Final Language Report: Prime

Alexander Liebeskind (al3853), *Project Manager*
Nikhil Mehta (nm3077), *Language Guru*
Pedro B T Santos (pb2751), *Tester*
Thomas Tran (tkt2120), *Systems Architect*

Programming Languages and Translators, Spring 2021

Contents

1 References

We cite the following sources and thank the respective owners:

1. https://gmplib.org/manual/ The GMP library for large number functions
2. Professor Dorian Goldfeld slides for numbers and presentation snapshots
3. http://www.christelbach.com/ECCalculator.aspx to find some example prime numbers for our demos.

2 Introduction

2.1 Overview

PRIME is a programming language specifically created for the implementation of cryptography algorithms. The main features of PRIME aim to facilitate the implementation of encryption and decryption schemes. Since modular arithmetic, large numbers, and elliptic curve processing are common in cryptography, the main types and operators in PRIME address these topics. PRIME also includes more general basic features for ease of usage.

2.2 Background

The basis for all modern cryptography is mathematical trapdoor functions. That is, mathematical operations that are easy to do in one direction but very difficult to do in the backwards (inverse) direction. Modular arithmetic using large primes is at the heart of many trapdoor functions used in industry today so our language will focus on making this kind of arithmetic easy to implement. Additionally, the recent advent and widespread adoption of elliptic curve cryptography is changing how we encrypt data. Trusted protocols are now being re-implemented more securely with the use of modular elliptic curves. So, building off of our use of big numbers our language will make elliptic curve operations a core feature.

Modular Arithmetic

Modular arithmetic, sometimes referred to as clock arithmetic, is a mathematical system for integers that looks at the remainder of standard mathematical operations for a given modulus. For example, 13 divided by 6 is equal to 2 reminder 1. In modular arithmetic we only care about the remainder so we say 13 is congruent to 1 modulo 6 or 1 mod 6 for short. The notation for this is:

$$13 \equiv 1 \pmod{6}$$

We can then build up from here to denote entire mathematical expressions as being modded by a number, such as:

$$2^4 \equiv x \pmod{5}$$

In this case we find that x is congruent to 1. Modular arithmetic is really easy to compute in one direction. But take the expression:

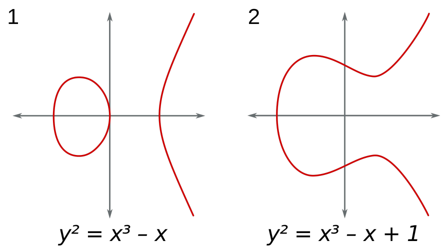
$$3^x \equiv 4 \pmod{7}$$

It's much harder to find x now. When we replace these single digit integers with massive numbers, say around 200 digits long, this problem becomes impossible even for a computer to solve.

Elliptic Curves

An elliptic curve is defined as the function

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



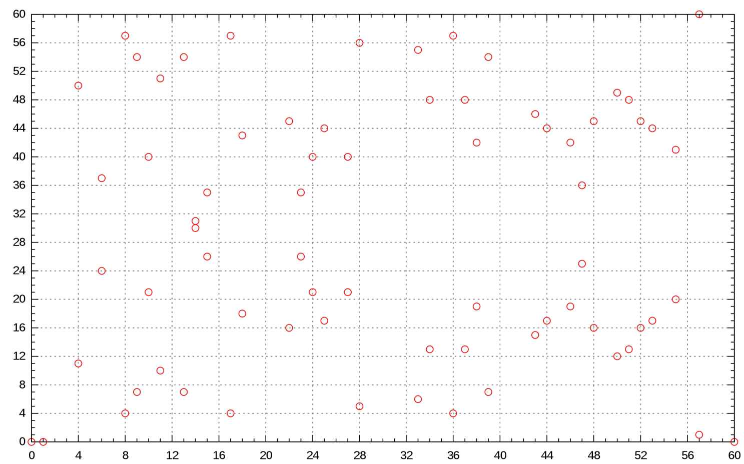
source: Wikimedia Commons

We then define the operation point addition. For points p, q, their sum $r = p + q$ is taken as the third point of intersection on the curve with the line formed by points p and q and then reflected about the x-axis. Lines that are tangent to the curve count as intersecting the line twice at the point of tangency. In elliptic curve cryptography we define a curve over a modulus.

$$y^2 = x^3 + ax + b \pmod{c}$$

The instead of working with an infinite set of points over the $\mathbb{R} \times \mathbb{R}$ we now have a finite set of integer points. When c is a prime number this set forms a finite abelian group, which is therefore cyclic over the point addition with a generator point g. That is a complicated way is saying a few things. First, for any points p and q, $p + q = q + p$. Second, there exists a point g where the repeated addition of g onto itself will yield every point on the modular elliptic curve. Third, since a modular elliptic curve forms a group, there is an identity element. This element is the point at infinity. The point at infinity added to any point p on the curve will yield p. Lastly, for every p on the curve, there exists an inverse denoted $-p$ such that $p + -p$ is equal to the point at infinity.

With the use of computers and calculators, computing the repeated sum of a point p by k number of times times, denoted $q = k * p$ is easy. But, if given only points q and p, it can become impossibly difficult to find k. This is how trapdoor functions are built using elliptic curves and where they become useful in cryptographic contexts.



The set of points in $y^2 = x^3 - x \pmod{61}$
source: Wikimedia Commons

2.3 Related Work

Due to syntactical and functional similarities between PRIME and C, the C Reference Manual (`https://www.bell-labs.com/usr/dmr/www/cman.pdf`) is a helpful resource in understanding PRIME and its underlying mechanics.

2.4 Goals

2.4.1 Big Number and Modular Arithmetic

The foremost goal of PRIME is implement features related to modular arithmetic of big numbers. To accomplish this we wrapped the GMP GNU Big Number Library with our language. This library is commonly used in cryptography applications today, however, the GMP GNU library is unintuitive to use and clunky to write. One of PRIME's largest benefits is the ease in which a user can declare and then work with a large number. While libraries like GMP GNU exclusively rely on the use of verbose function calls, PRIME utilizes built in operators and a simplified syntax. Building off of big numbers, one of PRIME's core functionalities is the support of elliptic curves and elliptic curve arithmetic. Elliptic curves are not built into standard big number libraries. So a function like adding two points on an elliptic curve which took around 200 lines to write in the GMP GNU big number library has been reduced to a single character operator in PRIME. This allows for commonly used cryptosystems and protocols to be implemented concisely in just a few lines of code.

2.4.2 Ease of Usage and Readability

As eluded to above, PRIME seeks to make arithmetic with big numbers more intuitive. As a result, we sought to make our arithmetic operators and expressions closely resemble standard mathematical notation so a user with little experience could start programming in PRIME on the fly. Below are a couple comparisons between mathematical notation and their PRIME implementations.
Multiplicative inverse of a modulo b
Mathematical notation: $a' \pmod b$

```
a`b /* PRIME implementation of Multiplicative inverse of a modulo b */
```

So, in addition to *a raised to b modulo c*
Mathematical notation: $a \hat{} b \pmod c$

```
a ^ b @ c /* PRIME implementation of a raised to b modulo c */
```

So, in addition to allowing for concise implementations cryptographic functions, PRIME makes code intelligible for other users. This is made possible by complex algorithms and functions that we wrote or wrapped behind the scenes. If a user chose they would have to learn many of these algorithms and implement them themselves.

3 Language Tutorial

For installation, unzip the targ.gz file, then run make to build and run the regression test suite. Run this in a docker container that corresponds to Professor Edward's microc docker hub container. Make will install the gmp library via apt for you.
The first thing to write is the main function as this will be the entrypoint to the program. This should return 0 if all is successful as is convention like so:

```
int main()
{
    return 0;
}
```

To then start adding some interesting functionality, one can dive in and start using large integers (lints).

```
/* Inside a function */
lint a;
/* This or larger and terminated by an l */
a = 123098123081283012312312312312312312312312312312312l;
/* Now you can print them and their operations */
printl(a);
printl(a*21); /* note that these must be the same type */
```

One can then use these lints to declare curves where our points will then lie.

```
lint a;
lint b;
curve c;

a = 31;
b = 21;
c = [(11, b) : a];
printc(c);
```

Now to start getting to more advanced functionality, we combine lints and curves to create points.

```
/* At very start of function */
pt p;
/*
.
.
.....sometime later */
/* Following on from code in previous box */
p = [a, b] & c;
printpt(p);
printpt(p + p); /* Point addition on elliptic curve */
```

Now for familiar cryptography applications you may want to have some strings there too

```
string s;
lint a;
s = "Hello World";
a = encode(s); /* turns string into its numerical equivalent */
printl(a);
prints(decode(a)); /* decodes the string */
```

Then you can wrap some logical statements together into a function:

```
/* function returning a new lint */
lint newLint(string s)
{
    return encode(s);
}
```

Several other code examples are presented throughout the course of the report.

4 Language Manual

4.1 Types

Types in PRIME are similar to that of C and C++ with a few others that are particularly useful for cryptography operations. Note that in the below CFG, the term constant is used to refer to fundamental literal types, which are then used to create all of the basic types in PRIME.

4.1.1 Initialization

In the absence of variable assignment, types are handled as literals (here 'constants', in the CFG). Initializing variables in PRIME is similar across all types, requiring the type of the variable to be declared before initialization with the assignment operator. All declaration must occur on its own lines, before any variable is set to a value.

Examples:

```
int x; /* Declaration x to be of type int*/
x = 4; /* Assign 4 to x */
int y; /* Parsing error! */
```

4.1.2 int

A `int` can take signed 32-bit integer values ranging from $-2,147,483,648$ to $2,147,483,647$. `int`s are declared without an original value. After initialization, `int` can then be set to a value. The value of the int may be reassigned, set using an expression, or set to the value of another int variable. The value of an uninitialized int is undefined, and is a garbage number.

Examples:

```
    int foo;
    int bar;

    foo = 4;       /* Assign 4 to x */
    foo = 6 * 5;   /* Reinitialize foo */
    bar = foo;     /* Assign bar to foo's value */
    print(bar);    /* prints 30 */
```

4.1.3 string

A string is a datatype meant to hold a sequence of ASCII characters. We denote string assignment by using quotations for literal sequences of characters. strings can be declared with or without an original value. The value of an uninitialized string is undefined, and is a garbage string. After initialization, a string can be reassigned, set using a string literal, or set to the value of another string variable.

Examples:

```
    int foo;
    int bar;

    foo = "test1";
    foo = "test2";
    bar = foo;      /* Assign bar to foo's value */
    prints(bar);    /* prints test2 */
```

4.1.4 lint

lints are the primary type of the PRIME language and serve as the building blocks for all of our cryptographic types and the operands in much of our arithmetic. A lint or large integer is a positive or negative integer with minimal restraints on size. A lint can hold up to 2^{32} bits to conform with current state-of-the-art cryptographic security requirements. Large integer types are usually used to hold large primes for later use in computationally expensive products or exponents in practice.

Similar to other types, lints must be declared first, and then initialized. When initializing, digits are followed by an 'l' to distinguish between integers and large integers. ints may also be cast into lints using tolint(), but not vice versa.

Examples:

```
    lint foo;
    lint bar;
    int x;

    foo = 10027100271002710027100271;
    x = 100;

    bar = tolint(x);

    printl(bar); /* prints 100 */
```

4.1.5 curve

curve defines a third degree univariate polynomial function with respect to a positive lint modulus. Within elliptic curve cryptography, all curves are defined as

$$y^2 \equiv x^3 + c_1x + c_2 \ (mod\ n)$$

Thus, curves in PRIME only need to define in c_1 and c_2 and a base modulus n. In PRIME curve literals are written encased in brackets. First, The two coefficients are written inside parenthesis and separated by a comma. This is followed by a colon and then the modulus. More simply put, the curve above can be written as

$$[(c_1, c_2) : n]$$

in PRIME. Both the coefficients and the modulus must be of type `lint`. `curves` are immutable so once initialized, none of their values may be changed. However, `curves` can be reassigned. `curves` are used as a building block for points and thus have no arithmetic nor relational operators defined for them.

Much of modern day elliptic curve cryptography is based on elliptic curves being defined over a prime modulus. Primality testing is an entire sub-field of mathematics and for large numbers, most tests can only give high likelihoods of numbers being prime, not a guarantee. So onus is on the user to pass in a valid prime number.
Examples:

```
curve bar;

bar = [(5l, 12l) : 13l];
printc(bar);            /* prints [(5, 12) : 13]; */
bar = [(3l, 7l) : 23l]  /* bar reassigned */
```

4.1.6 pt

A `pt` is a datatype meant to represent a point on an elliptic curve. When points are initialized, they must be defined with regards to a `curve`. Additionally, they may also specify two coordinates of `lint` types, representing the x and y coordinates.

When a `pt` is initialized with a set of immutable coordinates, a value must be given for every dimension. These values must be enclosed in brackets and separated by commas. A `pt` declared without 2 `lints` is undefined. Furthermore, they must be defined with respect to a `curve` using `&`. See examples below.

