main() {
    stone a;
    stone b;
    stone c;
    stone d;

    stone a1;
    stone b1;
    stone p;

    a = "25";
    b = "37";
    c = "19";
    d = "8";
```

```
    a1 = "7";
    b1 = "26";
    p = "61";

    mint A;
    mint B;
    curve *E;

    A = <a1, p>;
    B = <b1, p>;
    E = <A, B>;

    point *P;
    point *Q;

    P = <E, a, b>;
    Q = <E, c, d>;

    print_point(P+Q); // should print (59, 2)
    return 0;
}
int main() {
    stone a;
    stone b;

    stone a1;
    stone b1;
    stone p;

    a = "25";
    b = "37";

    a1 = "7";
    b1 = "26";
    p = "61";

    mint A;
    mint B;
    curve *E;

    A = <a1, p>;
    B = <b1, p>;
    E = <A, B>;

    point *P;

    P = <E, a, b>;

    print_point(P+P); // should print (27, 16)
    return 0;
}
int main() {
    stone c;
    stone d;

    stone a1;
    stone b1;
    stone p;
```

```
    c = "19";
    d = "8";

    a1 = "7";
    b1 = "26";
    p = "61";

    mint A;
    mint B;
    curve *E;

    A = <a1, p>;
    B = <b1, p>;
    E = <A, B>;

    print_curve(E);

    point *Q;
    point *R;

    Q = <E, c, d>;

    d = "53";
    R = <E, c, d>;

    print_point(Q+R); // should print inf
    return 0;
}
int main() {
    stone a;
    stone b;

    stone a1;
    stone b1;
    stone p;

    a = "25";
    b = "37";

    a1 = "7";
    b1 = "26";
    p = "61";

    mint A;
    mint B;
    curve *E;

    A = <a1, p>;
    B = <b1, p>;
    E = <A, B>;

    point *P;

    P = <E, a, b>;

    print_point(b1 * P); //abuse of variables, but should print 26P = (42, 54)
    return 0;
```

```
}
int main() {
    stone a;
    stone b;

    stone a1;
    stone b1;
    stone p;

    a = "25";
    b = "37";

    a1 = "7";
    b1 = "26";
    p = "61";

    mint A;
    mint B;
    curve *E;

    A = <a1, p>;
    B = <b1, p>;
    E = <A, B>;

    point *P;

    P = <E, a, b>;

    print_point(P);
    return 0;

}
int main() {
    stone a;
    stone b;
    stone c;
    stone d;

    stone a1;
    stone b1;
    stone p;

    a = "25";
    b = "37";
    c = "19";
    d = "8";

    a1 = "7";
    b1 = "26";
    p = "61";

    mint A;
    mint B;
    curve *E;

    A = <a1, p>;
    B = <b1, p>;
    E = <A, B>;
```

```
    point *P;
    point *Q;

    P = <E, a, b>;
    Q = <E, c, d>;

    print_point(P-Q); // should print (58, 51)
    return 0;
}
int main() {
    printf("%s\n%s\n", "Hello World!", "\t o k \'\"");
    return 0;
}
int main() {
    int i;
    for (i = 0; i < 5; i=i+1) {
        int x;
        x = 3 + i*i;
        printf("%d\n", x);
    }
    int x;
    x = 6;
    printf("%d\n", x);
}

int main() {
    int x;
    x = 1;
    {
         int y;
         y = 3;
         {
             int z;
             z = 3;
             printf("%d\n", z*y+x); // prints 10
         }
         y = 4;
         printf("%d\n", y); //prints 4
    }
    x = 5;
    printf("%d\n", x); // prints 5
    return 0;
}
int main() {
    int i;
    for (i = 0; i < 5; i=i+1) {
        int x;
        x = 3 * i;
        printf("%d\n", x);
    }
    i = 10;
    printf("%d\n", i);
    return 0;
}
int main () {
    stone s1;
    stone s2;
```