The coordinates within a `pt` can be accessed using `pt.x` and `pt.y`; however, the coordinates cannot be reinitialized. Since `pts` are specific to an elliptic curve, a `pt`'s `curve` may not be changed. However, a `pt` may be reassigned.

A given modular elliptic curve has a set of valid points associated with it. Building safe elliptic curves and generator points for those curves is a feat in it of itself so the onus is on the user to use valid points for a given elliptic curve. This includes using coordinates that are non-negative `lints` less than the elliptic curve modulus. The one exception to this rule is the point (-1, -1) which is written as `[-1l, -1l] & crv` for a given `curve crv`. This represents the point at infinity. The point at infinity exists with respect to any curve and is the identity element for point arithmetic.

Examples:

```
curve bar;
pt foo;

bar = [(5l, 12l) : 13l];
foo = [2l, 3l] & bar; /*x = 2, y = 2, defined on the bar curve*/

printl(foo.x);        /* prints 2 */
printl(foo.y);        /* prints 3 */
printpt(foo);         /* prints [2, 3] & [(5, 12) : 13] */

foo = [-1l, -1l] & [(7l, 2l) : 23l];
printpt(foo)          /* prints [-1, -1] & [(7, 2) : 23] */

foo.x = 4             /* Parsing error */
```

4.2 Operators

4.2.1 Unary Operators

Unary operators include ! and -. Unary operators take precedence above all other mathematical operators.

Not: !

$$!\, expression$$

The logical negation operator obtains the logical opposite of a value. It works on `int`s and `lint`s. Since there are no booleans in PRIME, true and false may be represented by 0 and non-zero integers respectfully. If the expression evaluates to a non-zero integer, $!\, expression$ will return a 0. If the expression evaluates to zero, $!\, expression$ will return 1. The same applies for `lint`s, returning another lint.

Examples:

```
int foo;
lint bar;
lint baz;

foo = 1;
bar = 0l;
baz = !bar;
print(!foo);                /* prints 0 */
if (baz) {                  /* evaluates to true */
    print("baz is true");
}
```

Negative: -

$$-expression$$

The negative unary operator is used to obtain the mathematical opposite of a value. It works on `int`s, `lint`s, and `pt`s. When performed on an `int` or `lint`, the opposite value will be returned. For example, 1 becomes -1 and vice versa. When performed on an `pt`, it is the additive inverse of the `pt` with respect to its elliptic curve. The sum of a point and its additive inverse is always the point at infinity (-1, -1), which serves as the identity element in modular elliptic curves. The point at infinity can be written in PRIME as `[-1l, -1l] & crv` for a given `curve crv`.
Examples:

```
int foo;
int bar;
pt baz;

foo = 1;
bar = -foo;
print(bar);                              /* prints -1 */
baz = [2l, 2l] & [(5l, 12l) : 13l];
printpt(-baz);                           /* prints additive inverse of foo
                                          * which is [2l, 11l] & [(5l, 12l) : 13l] */
```

4.2.2 Multiplicative Operators

Multiplicative operators include multiplication, division, modulo, power, and the multiplicative invert operator. Multiplicative operators have a ranges of precedences which can be viewed in following section.

Multiplication: *

$$expression * expression$$

The multiplication operator is used to obtain the product of two numbers. The multiplication operator functions on `int`s, `lint`s, and `pt`s. `int`s may only be multiplied by other expressions that evaluate to `int`s. Multiplication of `lint`s by `lint`s and `int`s by `int`s return their same type. When multiplication is performed on a `pt`, the other expression must be a positive `lint`. Multiplication of a `pt p` by `lint l` and vice versa acts as a repeated addition of `pt p` onto itself `l` times. If `l` is negative the behavior is undefined. `pt`s may not be multiplied by other `pt`s. `pt` multiplication returns a `pt`.
Examples:

```
lint foo;
lint bar;
int three;
int four;
pt baz;

one = 2147483659l;
two = 1837641987193287l;
foo * bar;                               /* is a lint */

three = 3;
four = 4;
three * four;                            /* is an int */

baz = [2l, 2l] & [(5l, 12l) : 13l];
baz * 3l;                                /* evaluates to: baz + baz + baz + baz */
three * baz;                             /* same as line above */
```

Division: /

expression / *expression*

The division operator will be used to obtain the integer quotient of two integers. It will function on `int`s and `lint`s. `int`s may only be divided by or divide expressions of type `int`. Similarly, `lint`s may only be divided by or divide expressions of type `lint`. If the divisor, the right expression, does not divide the left expression, the quotient will be truncated. Division returns the same type as the two operands in the expression.
Examples:

```
lint bar;
int three;
int four;

bar = 1837641987193287l;
2147483659l/bar;                     /* is a lint */

three = 3;
four = 4;
four/three;                          /* is an int */
```

Modulo: %

expression % *expression*

The modulo operator will be used to obtain the positive remainder from euclidean integer division. It will function on `int`s and `lint`s. `int`s may only be modded by other `int`s and `lint`s by other `lint`s. The return type is the same as the type of the two operand expressions.
Examples:

```
lint foo;
lint bar;
int three;
int four;

foo = 2147483659l;
bar = 1837641987193287l;
bar%foo;                             /* is a lint */

three = 3;
four = 4;
four%three;                          /* evaluates to 1 as an int */
```

Exponent: /\

expression/*expression*

The exponent operator will be used to compute the repeated multiplication of integer values. The exponent operator is written as forward slash immediately followed by a backslash. The base expression

must be of type `lint` and the exponent expression must be of type `int`. The exponent expression may negative but the output will truncate to zero. The return type is always a `lint`.
Examples:

```
lint foo;
int bar;

foo = 10000;
bar = 2;
bar/\foo;                    /* is a lint */
```

Multiplicative Inversion: ‘

$$expression \text{ ‘ } expression$$

The multiplicative inversion operator is used to find the multiplicative inverse of the left expression modulo the right expression. A multiplicative inverse of $a(mod\ b)$ is the smallest positive integer c such that $a * c$ is congruent to $1(mod\ b)$. The multiplicative inversion operator is denoted with the backward apostrophe typically found just left of the '1' key on American keyboards. The multiplicative inverse is found efficiently using the extended euclidean algorithm. Both operand expressions must be of type `lint` and the resultant output value will also be a `lint`. Not every integer has a multiplicative inverse modulo another integer. If that is the case the output will be 0. 0 is never the multiplicative inverse of any integer.
Examples:

```
lint a;
lint b;
lint c;
lint ;

a = 3l;
b = 7l;

             /* c is equal to 5
              * 5*3 = 15
              * 15 is congruent to 1 (mod 7) */
c = a ‘ b;

printl(c);   /* prints "5" */

d = 2l ‘ 4l  /* 2 has no multiplicative inverse mod 4 */
printl(d);   /* prints "0" */
```

4.2.3 Additive Operators

Additive operators include addition and subtraction and have lower precedence than multiplicative operators.

Addition: +

$$expression + expression$$

The addition operator will be used to obtain integer and point sums. It will function on `int`s, `lint`s, and `pt`s. `int`s may only be added to `int`s and `lint`s to `lint`s.
`pt` addition is always defined with respect to a specific elliptic curve and so `pt`s may only be added to other `pt`s of the same `curve`. If two `pt`s of different `curve`s are added to one another the behavior is undefined. The identity element in elliptic curve point addition is the point at infinity. In our language we denote this point as (-1, -1). In PRIME this is written as `[-1l, -1l] & crv` for a given `curve` `crv`. Any `pt` on `curve crv` added to the point at infinity will return itself. A `pt` added with its additive inverse will return the point at infinity.
Addition always returns the same type as the two operands in the expression.
Examples:

```
lint one;
lint two;
int three;
```

```
    int four;
    pt baz;
    pt foo;

    one = 2147483659l;
    two = 1837641987193287l;
    one + two;                          /* is a lint */

    three = 3;
    four = 4;
    three + four;                       /* is an int */

    baz = [2l, 2l] & [(5l, 12l) : 13l];
    foo = -baz;                         /* set foo to the additive inverse of baz */
    baz + baz;                          /* is a pt */
    baz + [-1l, -1l] & [(5l, 12l) : 13l]; /* evaluates to baz */
    baz + foo;                     /* evaluates to [-1l, -1l] & [(5l, 12l) : 13l] */
```

Subtraction: -

expression - *expression*

The subtraction operator is used to obtain integer differences. It will function on `int`s and `lint`s. `int`s may only be subtracted with other `int`s and `lint`s may only be subtracted from other `lint`s. Subtraction returns the same type as the two operands in the expression.
Examples:

```
    lint foo;
    int three;

    foo = 2147483659l;
    foo - 1837641987193287l;            /* is a lint */

    three - 4;                          /* is an int */
```

4.2.4 Relational Operators

$$expression < expression$$

$$expression > expression$$

$$expression <= expression$$

$$expression >= expression$$

The relational operators (`<`, `>`, `<=`, `>=`) denote less than, greater than, less than or equal to, and greater than or equal to, respectively. These operators compare two expressions of type `int` or `lint` and will return an integer value 1 (true) or 0 (false) based on a relational comparison in $\mathbb{R}$. These relational operators are equivalent in precedence and have a lower precedence than multiplicative and additive operators.
Examples:

```
    lint foo;
    lint bar;
    int three;

    foo = 2147483659l;
    bar = 1837641987193287l;
    bar > foo;                        /* returns 1 as an int */

    three = 3;
    three <= -15;                     /* returns 0 as an int */
```

4.2.5 Equality Operators

$$expression == expression$$

$$expression\ != expression$$

The equality operators (`==`, `!=`) denote equals and not equals, respectively. These operators compare two expressions of type `int`, `lint`, or `pt`. The `==` operator returns an integer value 1 (true) if the two expressions are equivalent by value, and 0 (false) otherwise. `!=` operator returns an integer value 1 (true) if the two expressions are **not** equivalent by value, and 0 (false) otherwise. Two `pts` on the different elliptic curves will never be equal to one another regardless of their coordinates. Two `pts` on the same elliptic curve may be equal if their respective coordinates are equal. These equality operators are equivalent in precedence to one another and have a lower precedence than relational operators. A `lint` may only be compared to another `lint`, `ints` only with other `ints`, and `pts` only with other `pts`. *Examples*:

```
lint foo;
lint bar;
int three;
pt baz;

foo = 2147483659l;
bar = 1837641987193287l;
bar != foo;                            /* returns 1 as an int */

three = 3;
three == 3;                            /* returns 0 as an int */

baz = [2l, 2l] & [(5l, 12l) : 13l];
baz == [2l, 2l] & [(5l, 12l) : 13l]; /* returns 1 as an int */
baz != [2l, 2l] & [(5l, 7l) : 13l];  /* returns 1 as an int */
```

4.2.6 Logical Operators

$$expression\ \&\&\ expression$$

$$expression\ ||\ expression$$

The logical operators (`&&`, `||`) denote logical and and logical or, respectively. These operators compare two expressions, both of type `int` or `lint`. The `&&` operator returns an integer value 1 (true) if the two expressions are nonzero, and 0 (false) otherwise. The `||` operator returns an integer value 0 (false) if the two expressions are zero, and 1 (true) otherwise. These logical operators are evaluated from left to right, equivalent in precedence and have a lower precedence than equality operators. A logical expression must only have types that are all `ints` or all `lints`. *Examples*:

```
lint foo;
lint bar;
int three;

foo = 2147483659l;
bar = 0l;
bar && foo;                            /* returns 0 as an int */

three = 3;
three || 0 || 1;                       /* returns 1 as an int */

1 || (bar || foo)                      /* returns 1 as an int */
```

$$variable = expression$$

The equals sign is the assignment operator in our language. Once the right hand expression has been evaluated, it is stored in the variable. The expression to the left of the equals sign must be a variable of the same type as the expression. Assignment must happen in a separate statement to that of variable declaration. An assignment expression returns void. *Examples*:

```
lint bar;
int three;
pt baz;


baz = [21, 21] & [(51, 121) : 131];
bar = 01;
three = 3;
```

4.2.7 Ternary Operators

In PRIME there are two ternary operators that can be used to form a single ternary operation. The ternary operators take precedence above all other arithmetic, relational, equality, and logical operators.

$$expression \;\hat{}\; expression \;@\; expression$$

The ternary operation using the carrot and then the at symbol functions only on `lints` and is taken to mean value of the first expression raised to the power of the middle expression modulo the right expression. Or, more simply put, $a^b\%c$ where a and b are the first two expressions respectively and c is the last expression. The return type of ternary operation is a `lint`. The tenary operator is the only explicit way to raise a `lint` to the power of another `lint`.
Examples:

```
lint a;
lint b;
lint c;

a = 1234567891;
b = 9876543211;
c = 231

a ^ b @ c /* mathematically the same as (a^b)%c */
```

4.3 Precedence

Precedence in PRIME is defined as follows. If two operators of equal precedence are present, they will be parsed from left to right within a line and top to bottom if they include operators from different precedence levels as according to this table.

1	( )
2	`.` (Access)
3	`^`
4	`@`
5	`&`
6	/\
7	`!`, Unary Minus
8	\`
9	`*`, `/`
10	+, − (Subtraction)
11	`%`
12	<, <=, >, >=
13	`==`, `!=`
14	`&&`, `||`
15	=

4.4 Lexical Conventions

4.4.1 Key Words

PRIME has several keywords that are reserved in the language to prevent ambiguity.

Type Related Keywords

The following are the types that exist in our language. We do not allow for users to define their own types. For cryptography uses these should be sufficient.

- int
- lint
- pt
- curve
- string

Statement Related Keywords

The following are the only keywords we have for specific statements.

- if
- else
- for
- while
- return

4.4.2 Variable Names

Variables must be named starting with a letter a through z lower or uppercase followed by any number of alphanumeric characters or underscores.

4.4.3 Comments

Comments in Prime are initiated using the character sequence /* and concluded using */. All text between the /* and the */ will be ignored by the compiler, including across lines. This commenting format applies to both single line and multiline comments.

4.5 Program Structure

The following structural elements dictate the control flow of a program in PRIME.

4.5.1 Conditionals

Conditionals in PRIME come in two forms, both of which begin with an if statement:

`if` (*expression*) *statement*

`if` (*expression*) *statement* `else` *statement*

In each of the above cases, the *expression* following the initial `if` statement is evaluated and if the value returned is nonzero, the following *statement* is executed. The `else` keyword allows for an alternative statement to be executed if the *expression* evaluates to zero. Each `else` will be paired to the last encountered if that is not already paired with an `else`.

```
    int foo;
    int bar;
    foo = 2;
    bar = 0;
    if (foo > 1) {
        print("entered if");     /* prints */
    }else {
        print("entered else");
    }

    if (bar) {
        print("entered if");
    }else {
```

```
        print("entered else");  /* prints */
    }
```

4.5.2 Loops

Loops are used for iteration in PRIME. They may be implemented as while or for loops, described in detail here.

While

While loops in PRIME are of the form:

`while` (*expression*) *statements*

This functions similarly to an `if` statement, except for the fact that the statements following a `while` statement will be repeatedly executed for as long as the expression following the `while` evaluates to a nonzero value. For this reason, it is essential that the expression be updated at some point over the course of the `while` loop, to prevent infinite iteration. The evaluation and check of the expression occurs at the beginning of each iteration.

Examples:

```
    int foo;
    int bar;
    bar = 0;
    foo = 0;
    while (bar < 5) {
        foo = foo + 1;
        bar = bar + 1;
    }
    print(foo); /* prints 5 */
    print(bar); /* prints 5 */
```

For

For loops in PRIME are of the form:

`for (` *expression1* **;** *expression2* **;** *expression3* `)` `statement`

expression1 is executed prior to the first iteration of the loop. *expression2* is evaluated prior to every iteration of the loop, and the loop is entered if *expression2* evaluates to true. *expression1* is executed prior to every iteration of the loop. *statements* are executed during every iteration. The last expression in the for loop statement is optional (in which case this is like a while loop with the initialization done by expression1.

Examples:

```
    int foo;
    int bar;
    for(foo = 0; foo < 5; foo = foo+1){
        print(a);   /* prints 0 1 2 3 4 */
    }
    for(bar = 5; bar < 5; bar = bar+1){
        print(bar); /* no printing, does not enter for loop */
    }
```

Loop Equivalence

As an example of loop equivalence between while and for loops in PRIME, the following implementations are identical in functionality:

expression1 **;**
`while` (*expression2*) {
statement **;**
expression3 **;**
}

for (*expression1* ; *expression2* ; *expression3*) *statement*

4.5.3 Functions

Functions in PRIME are of the form:

return-type function-name (parameters) *statement*

Note that all functions in PRIME must return some value (as there is no void type), and the *statement* (i.e. the function body) must therefore contain some return line. This is so that the aggregate types point and lints can be passed in under the hood as a pointer implicitly into the function (allowing for less wasteful memory use), similar to what clang does for some C functions. The return statement is the keyword return followed my a space and the return expression followed by a semicolon. The parameters list can be empty.

```
return expression;
```

4.5.4 Return

The return statement is used to return some value from a function to its caller. Each return statement must have an expression of the same type of the function that it is defined for. The user cannot declare a void return type as that is not a keyword in our language. This is so that implicit pointer passing can be done when calling a function that has a point or lint return type. Return types include `int`s, `lint`s, `pt`s, and `string`s but not curves. Note that behaviour when attempting to return a curve is left undefined and is not advisable. This is because points are usually to be defined on curves that should be known and set explicitly.

4.6 Scope

4.6.1 Functions

Each function has its own scope. In our language, formal symbols overwrite globals, and locals overwrite formals in the function's symbol table. Variables defined in one function are not accessible to another function.

Examples:

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

int main()
{
  int a;
  a = add(39, 3);
  print(a); /* prints 42 from add(39, 3) */
  return 0;
}
```

4.6.2 Curly Brackets

Curly brackets are used to define the scope within which variable identifiers are recognized. Invoking variables and functions must occur within the same scope where the variables/function was defined. Functions, loops, and conditionals all have their own scope, and attempting to access variables defined within this scope externally will lead to an error.

Examples:

```
int i;
i = 0;
while (i < 5) {
  i = i + 1;
  int a = 0;
}
return a; /* ERROR: 'a' defined within while loop */
```

```
int i;
int a;
while (i < 5) {
  i = i + 1;
  a = 0;
}
return a; /* works since 'a' defined prior to while loop */
```

4.6.3 Semicolons

A semicolon character denotes the end of an expression and the start of the next (since empty space will be ignored). This applies to loops, declarations, conditionals, and all other statement types.

Examples:

```
  int n;
  n = 4 + 3
   /* still going... */
  +2; /* done */
```

4.7 Built-Ins

4.7.1 Printing

Though we would like to have a Python-style print function that infers our types at compile time, we settled for various print functions for the different types. Note that these print statements add a newline. These are the following, with their corresponding types:

- print - prints out integers.
- printl - prints out lints.
- printpt - prints points.
- prints - prints out strings.
- printc - prints out curves.

Errors with these types are caught by our semantic checker by comparing to our built-in declared types.

4.7.2 Type Conversions

Since our central types are lints, as curves and points are both built on these, we have ways to convert to lints from ints and strings.
For integers, since the size of these is a strict subset to the possible size of a lint, this is a straightforward conversion given by the function *tolint()*. Please see the following example:

```
  int a;
  lint b;
  a = 2387468;
  printl(tolint(a));
  b = tolint(a);
```

For strings, this is a little harder since we need to get the integer representation of the particular sequence of characters. Since this is a cryptography language, we do this with encode() and decode() built in functions. These methods, done with a C interface, take the string and convert to a large integer (so that they can take somewhat longer length strings) with padding. The decoding step then takes the encoded number, turns it back into a string, adds whatever padding is required, and turns it character

by character into the input word.
Example usage is as follows:

```
string s;
lint a;
s = "Hello World";
a = encode(s);
printl(a);
prints(decode(a));
```

4.7.3 Random

Because this is a cryptographic language, it would be nice to have a way of obtaining (pseudo)random numbers. For this purpose, we provide a random function that allows provision of a seed and a max number and returns the random number. This is done through C interfacing with GMP external library. Example usage is as follows:

```
lint max; lint seed;
lint rand;
max = 123451;
seed = 101;
rand = random(seed, max);
printl(rand);
```

5 Project Plan

5.1 Process

5.1.1 Planning

We used the milestones set out by the course and started working towards them at least two weeks before each deadline. For the final deliverable, we worked every week from the midterm until the final deadline.

Each week when we met we would update the team on what areas of individual progress each had made, what roadblocks had been encountered and what was on the list to do later that week. Main roadblocks would then be discussed as a team, even debugged and then delegated to someone to complete. At the end of each meeting we would set out goals for the week and what times were best to meet the next time. Any major roadblocks led us to OH. We met with Professor Edwards each week after the middle of the term for updates and ideas on how to solve them.

5.1.2 Testing

For testing, we would write a test for successful use of a feature (developed on separate feature branches) and run the entire regression test suite to ensure that the new addition worked AND did not break anything. Fail test cases were then added and semantic checking and scanning changes were done to ensure that these were caught.

5.1.3 Style Guide

We followed two general style guides: one for writing our source code in OCaml/C, and then another for writing our PRIME language test files.

For C, we tried to follow the Linux Kernel C style as shown in Linux Style. In general, this means that:

- functions have braces starting on newlines
- if, while, and for don't have braces if they just have a one-line statement
- types for a sequence of logical statements are defined above them
- struct definitions at the top of the file or in a header

For OCaml we tried to follow that given by OCaml Style Tutorial.

In general, since this was a large code base, the main idea was to make things as readable as possible so that different members of the team could help out, spending as little time as possible to parse what someone else had written. To this end, When writing OCaml:

- Statements were separated or started on different lines if they would become too long to read on one screen.
- Match cases would be aligned with their corresponding match statement unless that contradicted the whitespace issue.
- Different large logical segments such as function definitions for the C libraries would be separated by at least one blank line and commented descriptively to denote their purpose.
- Complex LLVM statements should have a name before passing the builder argument so that they are readable both in OCaml and then with the generated code.
- When similar logic applied over different match cases, keep the bind namings the same.

For our PRIME language, we recommend following a style similar to that proposed by the C style guide. However, since much of cryptography can involve complex mathematical operations, we recommend commenting more around different parts of algorithms.

For C files, we separated core lint functionality, our aggregate types (point and curve) and input parsing into different files as they had different roles and could be tested iteratively at different points.

5.1.4 Timeline

February 3	Project Proposal
February 14	First Commit
March 24	Hello World
March 30	Lint and Point Types
April 19	Curve and Modified Point Types
April 22	Point Operators
April 25	Demonstrations
April 26	Final Presentation

5.1.5 Software Development Environment

The languages used were: OCaml (including OCamllex for scanner and Ocamlyacc for parser), C for interfacing and Bash scripts for testing. A Makefile was used for compilation, set-up and general orchestration. The interfacing with C was done to take advantage of the GNU Multiple Precision Library which allowed for our large number implementation. All credits for that library go to their creators seen through the link and Contributors section.

For writing code, we used a combination of Vim, VSCode and PyCharm depending on each individual's comfort with the system in question. To test the code, we used an altered MicroC bash test script and ran it on a docker environment.

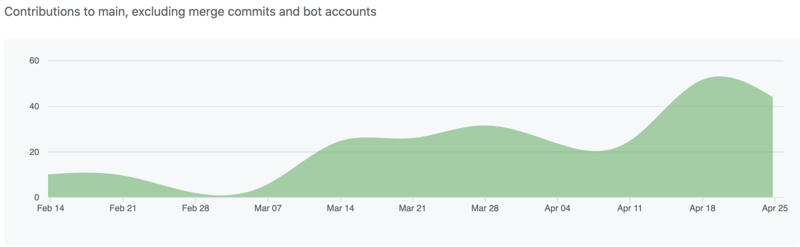
We used Git for version control. We used a GitHub private repository to store our combined code as we built it and integrated it with CircleCI to allow for quicker understanding of build success particularly after merges (see testing section for more information on this).

5.1.6 Roles and Responsibilities

Our roles and responsibilities were functionality driven, so all team members contributed to all portions of the code (including testing) while implementing a given function. For example, to implement a new operator, a team member would add the operator to `parser.mly`, `ast.ml`, `sast.ml`, `scanner.mll`, `semant.ml`, `codegen.ml`, and C libraries where applicable. We also collaborated on more complex features (i.e. multiple team members worked on lints, points, curves, and associated operators).

5.2 Project Log

This project log shows a history of commits from February 14rd to April 26th. As you can see, each member was heavily involved in the development of the project.