```
    stone s3;
    stone s4;
    stone s5;

    s1 = "7";
    s2 = "13";
    s3 = "16";
    s4 = "20";
    s5 = "123";

    print_stone(s1);
    print_stone(s2);
    print_stone(s3);
    print_stone(s4);
    print_stone(s5);
}

int main () {
    stone s1;
    stone s2;
    stone s3;
    stone s4;

    s1 = "7";
    s2 = "8";

    s3 = s1 + s1 * s2 + s2;
    print_stone(s3);

    s3 = s1 * s1 ^ s2 * s2;
    print_stone(s3);

    s1 = "100";
    s2 = "10";
    s3 = "2";

    s4 = s1 - s2 - s3;
    print_stone(s4);
}
int main () {
    stone s1;
    stone s2;
    stone s3;
    stone s4;

    s1 = "2";
    s2 = "3";
    s3 = "5";

    s4 = s1 ^ s2 ^ s3;
    print_stone(s4);
}
int main () {
    stone s1;
    stone s2;

    s1 = "71823746";
    s2 = "131258673845432413245 34";
```

```
    print_stone(s1);
    print_stone(s2);
}

int main () {
    stone s1;
    stone s2;
    stone s3;
    stone s4;

    s1 = "148";
    s2 = "2017";
    s3 = s1 + s2;
    s4 = s2 - s1;

    print_stone(s1);
    print_stone(s2);
    print_stone(s3);
    print_stone(s4);
}

int main () {
    stone s1;
    stone s2;
    stone s3;
    stone s4;

    s1 = "14854673828914735847914672361";
    s2 = "20178241273416559324867214678259364 7";
    s3 = s1 + s2;
    s4 = s2 - s1;

    print_stone(s1);
    print_stone(s2);
    print_stone(s3);
    print_stone(s4);
}
int main () {
    stone s1;
    stone s2;
    stone s3;
    stone s4;

    s1 = "12";
    s2 = "4";
    s3 = s1 * s2;
    s4 = s2 * s1;

    print_stone(s1);
    print_stone(s2);
    print_stone(s3);
    print_stone(s4);
}
int main () {
    stone s1;
    stone s2;
    stone s3;
```

```
    stone s4;

    s1 = "426137928471234912374619234 5";
    s2 = "234817894231762341676742176 6";
    s3 = s1 * s2;
    s4 = s2 * s1;

    print_stone(s1);
    print_stone(s2);
    print_stone(s3);
    print_stone(s4);
}
int main () {
    stone s1;
    stone s2;
    stone s3;
    stone s4;

    s1 = "5";
    s2 = "0";
    s3 = "1";

    s4 = s1 * s2;
    print_stone(s4);

    s4 = s1 * s3;
    print_stone(s4);
}
int main () {
    stone s1;
    stone s2;
    stone s3;
    stone s4;

    s1 = "4";
    s2 = "2";

    s3 = s1 ^ s2;
    print_stone(s3);

    s2 = "3";
    s3 = s1 ^ s2;
    print_stone(s3);
}
int main () {
    stone s1;
    stone s2;
    stone s3;

    s1 = "426137928471234912374619234 5";
    s2 = "8";
    s3 = s1 ^ s2;

    print_stone(s3);
}
int main()
{
   stone a;
```

```
   stone b;

   a = "123";
   b = "456";

   printf("%d\n", a < b);
   printf("%d\n", a == b);
   printf("%d\n", a != b);
   printf("%d\n", a > b);
   printf("%d\n", a >= b);
   printf("%d\n", a <= b);
}
int main()
{
    //test unop NEG on int types
    int x;
    x = 45;
    printf("%d\n", -x);
    return 0;
}
int main()
{
    // test for NOT on int types
    int x;
    int y;

    x = 4;
    y = 0;

    printf("%d\n", !x);
    printf("%d\n", !y);

    return 0;
}
int main()
{
  int a;
  a = 42;
  printf("%d", a);
  return 0;
}
int a;

void foo(int c)
{
  a = c + 42;
}

int main()
{
  foo(73);
  printf("%d", a);
  return 0;
}
int main()
{
  int i;
  i = 5;
```

```
  while (i > 0) {
    printf("%d\n", i);
    i = i - 1;
  }
  printf("%d\n", 42);
  return 0;
}
int foo(int a)
{
  int j;
  j = 0;
  while (a > 0) {
    j = j + 2;
    a = a - 1;
  }
  return j;
}

int main()
{
  printf("%d", foo(7));
  return 0;
}
int main() {
    stone a;
    stone b;
    stone c;
    stone d;

    stone a1;
    stone b1;
    stone p;

    a = "25";
    b = "37";
    c = "19";
    d = "8";

    a1 = "7";
    b1 = "26";
    p = "61";

    mint A;
    mint B;
    curve *E;

    A = <a1, p>;
    B = <b1, p>;
    E = <A, B>;

    point *P;
    point *Q;