```
    // git log --pretty="%C(Yellow)%h  %C(reset)%ad (%C(Green)%cr%C(reset))%x09 %C(Cyan)
    %an: %C(reset)%s" > logs.tex

c437432  Mon Apr 26 13:35:13 2021 -0400 (84 minutes ago)   Thomas Tran: Merge pull
    request #49 from thomasundo2/demos2
c32306b  Mon Apr 26 13:33:11 2021 -0400 (86 minutes ago)   alex-liebeskind: Merge branch
     'main' into demos2
7a38466  Mon Apr 26 13:29:54 2021 -0400 (89 minutes ago)   alex-liebeskind: added
    recursion testing
a5616f1  Mon Apr 26 12:24:42 2021 -0400 (3 hours ago)  nmofficial: small changes and
    tests to demos
cfe9d7d  Mon Apr 26 04:44:34 2021 -0400 (10 hours ago)   nmofficial: updated rsa demos
a4735f0  Mon Apr 26 04:22:47 2021 -0400 (11 hours ago)   nmofficial: Merge pull request
    #48 from thomasundo2/demos2
fda8caa  Mon Apr 26 04:21:46 2021 -0400 (11 hours ago)   nmofficial: adding rsa cleaner
    code version
79aee3f  Mon Apr 26 04:19:10 2021 -0400 (11 hours ago)   nmofficial: Merge pull request
    #47 from thomasundo2/demos2
4a76364  Mon Apr 26 04:16:58 2021 -0400 (11 hours ago)   nmofficial: merging new demos
    Merge branch 'demos2' of https://github.com/thomasundo2/Prime into demos2
657e8c6  Mon Apr 26 04:14:13 2021 -0400 (11 hours ago)   nmofficial: added string test
    case
0ebf524  Mon Apr 26 00:46:24 2021 -0400 (14 hours ago)   alex-liebeskind: Diffie-Hellman
     demo working
457dc69  Sun Apr 25 23:41:21 2021 -0400 (15 hours ago)   pbt-santos: Merge pull request
    #46 from thomasundo2/string_parsing
93ea4d2  Sun Apr 25 23:35:20 2021 -0400 (15 hours ago)   nmofficial: finished RSA Demo
24f8b30  Sun Apr 25 23:14:19 2021 -0400 (16 hours ago)   pbt-santos: change encode
    syntax
5b9b7e0  Sun Apr 25 22:48:26 2021 -0400 (16 hours ago)   nmofficial: added ecc demo,
    fixed string parsing
32b646c  Sun Apr 25 21:35:54 2021 -0400 (17 hours ago)   nmofficial: Merge pull request
    #45 from thomasundo2/access
89265da  Sun Apr 25 21:34:59 2021 -0400 (17 hours ago)   nmofficial: removed test file
7d2a146  Sun Apr 25 21:30:17 2021 -0400 (17 hours ago)   pbt-santos: Fix point ret
    return and ocaml warnings
8e3d92e  Sun Apr 25 20:26:25 2021 -0400 (19 hours ago)   pbt-santos: Add point returns
621ba4d  Sun Apr 25 20:00:05 2021 -0400 (19 hours ago)   nmofficial: chaning to printc
    from printpoly Merge branch 'access' of https://github.com/thomasundo2/Prime into
    access
914fae2  Sun Apr 25 19:59:34 2021 -0400 (19 hours ago)   nmofficial: upated printc
a63dfbe  Sun Apr 25 19:57:42 2021 -0400 (19 hours ago)   nmofficial: semant for crv
    access
4bde4fa  Sun Apr 25 19:32:19 2021 -0400 (19 hours ago)   alex-liebeskind: update curve
    printing
a1467e8  Sun Apr 25 19:22:02 2021 -0400 (20 hours ago)   nmofficial: small change
46593d9  Sun Apr 25 19:11:29 2021 -0400 (20 hours ago)   pbt-santos: Merge branch 'main'
     of https://github.com/thomasundo2/Prime into main
b1fb591  Sun Apr 25 19:11:21 2021 -0400 (20 hours ago)   pbt-santos: Remove int exp
97c9ff6  Sun Apr 25 18:30:39 2021 -0400 (20 hours ago)   nmofficial: fixed test file
75e6ed3  Sun Apr 25 18:14:46 2021 -0400 (21 hours ago)   alex-liebeskind: Merge pull
    request #44 from thomasundo2/test
f568618  Sun Apr 25 18:09:36 2021 -0400 (21 hours ago)   alex-liebeskind: merge for lint
     returns
3cb5224  Sun Apr 25 18:07:53 2021 -0400 (21 hours ago)   nmofficial: added real life
    test cases for big numbers
f4fe1ca  Sun Apr 25 16:45:43 2021 -0400 (22 hours ago)   pbt-santos: Merge branch 'main'
     of https://github.com/thomasundo2/Prime into main
```

```
5fa5387  Sun Apr 25 16:45:36 2021 -0400 (22 hours ago)   pbt-santos: Add lint user
    defined function passing
c015e8b  Sun Apr 25 16:01:23 2021 -0400 (23 hours ago)   alex-liebeskind: Merge branch '
    test' of https://github.com/thomasundo2/Prime into test
a3c3923  Sun Apr 25 16:01:13 2021 -0400 (23 hours ago)   alex-liebeskind: update demos
0b560d6  Sun Apr 25 15:08:02 2021 -0400 (24 hours ago)   nmofficial: fixed warnings w
    point binop
642ca8c  Sun Apr 25 14:53:30 2021 -0400 (24 hours ago)   alex-liebeskind: codegen errors
    /tests
998f100  Sun Apr 25 13:25:10 2021 -0400 (26 hours ago)   alex-liebeskind: added dh
0960bb6  Sun Apr 25 13:10:03 2021 -0400 (26 hours ago)   nmofficial: Merge pull request
    #43 from thomasundo2/poly
4f5a872  Sun Apr 25 13:08:33 2021 -0400 (26 hours ago)   nmofficial: resolved merge
    conflicts w main
acfed4d  Sun Apr 25 12:41:39 2021 -0400 (26 hours ago)   nmofficial: taking out this
    test for now until we get return special types working
e13d04a  Sun Apr 25 12:32:55 2021 -0400 (26 hours ago)   nmofficial: changed poly to
    curve (in scanner) only. Updated test cases
551bfa5  Sun Apr 25 11:40:24 2021 -0400 (27 hours ago)   pbt-santos: Update test_all.sh
aa87e63  Sun Apr 25 11:36:10 2021 -0400 (27 hours ago)   pbt-santos: Merge pull request
    #41 from thomasundo2/input_encoding
46f71e7  Sun Apr 25 11:31:52 2021 -0400 (27 hours ago)   pbt-santos: Fix decode, add
    test cases, update Make and test
6a81a0f  Sun Apr 25 10:58:53 2021 -0400 (28 hours ago)   Thomas Tran: Merge pull request
     #40 from thomasundo2/debug
d07f825  Sun Apr 25 10:54:47 2021 -0400 (28 hours ago)   alex-liebeskind: mpow fail case
4ea6452  Sun Apr 25 10:49:01 2021 -0400 (28 hours ago)   alex-liebeskind: trnops
5fae68a  Sun Apr 25 10:40:29 2021 -0400 (28 hours ago)   alex-liebeskind: codegen done (
    revisit lclear_func and pt_mul_func)
00e0a32  Sun Apr 25 10:33:48 2021 -0400 (28 hours ago)   alex-liebeskind: fix mor
    codegen errors
ac4fd51  Sun Apr 25 09:59:23 2021 -0400 (29 hours ago)   alex-liebeskind: remove
    extraneous matching
24dd9f7  Sun Apr 25 09:47:12 2021 -0400 (29 hours ago)   alex-liebeskind: fix codegen
    errors
18892cb  Sun Apr 25 05:08:52 2021 -0400 (34 hours ago)   nmofficial: fixed all shift/
    reduce conflicts
b07650f  Sun Apr 25 04:08:04 2021 -0400 (35 hours ago)   nmofficial: != working for pts
    and good test case
ea60167  Sun Apr 25 03:52:36 2021 -0400 (35 hours ago)   nmofficial: == working for pts,
     added goo test case
f885e34  Sun Apr 25 03:28:34 2021 -0400 (2 days ago)   nmofficial: neg works with good
    test case
c1e8132  Sun Apr 25 02:25:24 2021 -0400 (2 days ago)   nmofficial: added fail case for
    mult
f8d1e6b  Sun Apr 25 02:12:15 2021 -0400 (2 days ago)   nmofficial: multiplication works,
     added good test cases, cleaned addition code. still mallocing
39ff25d  Sun Apr 25 00:44:30 2021 -0400 (2 days ago)   pbt-santos: add decode from lint.
     may change args
74a35ab  Sun Apr 25 00:17:01 2021 -0400 (2 days ago)   Thomas Tran: remove diff files
    and add to gitignore
27bbffd  Sun Apr 25 00:14:27 2021 -0400 (2 days ago)   Thomas Tran: add land -- Merge
    pull request #39 from thomasundo2/l_a_ops
d6e366c  Sun Apr 25 00:07:44 2021 -0400 (2 days ago)   alex-liebeskind: add back
    test_point_acc_fail
f148f97  Sat Apr 24 23:56:46 2021 -0400 (2 days ago)   alex-liebeskind: remove extra
    files
0e15e33  Sat Apr 24 23:41:24 2021 -0400 (2 days ago)   alex-liebeskind: remove extra
    files
a9c5cd0  Sat Apr 24 23:40:56 2021 -0400 (2 days ago)   alex-liebeskind: lint ops
    complete
40bb5c9  Sat Apr 24 23:33:30 2021 -0400 (2 days ago)   alex-liebeskind: update lint op
    testing/conditionals
359c5d5  Sat Apr 24 23:30:06 2021 -0400 (2 days ago)   nmofficial: changed test_lint_ret
    .pr
7fdda40  Sat Apr 24 23:19:13 2021 -0400 (2 days ago)   nmofficial: merge and mult
    working
b9a9f7f  Sat Apr 24 23:10:12 2021 -0400 (2 days ago)   alex-liebeskind: remove extra
    files
db49ded  Sat Apr 24 23:09:05 2021 -0400 (2 days ago)   alex-liebeskind: codegen update
    lint ops
baafa1f  Sat Apr 24 22:36:51 2021 -0400 (2 days ago)   alex-liebeskind: merge
5d3c6db  Sat Apr 24 22:36:15 2021 -0400 (2 days ago)   pbt-santos: Merge branch 'main'
```

```
    into input_encoding
9ce06f3  Sat Apr 24 22:31:04 2021 -0400 (2 days ago)   pbt-santos: add encoding of
    string to 3 digit numbers
917aab3  Sat Apr 24 22:22:00 2021 -0400 (2 days ago)   nmofficial: fixed mult semantics
    issue
ac7aa34  Sat Apr 24 19:59:08 2021 -0400 (2 days ago)   pbt-santos: Change testfile to
    report all output
e5cb3bf  Sat Apr 24 19:28:44 2021 -0400 (2 days ago)   pbt-santos: Fix merge issues
8b9e49f  Sat Apr 24 19:12:54 2021 -0400 (2 days ago)   nmofficial: Merge pull request
    #38 from thomasundo2/poly
6cc8bec  Sat Apr 24 19:11:50 2021 -0400 (2 days ago)   nmofficial: Update parser.mly
cdaa2d4  Sat Apr 24 19:08:48 2021 -0400 (2 days ago)   nmofficial: Merge branch 'main'
    into poly
06ad7d1  Sat Apr 24 16:32:33 2021 -0400 (2 days ago)   nmofficial: starting
    multiplication for points
2315e59  Sat Apr 24 15:55:44 2021 -0400 (2 days ago)   alex-liebeskind: hash table for
    conversion
d4dd6a2  Sat Apr 24 03:45:27 2021 -0400 (2 days ago)   nmofficial: fixed .out file for
    test_lint2
6460397  Sat Apr 24 03:44:04 2021 -0400 (2 days ago)   nmofficial: lint_2 not a fail
    case
e20c7b1  Sat Apr 24 03:39:25 2021 -0400 (2 days ago)   nmofficial: pulling Merge branch
    'main' of https://github.com/thomasundo2/Prime into main
4878c50  Sat Apr 24 03:39:07 2021 -0400 (2 days ago)   nmofficial: removed pointadd test
    . point addition test cases will come from poly branch
3cfd9bf  Sat Apr 24 03:32:43 2021 -0400 (2 days ago)   nmofficial: deleted print
    statments in add function, added good test cases, everything passes
993990a  Sat Apr 24 02:26:53 2021 -0400 (3 days ago)   nmofficial: pt add works
255d86b  Fri Apr 23 20:51:47 2021 -0400 (3 days ago)   alex-liebeskind: Merge branch '
    tests' into main
ab026d1  Fri Apr 23 20:51:17 2021 -0400 (3 days ago)   alex-liebeskind: added test case
    / update naming convention
04bdabc  Fri Apr 23 18:51:48 2021 -0400 (3 days ago)   alex-liebeskind: Merge branch '
    main' of https://github.com/thomasundo2/Prime into main
4187535  Fri Apr 23 18:51:35 2021 -0400 (3 days ago)   alex-liebeskind: added test_func.
    pr
d89c7a5  Fri Apr 23 18:41:14 2021 -0400 (3 days ago)   nmofficial: Merge pull request
    #37 from thomasundo2/lint_unops
31e5f13  Fri Apr 23 18:39:59 2021 -0400 (3 days ago)   nmofficial: neg and not working
    for lints
057dafe  Fri Apr 23 18:23:07 2021 -0400 (3 days ago)   nmofficial: not working for lints
ad101ca  Fri Apr 23 17:51:06 2021 -0400 (3 days ago)   nmofficial: addition throws seg
    fault
4b8a2da  Fri Apr 23 12:30:05 2021 -0400 (3 days ago)   pbt-santos: fix merging changes
b0dcc0f  Fri Apr 23 12:16:59 2021 -0400 (3 days ago)   pbt-santos: Merge pull request
    #36 from thomasundo2/casting
b710371  Fri Apr 23 12:16:47 2021 -0400 (3 days ago)   pbt-santos: Merge branch 'main'
    into casting
c9748b1  Fri Apr 23 12:12:30 2021 -0400 (3 days ago)   pbt-santos: Merge pull request
    #35 from thomasundo2/point
9b9cc72  Fri Apr 23 12:11:53 2021 -0400 (3 days ago)   pbt-santos: Merge branch 'main'
    into point
a0ea612  Thu Apr 22 15:19:08 2021 -0400 (4 days ago)   nmofficial: Merge pull request
    #33 from thomasundo2/lint_sops
335ea5b  Thu Apr 22 15:16:41 2021 -0400 (4 days ago)   nmofficial: fixed test cases,
    made rand same seed so test works
5ec95ac  Thu Apr 22 04:18:47 2021 -0400 (4 days ago)   nmofficial: ternary ops
    implemented successfully, power-mod works
fa36922  Thu Apr 22 03:33:47 2021 -0400 (4 days ago)   nmofficial: debugged parser,
    semant
5826624  Thu Apr 22 03:28:18 2021 -0400 (5 days ago)   nmofficial: ternops done except
    codegen
d64e8c5  Thu Apr 22 02:39:47 2021 -0400 (5 days ago)   nmofficial: changed ^ to /\ and
    started ternops
83903b1  Thu Apr 22 02:10:00 2021 -0400 (5 days ago)   nmofficial: add option to seed
    with time to random
e24b4cd  Thu Apr 22 00:48:09 2021 -0400 (5 days ago)   nmofficial: random works
4a98a33  Wed Apr 21 22:12:13 2021 -0400 (5 days ago)   nmofficial: neatened up code for
    randomstill not fully functional
131eebb  Wed Apr 21 22:06:12 2021 -0400 (5 days ago)   nmofficial: basics for random.
    not fully implemented
fbb8c43  Wed Apr 21 18:52:23 2021 -0400 (5 days ago)   nmofficial: fixed small bugs with
     inverse. all cases work although test_all shows intended issue with test_inv_fail
```

```
9a027ce  Wed Apr 21 16:02:06 2021 -0400 (5 days ago)   alex-liebeskind: linv fixed ast/codegen/scanner, needs error handling for when linv does not exist
051b16e  Wed Apr 21 15:29:51 2021 -0400 (5 days ago)   nmofficial: fixed small bugs inverse causing parse error
e485e69  Wed Apr 21 14:52:08 2021 -0400 (5 days ago)   nmofficial: minor test_file.sh changes
da6ac4d  Wed Apr 21 14:29:48 2021 -0400 (5 days ago)   nmofficial: base implementation for invert not yet functional
4bf5823  Tue Apr 20 18:37:27 2021 -0400 (6 days ago)   Thomas Tran: points defined under a poly
df6dd60  Tue Apr 20 17:48:45 2021 -0400 (6 days ago)   pbt-santos: add one more test for int cast
d935585  Tue Apr 20 17:46:21 2021 -0400 (6 days ago)   pbt-santos: add casting from ints to lints
36611d9  Tue Apr 20 11:44:40 2021 -0400 (6 days ago)   Thomas Tran: polys working
edc16bf  Mon Apr 19 17:51:30 2021 -0400 (7 days ago)   Thomas Tran: modify polys to match point struct
b8e35c0  Mon Apr 19 17:14:00 2021 -0400 (7 days ago)   Thomas Tran: merge point onto poly
d10f1db  Mon Apr 19 17:08:41 2021 -0400 (7 days ago)   Thomas Tran: polys working, but printpoly doesnt work with variables
4bf7320  Mon Apr 19 15:46:50 2021 -0400 (7 days ago)   alex-liebeskind: Merge branch 'forwhile' into main
8b4ddd4  Mon Apr 19 15:45:24 2021 -0400 (7 days ago)   alex-liebeskind: add lint operators to int operators
fc32f75  Mon Apr 19 19:21:23 2021 +0000 (7 days ago)   alex-liebeskind: added relop and implemented relational operators for lints
9fc4970  Mon Apr 19 15:11:02 2021 -0400 (7 days ago)   Thomas Tran: poly semant checking
f143f27  Mon Apr 19 15:04:42 2021 -0400 (7 days ago)   Thomas Tran: polys up until ast, deprecate ints with points
9c5ab63  Mon Apr 19 14:15:35 2021 -0400 (7 days ago)   pbt-santos: add point access and test cases
b197b9a  Mon Apr 19 14:12:04 2021 -0400 (7 days ago)   alex-liebeskind: fixed semant for lint comparators
01a04ec  Mon Apr 19 14:02:55 2021 -0400 (7 days ago)   alex-liebeskind: lint comparators
81111b4  Mon Apr 19 12:05:24 2021 -0400 (7 days ago)   Thomas Tran: Merge remote-tracking branch 'origin/point' into poly
bc36dde  Mon Apr 19 11:36:22 2021 -0400 (7 days ago)   pbt-santos: clean up code points
0b9dffd  Mon Apr 19 02:45:09 2021 -0400 (8 days ago)   nmofficial: Update README.md
44a54b9  Mon Apr 19 02:38:50 2021 -0400 (8 days ago)   Thomas Tran: Merge pull request #32 from thomasundo2/forwhile
e583e97  Mon Apr 19 02:31:39 2021 -0400 (8 days ago)   nmofficial: not works for ints, test cases added (all binops, unops completed for ints)
b3a603f  Mon Apr 19 02:08:47 2021 -0400 (8 days ago)   nmofficial: and/or functionality for ints and tests cases
7edeeb0  Mon Apr 19 01:48:18 2021 -0400 (8 days ago)   nmofficial: changed comp ops to L.build_zext to finx -1 issue
961495d  Sun Apr 18 22:57:26 2021 -0400 (8 days ago)   nmofficial: fixed bash script modified tests cases for for/while
c938b30  Sun Apr 18 22:42:48 2021 -0400 (8 days ago)   nmofficial: fixed while/for, wrote test_file.sh script to test individual files
b32e78f  Sun Apr 18 21:47:08 2021 -0400 (8 days ago)   nmofficial: fixed conditional operators for ints (not including and/or))
22f35c9  Sun Apr 18 15:49:47 2021 -0400 (8 days ago)   nmofficial: if works with single variable predicates
ab61871  Sun Apr 18 14:46:07 2021 -0400 (8 days ago)   pbt-santos: fix point assignment
47fd128  Sun Apr 18 12:18:15 2021 -0400 (8 days ago)   nmofficial: int comp ops return ints, var in if statement not working
3184cb7  Sat Apr 17 19:58:09 2021 -0400 (9 days ago)   nmofficial: added not back (returns int)
cb1feb7  Sat Apr 17 19:02:34 2021 -0400 (9 days ago)   alex-liebeskind: push ifelse testing
4add0d8  Sat Apr 17 17:57:02 2021 -0400 (9 days ago)   alex-liebeskind: for while loops working
9409e19  Sat Apr 17 17:26:25 2021 -0400 (9 days ago)   alex-liebeskind: Merge branch 'forwhile' of https://github.com/thomasundo2/Prime into forwhile
e381deb  Sat Apr 17 17:24:52 2021 -0400 (9 days ago)   Thomas Tran: fixed precedence
48c23d7  Sat Apr 17 17:22:27 2021 -0400 (9 days ago)   Thomas Tran: for while working
a3a9c68  Sat Apr 17 16:59:01 2021 -0400 (9 days ago)   alex-liebeskind: Merge branch 'ifelse' into forwhile
a65549c  Sat Apr 17 16:55:50 2021 -0400 (9 days ago)   alex-liebeskind: semant checking for point types, merge Lint operators
331209f  Wed Apr 14 20:42:49 2021 -0400 (12 days ago)  alex-liebeskind: updating before
```