    P = <E, a, b>;
    Q = <E, c, d>;

    print_point(P+Q); // should print (59, 2)
    print_point(P); //should print original P <25,37>
```

```
    return 0;
}
#ifndef __MYADD_CM__
#define __MYADD_CM__

int myadd(int a, int b) {
    return a + b;
}

#include "myadd.cm" //buildguards plz
#endif
#include "myadd.cm"

int main() {
    printf("%d\n", myadd(5, 6));
    return 0;
}
int main() {
    char *a;

    a = malloc(10);
    scanf(a);
    printf("%s\n", a);

    return 0;
}
```

Listing 9.18: Grammar Pass and Fail Tests

```
int main() {
    mint m;
    access m;
}

int main() {
    mint m;

    access m + 2;
    access m * 2;
}

int main() {
    int x;

    x = 1 + 2 + 3;
}

int main() {
    char c;
    c = 'x';
}

int main() {
    char c;
    c = '\n';
}

int main() /* This is the main function */ {
```

```
    int x; /* Int declaration */
} /* That was a cool test, huh? */

int main() {
    mint m1;
    mint m2;
    curve c;

    c = <m1, m2>;
}

int main() {
    int x;

    x = 6 / 2 - 6 / 3;
}
int main() {
    int x;
    x = 0;

    do {
        x = x + 1;
    } while (x < 5)
}

int main() {
    int i;
    int x;

    x = 0;

    for (i = 0; i < 10; i = i+1) {
        x = x + 2;
    }
}

int main() {
    int x;

    if (5 > 3) {
        x = 1;
    } else {
        x = 0;
    }
}

int main() {
    int x;
    x = 0;

    x = (1 && !x) || 0;
}

int main() {
    int x;
    int y;

    x = 1;
```

```
    y = 0;

    x = (x > y);
    y = 0 || 1 && 0;
}

int main() {
    int x;
    int y;

    x = 1;
    y = 0;

    x = (1 == 0 < 7);
    y = (5 <= 5 != 0);
}

int main() {
    mint m;
    m = <5, 7>;
}

int main() {
    mint m;
    m = <1 || 0, 3 ^ 2>;
}

int main() {
    stone s1;
    stone s2;
    mint m;

    m = <s1, s2>;
}

int main() {
    int x;
    x = 20;

    x %= 2 ^ 2 - 1;
}

int main() {
    int x;

    x = 6 % 4 * 7 - 14;
}

int main() {
    int x;

    x = 1 * 2 + 3 * 4;
}

int main() {
    int x;
    int y;
```

```
    x = -2 * 10;
    y = -x;
}

int main() {
    int x;

    x = (1 + 2) * (6 / (3 - 2));
}

int main() {
    curve c;
    point p;

    p = <c, 5, 6>;
}

int main() {
    curve c;
    point pInf;

    pInf = <c, ~>;
}

int main() {
    int x;
    int y;
    int z;

    z = &x ^ *y;
}

int main() {
    int x;

    x = 6 * 2 ^ 2 / 8;
}

int main() {
    stone x;
    x = 5;
}

int main() {
    stone x;
    x = 9999999999999999;
}

int main() {
    char *arr;

    arr = "Hello, I am a test string!";
}

int main() {
    int x;

    x = 3 - 2 - 1;
```

```
}

int main() {
    int x;
    x = 0;

    while (1) {
        if (x = 1) {
            x = 2;
            continue;
        }

        x = x + 1;
        if (x = 10) {
            x = x * 2;
            break;
        }
    }
}

int main() {
    int x;
    x = 0;

    while (x < 5) {
        x = x + 1;
    }
}
```

9.4 Libraries

9.4.1 ElGamal Encryption

Listing 9.19: alice-decrypt.cm

```
int main() {
    stone g_div;
    stone h_div;
    stone p;
    mint g;
    mint h;

    //alice's private key
    stone y;
    stone y_neg;
    y = "131";
    y_neg = "-131";

    //public keys
    p = "977";
    g_div = "3";
    g = <g_div, p>;
    h = g^y;


    //shared secret g^xy
    mint s;
```

```
    char *x;
    x = malloc(100);
    int msg_len;
    int i;
    scanf(x);
    msg_len = atoi(x);
    for (i = 0; i < msg_len; i = i + 1) {
        scanf(x);
        stone t_div;
        mint t;
        t_div = x;
        t = <t_div, p>;
        s = t^y_neg;

        scanf(x);
        stone z_div;
        mint z;
        z_div = x;
        z = <z_div, p>;
        print_div(z * s);
    }
    return 0;
}
```

Listing 9.20: bob-encrypt.cm