```
    working on points
c83c5f3  Wed Apr 14 13:41:59 2021 -0400 (12 days ago)  Thomas Tran: start of points with
     lint
ec373ef  Mon Apr 12 14:08:53 2021 -0400 (2 weeks ago)  alex-liebeskind: testing ifelse
    and loops
f64af8c  Mon Apr 12 13:54:40 2021 -0400 (2 weeks ago)  Thomas Tran: fix syntax error
78d06cb  Mon Apr 12 12:57:16 2021 -0400 (2 weeks ago)  alex-liebeskind: adding ifelse
    forwhile to codegen
9fa4535  Mon Apr 12 12:32:51 2021 -0400 (2 weeks ago)  alex-liebeskind: Merge pull
    request #29 from thomasundo2/forwhile
9347785  Mon Apr 12 12:32:11 2021 -0400 (2 weeks ago)  alex-liebeskind: comparative
    operators tested
001eee8  Mon Apr 12 12:08:09 2021 -0400 (2 weeks ago)  alex-liebeskind: Merge pull
    request #28 from thomasundo2/forwhile
694baad  Mon Apr 12 12:06:41 2021 -0400 (2 weeks ago)  alex-liebeskind: and or bneq
    fully added up to codegen
bdb681e  Mon Apr 12 11:53:53 2021 -0400 (2 weeks ago)  alex-liebeskind: adding and or
    nbeq
f3cb7f4  Mon Apr 12 11:47:20 2021 -0400 (2 weeks ago)  alex-liebeskind: comparative
    operators implemented
fd06145  Mon Apr 12 11:36:16 2021 -0400 (2 weeks ago)  alex-liebeskind: Added boolean
    operators to semant.ml, ast.ml, sast.ml
9d8569f  Sat Apr 10 15:43:00 2021 -0400 (2 weeks ago)  Thomas Tran: Merge pull request
    #27 from thomasundo2/point
66e85d6  Sat Apr 10 14:58:27 2021 -0400 (2 weeks ago)  Thomas Tran: fixed point testing
911c4d3  Sat Apr 10 14:42:22 2021 -0400 (2 weeks ago)  Thomas Tran: Merge pull request
    #26 from thomasundo2/maintopoint
fe194dd  Sat Apr 10 14:42:09 2021 -0400 (2 weeks ago)  Thomas Tran: Merge branch 'point'
     into maintopoint
8e9a740  Sat Apr 10 14:27:15 2021 -0400 (2 weeks ago)  pbt-santos: Merge pull request
    #25 from thomasundo2/lints
9b52a8d  Sat Apr 10 14:25:39 2021 -0400 (2 weeks ago)  pbt-santos: Merge branch 'main'
    into lints
4cf96f1  Sat Apr 10 13:52:35 2021 -0400 (2 weeks ago)  pbt-santos: Finish lints and
    improve test_file
4170504  Fri Apr 9 14:40:09 2021 -0400 (2 weeks ago)   pbt-santos: Add subtract and pow
    for litns
252340c  Fri Apr 9 14:07:13 2021 -0400 (2 weeks ago)   pbt-santos: clean up lints code
ceef5b7  Fri Apr 9 13:19:43 2021 -0400 (2 weeks ago)   pbt-santos: resolve merge
    conflicts
67098a8  Fri Apr 9 13:18:27 2021 -0400 (2 weeks ago)   pbt-santos: start lint
    improvement
5ea8c8e  Thu Apr 8 23:43:07 2021 -0400 (3 weeks ago)   pbt-santos: fix: rename testing
    file in yaml
f5c4e7c  Thu Apr 8 23:40:19 2021 -0400 (3 weeks ago)   nmofficial: add works barely
f2bb123  Thu Apr 8 21:30:56 2021 -0400 (3 weeks ago)   nmofficial: add function for
    lints (not working)
0bddb1e  Wed Apr 7 23:57:10 2021 -0400 (3 weeks ago)   Thomas Tran: add back lint
    scanner
35f70f7  Wed Apr 7 23:24:29 2021 -0400 (3 weeks ago)   Thomas Tran: Merge branch 'point'
     of https://github.com/thomasundo2/Prime into point
04e7f65  Wed Apr 7 23:23:25 2021 -0400 (3 weeks ago)   Thomas Tran: Merge branch 'lints'
     into point
63771ef  Wed Apr 7 22:52:27 2021 -0400 (3 weeks ago)   alex-liebeskind: removed access
    from semant.ml
071dd9d  Tue Apr 6 14:49:12 2021 -0400 (3 weeks ago)   pbt-santos: Add functionality for
     lintlits in function calls
4e31408  Tue Apr 6 12:52:26 2021 -0400 (3 weeks ago)   pbt-santos: Fix code alignment
    and new lint print
332e6be  Mon Apr 5 12:31:41 2021 -0400 (3 weeks ago)   alex-liebeskind: added for/while
    tests
830c391  Mon Apr 5 12:26:27 2021 -0400 (3 weeks ago)   alex-liebeskind: added ifelse
    test cases
cac324c  Sun Apr 4 23:39:17 2021 -0400 (3 weeks ago)   pbt-santos: Add parsing for new
    lints and adding extern funcs
38c0ee8  Sun Apr 4 22:59:11 2021 -0400 (3 weeks ago)   alex-liebeskind: Merge pull
    request #23 from thomasundo2/point
780b762  Sun Apr 4 22:58:41 2021 -0400 (3 weeks ago)   alex-liebeskind: Merge pull
    request #22 from thomasundo2/point
d06ecfb  Sun Apr 4 22:57:47 2021 -0400 (3 weeks ago)   alex-liebeskind: ifelse forwhile
    code added - move to respective branches
4dca528  Sun Apr 4 22:57:04 2021 -0400 (3 weeks ago)   alex-liebeskind: Merge pull
    request #21 from thomasundo2/point
```

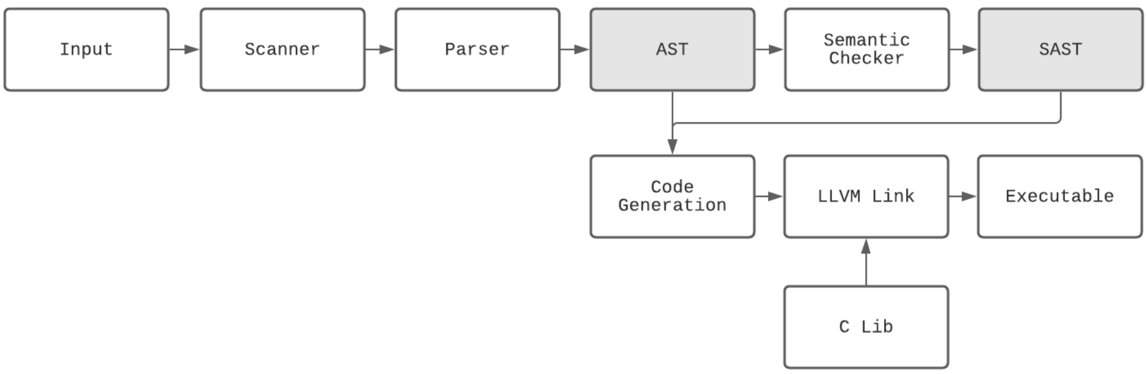
```
8203908  Sun Apr 4 21:57:44 2021 -0400 (3 weeks ago)   alex-liebeskind: added elliptic curve addition for b = 1
f60e5a4  Sun Apr 4 21:36:02 2021 -0400 (3 weeks ago)   alex-liebeskind: create and link structs.c
5438778  Sun Apr 4 21:22:01 2021 -0400 (3 weeks ago)   Thomas Tran: Merge pull request #19 from thomasundo2/point
a1aca41  Sun Apr 4 21:21:52 2021 -0400 (3 weeks ago)   Thomas Tran: Merge pull request #20 from thomasundo2/point
f7381da  Sun Apr 4 21:18:55 2021 -0400 (3 weeks ago)   thomasundo2: arithmetic-wise point addition and test cases added
b1bab2b  Sun Apr 4 20:51:56 2021 -0400 (3 weeks ago)   thomasundo2: access fully functional for points
6f415b1  Sun Apr 4 20:31:52 2021 -0400 (3 weeks ago)   Thomas Tran: Merge pull request #18 from thomasundo2/main
b7322b3  Sun Apr 4 20:26:41 2021 -0400 (3 weeks ago)   thomasundo2: accessing strings partially works
7b2c682  Sun Apr 4 20:04:48 2021 -0400 (3 weeks ago)   thomasundo2: add point structs and remove point access
b862401  Thu Apr 1 22:45:06 2021 -0400 (4 weeks ago)   pbt-santos: Merge branch 'lints' of https://github.com/thomasundo2/Prime into lints
5134179  Thu Apr 1 22:45:01 2021 -0400 (4 weeks ago)   pbt-santos: rename test script
fc7118e  Thu Apr 1 22:31:57 2021 -0400 (4 weeks ago)   nmofficial: sub and exp functions Merge branch 'lints' of https://github.com/thomasundo2/Prime into lints
68cfcff  Thu Apr 1 22:31:49 2021 -0400 (4 weeks ago)   nmofficial: added test_lint.c
c381b0e  Thu Apr 1 13:02:19 2021 -0400 (4 weeks ago)   pbt-santos: Merge branch 'lints' of https://github.com/thomasundo2/Prime into lints
5b1f5e1  Thu Apr 1 13:00:20 2021 -0400 (4 weeks ago)   pbt-santos: Add lint exponentiation
6adfd6d  Thu Apr 1 11:00:22 2021 -0400 (4 weeks ago)   pbt-santos: Delete gmpfunc.o
f8cd8f2  Thu Apr 1 10:59:41 2021 -0400 (4 weeks ago)   pbt-santos: Update .gitignore
03d9119  Wed Mar 31 23:38:57 2021 -0400 (4 weeks ago)  Thomas Tran: Merge pull request #17 from thomasundo2/point
bf1c50c  Wed Mar 31 23:35:12 2021 -0400 (4 weeks ago)  alex-liebeskind: Point (#16)
f359a0b  Wed Mar 31 23:06:29 2021 -0400 (4 weeks ago)  thomasundo2: ints for points working
ba04fcd  Wed Mar 31 22:37:30 2021 -0400 (4 weeks ago)  thomasundo2: add accessing and points nonfunctional
62a2ddc  Wed Mar 31 21:06:50 2021 -0400 (4 weeks ago)  Thomas Tran: Lints branch into point  (#14)
2f9cc20  Wed Mar 31 21:06:21 2021 -0400 (4 weeks ago)  Thomas Tran: Merge branch 'point' into lints
a3c126b  Tue Mar 30 21:47:40 2021 -0400 (4 weeks ago)  pbt-santos: fix: link gmp library for tests
05f5026  Tue Mar 30 21:43:01 2021 -0400 (4 weeks ago)  pbt-santos: Add lint addition (memory leaks)
13a7c11  Tue Mar 30 13:16:36 2021 -0400 (4 weeks ago)  pbt-santos: Add lint initialization
7bbe9d0  Mon Mar 29 22:02:50 2021 -0400 (4 weeks ago)  alex-liebeskind: changed recursive expression evaluation per Professor Edward's advice
d4133ea  Mon Mar 29 00:18:21 2021 -0400 (4 weeks ago)  pbt-santos: Fix merge and add test output
87b778f  Mon Mar 29 00:14:45 2021 -0400 (4 weeks ago)  pbt-santos: Add gmp c file for lint reference
f9a8828  Sun Mar 28 23:33:06 2021 -0400 (4 weeks ago)  nmofficial: assign works, updated sast sexr to account for binop/unop, deleted Semi from AST operators
d75fc48  Sun Mar 28 21:47:58 2021 -0400 (4 weeks ago)  alex-liebeskind: Point type added to codegen.ml
e596012  Sun Mar 28 21:03:10 2021 -0400 (4 weeks ago)  alex-liebeskind: 2D points
33ef62c  Sun Mar 28 20:53:52 2021 -0400 (4 weeks ago)  thomasundo2: change test_point file to print points
9407bf6  Sun Mar 28 20:51:34 2021 -0400 (4 weeks ago)  thomasundo2: printing for points
0f36d62  Sun Mar 28 20:47:04 2021 -0400 (4 weeks ago)  thomasundo2: change Ptlit back to one line
1c9722b  Sun Mar 28 20:15:26 2021 -0400 (4 weeks ago)  thomasundo2: changes to semant.ml
be3f1cf  Sun Mar 28 18:53:45 2021 -0400 (4 weeks ago)  nmofficial: continued to debug for assign, prime.native compiles, ERR: Fatal error: exception Failure("undeclared identifier test") when converting to LLVM
9fb31c4  Sun Mar 28 17:31:02 2021 -0400 (4 weeks ago)  nmofficial: Merge branch 'lints' of https://github.com/thomasundo2/Prime into lints merge to push pretty print for lints
2a84397  Sun Mar 28 17:30:42 2021 -0400 (4 weeks ago)  nmofficial: petty print for lints
d1e3977  Sun Mar 28 15:43:13 2021 -0400 (4 weeks ago)  pbt-santos: fix var assignment writing
```

```
bf7a05a  Sun Mar 28 15:29:33 2021 -0400 (4 weeks ago)  pbt-santos: Fix test_mod
7316ac6  Sun Mar 28 15:20:45 2021 -0400 (4 weeks ago)  pbt-santos: Fix test_add and update tests to run
afd0a80  Sun Mar 28 15:08:06 2021 -0400 (4 weeks ago)  pbt-santos: Add gmp install to make. Test all written
14f43a5  Sun Mar 28 14:03:19 2021 -0400 (4 weeks ago)  thomasundo2: Merge branch 'point' of https://github.com/thomasundo2/Prime into point
7c1d6e7  Sun Mar 28 14:02:36 2021 -0400 (4 weeks ago)  thomasundo2: add Ptlit to semant.ml, doesn't work yet...
e5eae07  Sun Mar 28 13:54:12 2021 -0400 (4 weeks ago)  alex-liebeskind: update semant.ml
9b8d7b6  Sun Mar 28 13:30:13 2021 -0400 (4 weeks ago)  alex-liebeskind: update semant.ml
4529fac  Sun Mar 28 13:24:11 2021 -0400 (4 weeks ago)  thomasundo2: fix defns for lbrack and rbrack
d2d025d  Sun Mar 28 13:06:10 2021 -0400 (4 weeks ago)  alex-liebeskind: point updates
70ac7c6  Sun Mar 28 13:02:36 2021 -0400 (4 weeks ago)  alex-liebeskind: point updates
b86de34  Sun Mar 28 12:41:25 2021 -0400 (4 weeks ago)  alex-liebeskind: update test function
6760130  Sun Mar 28 12:20:12 2021 -0400 (4 weeks ago)  alex-liebeskind: update codegen parser sast for points
0ad4ac9  Sun Mar 28 12:19:47 2021 -0400 (4 weeks ago)  alex-liebeskind: update ast.ml
40a8790  Sat Mar 27 13:23:24 2021 -0400 (4 weeks ago)  alex-liebeskind: create test print function
46351f9  Sat Mar 27 13:01:30 2021 -0400 (4 weeks ago)  nmofficial: not works for integers
1333442  Fri Mar 26 13:16:36 2021 -0400 (4 weeks ago)  pbt-santos: Properly clean up in makefile
2b05786  Fri Mar 26 13:13:04 2021 -0400 (4 weeks ago)  pbt-santos: Merge pull request #13 from thomasundo2/testing-ops
5497db6  Fri Mar 26 13:12:57 2021 -0400 (4 weeks ago)  pbt-santos: Merge branch 'main' into testing-ops
13de12c  Fri Mar 26 12:18:40 2021 -0400 (4 weeks ago)  pbt-santos: Add tests for integer operations; simplify conf
836b5d4  Fri Mar 26 12:08:26 2021 -0400 (4 weeks ago)  Thomas Tran: Merge pull request #12 from thomasundo2/strings
7886042  Fri Mar 26 12:08:18 2021 -0400 (4 weeks ago)  Thomas Tran: Merge branch 'main' into strings
2c4eb64  Fri Mar 26 16:04:56 2021 +0000 (4 weeks ago)  root: add strings, prints, and pretty print sast
a70323b  Fri Mar 26 11:56:07 2021 -0400 (4 weeks ago)  pbt-santos: Merge pull request #11 from thomasundo2/nikhil
243b07b  Fri Mar 26 11:55:57 2021 -0400 (4 weeks ago)  pbt-santos: Merge branch 'main' into nikhil
3618156  Fri Mar 26 02:55:49 2021 -0400 (5 weeks ago)  nmofficial: test files added
55e4457  Fri Mar 26 01:00:33 2021 -0400 (5 weeks ago)  nmofficial: added unary operators but ! (not) will give the opposite number (kind of)
7e3a15a  Fri Mar 26 00:00:31 2021 -0400 (5 weeks ago)  nmofficial: added standard binops for ints/ note that power is not functional, instead serves multiplication
86081e7  Thu Mar 25 12:35:33 2021 -0400 (5 weeks ago)  pbt-santos: Merge pull request #10 from thomasundo2/testing-hello
4a0584b  Thu Mar 25 12:20:51 2021 -0400 (5 weeks ago)  pbt-santos: Merge scanner main
83fffa2  Thu Mar 25 12:18:37 2021 -0400 (5 weeks ago)  pbt-santos: Remove float and extraneous code from scanner
5319ff4  Thu Mar 25 16:17:37 2021 +0000 (5 weeks ago)  root: modify string scanner
ce26fc4  Wed Mar 24 19:04:25 2021 +0000 (5 weeks ago)  root: fixed warnings in ast
10d779f  Wed Mar 24 14:42:07 2021 -0400 (5 weeks ago)  pbt-santos: Fix ocaml warnings
581723e  Wed Mar 24 14:20:17 2021 -0400 (5 weeks ago)  Thomas Tran: Merge pull request #9 from thomasundo2/nikhil
9d7f171  Wed Mar 24 14:15:11 2021 -0400 (5 weeks ago)  alex-liebeskind: Update README.md
c0750db  Wed Mar 24 14:14:51 2021 -0400 (5 weeks ago)  alex-liebeskind: Update README.md
45a5f50  Wed Mar 24 14:13:19 2021 -0400 (5 weeks ago)  root: fix string and chr format in scanner test
7a64019  Wed Mar 24 14:13:18 2021 -0400 (5 weeks ago)  nmofficial: fixed pattern matching warnings in codegen.ml (lines 40, 117) now serve void type or 0 int
311b561  Tue Mar 23 22:11:36 2021 -0400 (5 weeks ago)  pbt-santos: Add prints for string printing in semant checker
2d5f45d  Tue Mar 23 22:03:19 2021 -0400 (5 weeks ago)  pbt-santos: Add copy of scanner for print unit testing
fcb20c3  Tue Mar 23 23:25:25 2021 +0000 (5 weeks ago)  root: add pretty print ast (test/test_hello.pr)
13054c8  Tue Mar 23 14:49:38 2021 -0400 (5 weeks ago)  pbt-santos: Merge pull request #8 from thomasundo2/testing-hello
9c41f99  Tue Mar 23 11:08:22 2021 -0400 (5 weeks ago)  pbt-santos: Update README.md
148e333  Mon Mar 22 23:02:08 2021 -0400 (5 weeks ago)  pbt-santos: Make test print 0
```

```
d09cef3  Sun Mar 21 22:20:30 2021 -0400 (5 weeks ago)  pbt-santos: Merge pull request #7 from thomasundo2/nikhil
0a24d3f  Sun Mar 21 13:43:45 2021 -0400 (5 weeks ago)  nmofficial: fixed all bugs in parser, code compiles
6b82b52  Sun Mar 21 11:52:30 2021 -0400 (5 weeks ago)  nmofficial: fixed errors in codegen, parser, semant
1628443  Sat Mar 20 17:05:17 2021 -0400 (5 weeks ago)  nmofficial: changed to codegen
251cc77  Sat Mar 20 18:13:55 2021 +0000 (5 weeks ago)  pbt-santos: fix consistent naming for semant
e1974a6  Sat Mar 20 18:10:06 2021 +0000 (5 weeks ago)  pbt-santos: add ocamlbuild clean
fe4c5f2  Sat Mar 20 14:04:54 2021 -0400 (5 weeks ago)  nmofficial: added _tags file to include llvm module
42f70d4  Sat Mar 20 14:00:26 2021 -0400 (5 weeks ago)  nmofficial: Merge branch 'main' of https://github.com/thomasundo2/Prime into main
28a1526  Sat Mar 20 18:00:10 2021 +0000 (5 weeks ago)  pbt-santos: fix
43ec223  Sat Mar 20 13:58:04 2021 -0400 (5 weeks ago)  nmofficial: Merge branch 'main' of https://github.com/thomasundo2/Prime into main
dfaca5e  Sat Mar 20 17:57:40 2021 +0000 (5 weeks ago)  pbt-santos: Merge branch 'main' of https://github.com/thomasundo2/Prime into main
74a131f  Sat Mar 20 17:57:28 2021 +0000 (5 weeks ago)  pbt-santos: fix lets in prime.ml
2e9641a  Sat Mar 20 13:57:19 2021 -0400 (5 weeks ago)  nmofficial: updated comments
d19049b  Sat Mar 20 13:23:58 2021 -0400 (5 weeks ago)  nmofficial: commented lines to delete
b71b7e8  Sat Mar 20 16:55:24 2021 +0000 (5 weeks ago)  pbt-santos: Write barebones top-level
cfc02a7  Sat Mar 20 16:19:56 2021 +0000 (5 weeks ago)  pbt-santos: Merge branch 'main' of https://github.com/thomasundo2/Prime into main
89e964f  Sat Mar 20 16:13:59 2021 +0000 (5 weeks ago)  pbt-santos: Add semantic statement check for Return
4a74a25  Fri Mar 19 18:44:38 2021 -0400 (5 weeks ago)  nmofficial: removed unecessary lines
77d5548  Fri Mar 19 14:46:16 2021 -0400 (5 weeks ago)  alex-liebeskind: Merge branch 'main' of https://github.com/thomasundo2/Prime into main
20a736b  Fri Mar 19 14:42:10 2021 -0400 (5 weeks ago)  alex-liebeskind: Codegen attempt #1 (modified MicroC)
a4d3bf8  Fri Mar 19 12:32:45 2021 -0400 (5 weeks ago)  pbt-santos: Update Makefile
f2d91c3  Fri Mar 19 16:30:43 2021 +0000 (5 weeks ago)  pbt-santos: Add semantics for hello world
e172d67  Fri Mar 19 15:46:45 2021 +0000 (5 weeks ago)  pbt-santos: correct parser and scanner for joint compilation
c54da6c  Fri Mar 19 15:27:20 2021 +0000 (5 weeks ago)  pbt-santos: add return statement to hello world test
f634cd9  Thu Mar 18 20:36:45 2021 -0400 (6 weeks ago)  pbt-santos: Merge pull request #6 from thomasundo2/circleci
6cf3276  Thu Mar 18 20:35:40 2021 -0400 (6 weeks ago)  pbt-santos: add codegen file
8898581  Thu Mar 18 14:23:57 2021 -0400 (6 weeks ago)  pbt-santos: change circleci to compile only what's done fully
28aa269  Thu Mar 18 14:22:15 2021 -0400 (6 weeks ago)  pbt-santos: Add void for print. Add stripped down semant check
843b5a6  Wed Mar 17 23:03:38 2021 -0400 (6 weeks ago)  pbt-santos: Fix yaml indent
55c6a97  Wed Mar 17 23:02:23 2021 -0400 (6 weeks ago)  pbt-santos: fix config.yml
a68ce70  Wed Mar 17 22:58:28 2021 -0400 (6 weeks ago)  pbt-santos: Add semantics, update Makefile with microc style
319c64d  Wed Mar 17 22:00:10 2021 -0400 (6 weeks ago)  pbt-santos: Add my take on AST
07bc2f6  Wed Mar 17 21:53:06 2021 -0400 (6 weeks ago)  pbt-santos: Update circle CI and add hello world test
5a2f6da  Tue Mar 16 21:55:09 2021 -0400 (6 weeks ago)  pbt-santos: add execute permissions to the script
a91d272  Tue Mar 16 21:42:17 2021 -0400 (6 weeks ago)  pbt-santos: Update config.yml
edca46c  Mon Mar 15 11:35:43 2021 -0400 (6 weeks ago)  pbt-santos: Add: Create test directory and  hello world file
3b35aad  Mon Mar 15 11:29:59 2021 -0400 (6 weeks ago)  pbt-santos: Add test job to workflow
6d2d118  Mon Mar 15 11:07:06 2021 -0400 (6 weeks ago)  pbt-santos: Add MicroC test and workflow
b289bce  Mon Mar 15 09:48:36 2021 -0400 (6 weeks ago)  pbt-santos: Add first circleci config file
36c1ecd  Mon Mar 15 09:32:22 2021 -0400 (6 weeks ago)  pbt-santos: Give program AST a rule, change "calc" to "prime"
e72b8e2  Tue Feb 23 16:11:32 2021 -0500 (9 weeks ago)  pbt-santos: Merge pull request #5 from thomasundo2/pedro
92c3648  Tue Feb 23 16:04:29 2021 -0500 (9 weeks ago)  pbt-santos: Add parser comments
4b06489  Mon Feb 22 16:22:46 2021 -0500 (9 weeks ago)  pbt-santos: Add point dimensional
```

```
    assignment
e19cb3d  Mon Feb 22 15:50:00 2021 -0500 (9 weeks ago)  pbt-santos: Add access remove
    unneeded tokens
9f9960c  Mon Feb 22 13:23:21 2021 -0500 (9 weeks ago)  pbt-santos: Add Literals and
    point init
312d4c1  Sun Feb 21 23:04:35 2021 -0500 (9 weeks ago)  pbt-santos: Add char and string
    literal scan. Clean Parser
0747515  Sun Feb 21 21:42:17 2021 -0500 (9 weeks ago)  alex-liebeskind: Update README.md
645c4a0  Sun Feb 21 19:03:56 2021 -0700 (9 weeks ago)  Thomas Tran: Merge pull request
    #4 from thomasundo2/pedro
b4d5279  Sun Feb 21 20:59:34 2021 -0500 (9 weeks ago)  pbt-santos: Complete parser
    according to current CFG
d0bf53e  Sun Feb 21 18:30:19 2021 -0500 (9 weeks ago)  pbt-santos: Merge pull request #3
     from thomasundo2/thomas
3c3b756  Sun Feb 21 18:30:05 2021 -0500 (9 weeks ago)  pbt-santos: Merge branch 'main'
    into thomas
181c89f  Sun Feb 21 16:25:07 2021 -0700 (9 weeks ago)  thomasundo2: add brackets
784e08e  Sun Feb 21 18:23:15 2021 -0500 (9 weeks ago)  pbt-santos: Merge pull request #2
     from thomasundo2/pedro
2bb9cd7  Sun Feb 21 16:19:37 2021 -0700 (9 weeks ago)  thomasundo2: remove bool from
    scanner
a0a5a3b  Sun Feb 21 16:18:36 2021 -0700 (9 weeks ago)  thomasundo2: add newly created
    types
6c34fee  Sun Feb 21 16:09:28 2021 -0700 (9 weeks ago)  thomasundo2: add microc scanner
    and a few of our own
c84efcb  Sun Feb 21 12:10:21 2021 -0500 (9 weeks ago)  pbt-santos: Add poly back in
8f37801  Sat Feb 20 21:22:08 2021 -0500 (9 weeks ago)  pbt-santos: Add overload, string
    and char symbol scanning
1d5aa69  Sat Feb 20 21:13:37 2021 -0500 (9 weeks ago)  pbt-santos: FIX: name undefined
    symbols case
154067a  Sat Feb 20 21:12:45 2021 -0500 (9 weeks ago)  pbt-santos: Add scanning for
    rings and pt symbols
4a8e53c  Sat Feb 20 16:32:11 2021 -0700 (9 weeks ago)  Thomas Tran: Merge pull request
    #1 from thomasundo2/pedro
1afb90e  Sat Feb 20 10:47:55 2021 -0500 (9 weeks ago)  pbt-santos: Add symbols to
    scanner. Change = to ASSIGN
024ba7b  Tue Feb 16 18:19:15 2021 -0500 (10 weeks ago)  pbt-santos: Add pretty print
    and test bench file
556d351  Sun Feb 14 20:03:12 2021 -0700 (2 months ago)  thomasundo2: delete made files
894c7d6  Sun Feb 14 20:01:17 2021 -0700 (2 months ago)  thomasundo2: change Makefile to
     only use calc
987b38e  Sun Feb 14 19:58:06 2021 -0700 (2 months ago)  Thomas Tran: Delete calc.tb
02584fa  Sun Feb 14 19:56:51 2021 -0700 (2 months ago)  thomasundo2: add hw1 problem 3
    as base
b54a0fd  Sun Feb 14 13:45:54 2021 -0700 (2 months ago)  Thomas Tran: Initial commit
```

6 Architectural Design



6.1 Scanner

Relevant Files: `scanner.mll`
Implemented by: Alexander Liebeskind, Nikhil Mehta, Pedro B T Santos, Thomas Tran

The scanner takes a program as input and converts into a stream of tokens; these tokens are determined by a set of parsing rules. The tokens include variable names, keywords, types, operators, and literals. Comments and whitespace are ignored by the scanner and not converted to tokens. Relevant files are written in Ocamllex.

6.2 Parser

Relevant Files: `parser.mly`, `ast.ml`
Implemented by: Alexander Liebeskind, Nikhil Mehta, Pedro B T Santos, Thomas Tran

The parser generates an abstract syntax tree (AST) from a syntactically valid stream of tokens – the AST is explicitly defined in `ast.ml`. The hierarchy of parsing from top to bottom is as follows: program, function declarations, statements, and expressions.

Functions to print the AST are also defined in `ast.ml`; the parsed AST of a program can be printed using `Ast.string_of_program`.

6.3 Semantic Checking

Relevant Files: `semant.ml`, `sast.ml`
Implemented by: Alexander Liebeskind, Nikhil Mehta, Pedro B T Santos, Thomas Tran

Generates a type safe, semantically correct AST, otherwise known as an SAST, by detecting type mismatches, invalid assignments, and incorrect parameters prior to runtime. On success, it maps every element in the AST to its equivalent in the SAST, formally defined in `ast.ml`.

The SAST also contains functions for printing the syntax tree along with the semantically checked types.

6.4 Code Generation

Relevant Files: `codegen.ml`
Implemented by: Alexander Liebeskind, Nikhil Mehta, Pedro B T Santos, Thomas Tran

Using the semantically checked SAST, codegen traverses the syntax tree and generates LLVM byte code instructions. To monitor scope, codegen uses StringMap to keep track of local and global variables.

6.5 C Libraries

Relevant Files: `gmpfunc.c`, `structs.c`, `input.c`
Implemented by: Alexander Liebeskind, Nikhil Mehta, Pedro B T Santos, Thomas Tran

C libraries includes external libraries (e.g. GNU Multiple Precision Arithmetic Library), structs, struct declaration, and advanced functions such as point elliptic curve operations linked to the LLVM instructions.

7 Test Plan

7.1 Description

Test case selection:
For all additions, 1-3 test cases were added at a minimum. There was some unit testing of scanner and parser though most tests were done at our Prime language level (an integration test). This testing was intended to make sure all symbols were properly tokenized and to ensure the grammar was unambiguous (i.e. no shift/reduce or reduce/reduce errors). Test cases were also designed to ensure that our semantic checker would properly check types, and argument numbers for functions.

Positive test cases would be to check that functionality we desired was implemented. We used test-driven development, therefore we wrote our test cases and ensured that our compiler passed those cases. If they did not, then we worked on our code until they did. In certain instance, several examples of functioning PRIME code were included in a single test case. This primarily occurred when the feature added was relatively simple or if behavior was similar for several examples. Failing test cases were added to ensure that undefined behavior was not tolerated. We tailored these test cases to make sure that our compiler had as few bugs as possible.

Automation:
The main parts of automation were local and remote.

For local testing, we had the MicroC test_all script slightly modified/simplified. This would run the full regression and integration test suite outputting successes and differences in case of failure. This was key to check if new additions caused prior tests to fail.

For remote testing we used CircleCI, a continuous integration pipeline allowing us to run tests every time that commits were pushed to GitHub. This allowed us several advantages: all can know when the regression test suite is failing and at what particular points (delivered as a notification by email). Every commit shows whether the test suite is failing or successful and the stage of the failure is displayed.

Roles:
Pedro set up the CircleCI environment on Github and the entire team worked on test cases depending on what feature branch they were working on. It was everyone's responsibility to add to the test suite every time that they added a feature.

7.2 Test Cases

Expected output is shown above test code.

Fail Accessing