```
int main() {
    stone g_div;
    stone h_div;
    stone p;
    mint g;
    mint h;

    //public keys
    p = "977";
    g_div = "3";
    h_div = "249"; //alice's g^x for her secret x
    g = <g_div, p>;
    h = <h_div, p>;

    //bob's private key
    stone y;
    y = "77";

    //shared secret g^xy
    mint s;
    s = h^y;

    char *x;
    x = malloc(100);
    int msg_len;
    int i;
    scanf(x);
    msg_len = atoi(x);
    printf("%s\n", x);
    for (i = 0; i < msg_len; i = i + 1) {
        scanf(x);
        stone z_div;
```

```
        mint z;
        z_div = x;
        z = <z_div, p>;

        print_div(g^y);
        print_div(z * s);
    }
}
```

9.4.2 Standard Diffie Hellman

Listing 9.21: alice-dh.cm

```
int main()
{
  //Decls
  stone alice_g;
  stone alice_p;
  stone bob_div;

  stone alice_sec;

  mint alice_gap;
  mint bob_gap;
  char *bob_gap_input;

  alice_p = "153";
  alice_g = "161";

  alice_sec = "6";

  alice_gap = < (alice_g^alice_sec) , alice_p >;
  //Send p to Bob
  print_stone(alice_p);
  //Send g to Bob
  print_stone(alice_g);
  //print_stone(alice_g);
  print_div(alice_gap);

  //Send and recieve gaps
  //Allice recieves first
  bob_gap_input = malloc(100); //free this!
  scanf(bob_gap_input);
  //printf("SYM: %s\n", bob_gap_input);
  bob_div = bob_gap_input;


  bob_gap = <bob_div, alice_p>;

  print_div(bob_gap^alice_sec);

  return 0;

}
```

Listing 9.22: bob-dh.cm

```
int main()
{
```

```
  stone alice_g;
  stone alice_p;
  stone alice_div;

  stone bob_sec;

  mint bob_gap;
  mint alice_gap;
  char *alice_gap_input;

  char *alice_p_input;
  char *alice_g_input;

  bob_sec = "54";

  alice_p_input = malloc(100);
  alice_g_input = malloc(100);

  scanf(alice_p_input);
  scanf(alice_g_input);


  alice_gap_input = malloc(100);
  scanf(alice_gap_input);

  alice_div = alice_gap_input;
  //printf("%s\n %s \n", alice_p_input, alice_g_input);

  alice_p = alice_p_input;
  alice_g = alice_g_input;
  //print_stone(alice_p);
  //print_stone(alice_g);

  bob_gap = < (alice_g^bob_sec) , alice_p >;

  //Send and recieve gaps
  print_div(bob_gap);


  alice_gap = <alice_div, alice_p>;

  print_div(alice_gap^bob_sec);

  return 0;

}
```

9.4.3 Eliptic Curve Diffie Hellman

Listing 9.23: alice-dh-ec.cm

```
int main() {
    curve *E;
    mint A;
    mint B;

    stone a;
    stone b;
    stone p;
```

```
    stone t;

    a = "7";
    b = "26";
    p = "61";

    A = <a, p>;
    B = <b, p>;
    E = <A, B>;

    point *P;
    point *Q;

    a = "25";
    b = "37";

    P = <E, a, b>;
    t = "13"; // alice's private key
    P = t * P;

    print_point_sep(P); //send it over

    char *x;
    char *y;
    x = malloc(100);
    y = malloc(100);

    scanf(x);
    scanf(y);

    a = x;
    b = y;

    Q = <E, a, b>;

    print_point(t * Q);
    return 0;
}
```

Listing 9.24: bob-dh-ec.cm

```
int main() {
    curve *E;
    mint A;
    mint B;

    stone a;
    stone b;
    stone p;
    stone t;

    a = "7";
    b = "26";
    p = "61";

    A = <a, p>;
    B = <b, p>;
```

```
    E = <A, B>;

    point *P;
    point *Q;

    a = "25";
    b = "37";

    P = <E, a, b>;
    t = "23"; // bob's private key
    P = t * P;

    print_point_sep(P);// send it over

    char *x;
    char *y;

    x = malloc(100);
    y = malloc(100);

    scanf(x);
    scanf(y);

    a = x;
    b = y;

    Q = <E, a, b>;

    print_point(t * Q);
    return 0;
}
```