```
Fatal error: exception Failure("cannot access type: int in i.x")
```

```
int main()
{
    /* access an integer id */
    int i;
    i = 1;
    printl(i.x);
}
```

Fail Decode

```
Fatal error: exception Failure("illegal argument int expected lint in 0")
```

```
int main()
{
    decode(0);
    return 0;
}
```

Fail Decode2

```
Fatal error: exception Failure("illegal assignment lint = string in x = decode(67)")
```

```
int main()
{
    lint x;
    x = decode(67l);
    return 0;
}
```

Fail Encode

```
Fatal error: exception Failure("illegal argument lint expected string in 34")
```

```
int main()
{
    string x;
    x = "fail";
    /* encode(34l, x); UNDEFINED: set but can't reference */
    encode(34l);
}
```

Fail Encode2

```
Fatal error: exception Failure("expecting 1 arguments in encode")
```

```
int main()
{
    lint x;
    printl(encode(x, "3"));
}
```

Fail If else statements

```
Fatal error: exception Failure("expected integer expression in "s"")
```

```
int main()
{
    int test;
    int test2;
    test2 = 2;
    test = 0;
    if("s") {
  print(1);
    }
    else{
  print(0);
    }
    return 0;
}
```

Fail Inverse for Ints

```
Fatal error: exception Failure("illegal binary operator int ‘ int in test1 ‘ test2")
```

```
int main(){
    int test1;
    int test2;
    test1 = 2;
    test2 = 7;
    print(test1 ‘ test2);
    return 0;
}
```

Fail Lint Casting

```
Fatal error: exception Failure("illegal argument lint expected int in 12")
```

```
int main()
{
    lint x;
    x = tolint(12l);
}
```

Fail Lint Operations

```
Fatal error: exception Failure("illegal binary operator lint /\ lint in
    21973469182365874353456 /\ 2")
```

```
int main()
{
    /* for pow we raise by unsigned int */
    printl(21973469182365874353456l /\ 2l);
    return 0;
}
```

Fail Modular Power Op

```
Fatal error: exception Failure("illegal ternary operator lint ^ int @ lint in 3 ^ 5 @ 17
    ")
```

```
int main(){
    printl(3l ^ 5 @ 17l);
    return 0;
}
```

Fail Point Access

```
Fatal error: exception Failure("invalid access element z in a.z")
```

```
int main()
{
    pt a;
    a = [1l, 2l] & [(4l, 5l) : 7l];
    printl(a.z);
}
```

Fail Point Type Mismatch

```
Fatal error: exception Failure("points must have Lint coordinates and be defined under a
     Poly")
```

```
int main()
{
    pt x;
    printpt([192943924239l, 2] & [(1l,2l) : 3l]);
    return 0;
}
```

Fail Point Type Assign

```
Fatal error: exception Failure("points must have Lint coordinates and be defined under a
     Poly")
```

```
int main()
{
    pt x;
    x = ["s", 2l] & [(4l, 5l) : 7l];
    return 0;
}
```

Fail Poly Given Types

```
Fatal error: exception Failure("Polynomials must have Lint coefficients and a Lint
    modulus")
```

```
int main()
{
    curve x;
    x = [(1, 2l) : 3l];
    return 0;
}
```

Fail Point Multiplication

```
Fatal error: exception Failure("illegal binary operator Point * Point in p1 * p1")
```

```
int main()
{
    /* creation and assignment */
    pt p1;
    pt p2;
    curve crv;

```

```
    crv = [(5l, 12l) : 13l];
    p1 = [2l, 2l] & crv;
    p2 = [10l, 10l] & crv;
    printpt(p1 * p1);
    printpt(p1 * 8l);
    return 0;
}
```

Fail Return Statements

```
Fatal error: exception Parsing.Parse_error
```

```
int main()
{
  int x;
  x = 15;
  return x;
  print x;
}
```

Fail Variable assignment and operations

```
Fatal error: exception Failure("illegal assignment int = string in test = "hi"")
```

```
/* Assignment mismatch */
int main()
{
    int test;
    test = "hi";
    return 1;
}
```

Fail While Loop

```
Fatal error: exception Failure("expected integer expression in "s"")
```

```
int main()
{
  int i;

  while ("s") { /* should be integer expression */
    i = i + 1;
  }

  return 0;
}
```

Fail While Loop2

```
Fatal error: exception Parsing.Parse_error
```

```
int main()
{
  int i;

  while (i < 2) {
    i = i + 1;
    int a;
    a = 7;
  }

  print(a);/* out of scope */

  return 0;
}
```

Test Addition

```
3
```

```
int main()
{
    int test;
    print(1+2);
    return 0;
}
```

Test Assignment

```
5
```

```
int main()
{
        int test;
        test = 5;
        print(test);
        return 0;
}
```

Test Lints on big Curves

```
[550676558623444384489421342598247323651822623865,67199985943910549619115318268838008085818100907]
     &
    [(31768908125132550347631746413827693272746955927,48571406791775727346184082881005620597345426652)
     : 785963102379428822376694789446897396207498568951]
```

```
int main()
{
    /* creation variables */
    lint p;
    lint a;
    lint b;

    lint x1;
    lint y1;
    lint x2;
    lint y2;
    lint x3;
    lint y3;

    pt q;
    pt r;
    pt s;

    curve crv;

    /* create the curve */

    /* prime number from Microsoft Digital Rights Management
     * As seen in Lecture Slides 13 - MATH UN3025 Prof. Dorial Goldfeld - November, 2020
     *
     * coefficients and point q geneerated on
     * http://www.christelbach.com/ECCalculator.aspx
     */

    p = 785963102379428822376694789446897396207498568951l;
    a = 317689081251325503476317464138276932727469559271;
    b = 485714067917757273461840828810056205973454266521;

    crv = [(a, b) : p];

    /* use subgroup generator q */

    x1 = 771507216262649826170648268565579889907769254176l;
    y1 = 390157510246556628525279459266514995562533196655l;

    q = [x1, y1] & crv;
```

```
    printpt(q + q);

    return 0;
}
```

Test Large Lint

```
100000000000000000
300000000000000000
2000000000000000000000000000000000
100000000000000000
```

```
int main()
{
    lint l1;
    lint l2;
    l1 = 1000000000000000001;
    l2 = 2000000000000000001;

    printl(l1);
    printl(l1 + l2);
    printl(l1*l2);
    printl(l1^l1@(l1*3l));

}
```

Test Decode

```
HelloWorld
```

```
int main()
{
    lint x;
    string out;
    x = 72101108108111087111114108100l;
    out = decode(x);
    prints(out);
    return 0;
}
```

Test elseif Case

```
1
```

```
int main()
{
    int test;
    test = 0;
    if(test) {
      print(1);
    }
    else if(1){
      print(1);
    }
    return 0;
}
```

Test Encode

```
HelloWorld
72101108108111087111114108100
HelloWorld
```

```
int main()
{
    string in;
    lint encoded;
    in = "HelloWorld";
    prints(in);
```

```
    encoded = encode(in);
    printl(encoded);
    prints(decode(encoded));
    return 0;
}
```

Test For Loop

```
0
1
2
3
4
```

```
int main()
{
    int a;
    int b;
    for(a = 0; a < 5; a = a+1){
        print(a);
    }
    for(a = 5; a < 5; a = a+1){
        print(a);
    }
    return 0;
}
```

Test Function

```
42
```

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

int main()
{
  int a;
  a = add(39, 3);
  print(a);
  return 0;
}
```

Hello World Program

```
0
```

```
int main()
{
    int test;
    print(0);
    return 0;
}
```

Test Basic If Statement

```
1
```

```
int main()
{
    int test;
    test = 1;
    if(test) {
  print(1);
    }
    return 0;
}
```

Test IfElse Statement

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

```
int main()
{
    int test;
    int test2;
    test2 = 2;
    test = 0;
    if(test) {
  print(1);
    }
    else{
  print(0);
    }
    if(test2){
        print(1);
    }
    else{
        print(0);
    }
    if(test2 < 0){
        print(1);
    }
    else{
        print(0);
    }
    if(test2 > 0){
        print(1);
    }
    else{
        print(0);
    }
    if(0 > test2){
        print(1);
    }
    else{
        print(0);
    }
    if(test != test2){
        print(1);
    }
    else{
        print(0);
    }
    if(test == test2){
        print(1);
    }
    else{
        print(0);
    }
    return 0;
}
```

Test If With Variable Condition

```
1
0
```

```
int main(){
    int test;
    test = 5;
    if(test == 5){
        print(1);
    } else {
```

```
        print(0);
    }
    if(test == 6){
        print(1);
    }
    else{
        print(0);
    }
    return 0;
}
```

Test Inverse

```
4
4
4
4
```

```
int main()
{
    lint test1;
    lint test2;
    test2 = 7l;
    test1 = 2l;
    printl(2l ‘ 7l);
    printl(2l ‘ test2);
    printl(test1 ‘ 7l);
    printl(test1 ‘ test2);
    return 0;
}
```

Test Inverse Infinity

```
0
```

```
int main()
{
    printl(2l ‘ 4l);
    return 0;
}
```

Test Lint Operations

```
1
1
1
1
1
1
```

```
int main()
{
    print(2l > 1l);
    print(1l >= 1l);
    print(2l == 2l);
    print(2l != 1l);
    print(1l < 2l);
    print(1l <= 1l);
    return 0;
}
```

Test Lint And

```
1
```

```
int main()
{
    lint test;
    lint test2;
```

```
    test = 1l;
    test2 = 7l;
    if(test && test2) {
  print(1);
    }
    else{
  print(0);
    }

    return 0;
}
```

Test Lint And 1

```
0
```

```
int main()
{
    lint test;
    lint test2;
    test = 1l;
    test2 = 0l;
    if(test && test2) {
  print(1);
    }
    else{
  print(0);
    }

    return 0;
}
```

Test Lint Cast

```
2
1
3
```

```
int main()
{
    int x;
    lint y;
    lint z;
    x = 1;
    y = tolint(2);
    z = tolint(x);
    printl(y);
    printl(z);
    printl(tolint(3));
    return 0;
}
```

Test Lint

```
1234123413412313241321324124132412341 23
```

```
    printl(test);
    printl(12341234134123132413213241241324123412312);
    /* Now test different combinations of literals and IDs */
    printl(test + test);
    printl(test + 121);
    printl(131 + 121);
    /* Now the other operators */
    printl(test /\ 2); /* int on rhs expected */
    printl(test * 21);
    printl((test - 11) / 21);
    printl(test % 21);
    test2 = test;
    test2 = test2 - test;
    printl(test2);
    return 0;
}
```

Test Lint Unary Op

```
0
```

```
int main()
{
    printl(!23946532784568347651);
    return 0;
}
```

Test Lint Unary Op 1

```
0
```

```
int main()
{
    /* Unary operators should not work on lints */
    printl(!23946532784568347651);
    return 0;
}
```

Test Lint Return

```
42
42
```

```
lint retLint(lint a)
{
    printl(a);
    return a;
}

int main()
{
    printl(retLint(421));
}
```

Test Lint Negative

```
-1
-1
-1
```

```
int main()
{
    lint l1;
    lint l2;
    l1 = 1l;
    l2 = -1l;
    printl(l2);
    printl(-l1);
    printl(-1l);
    return 0;
}
```

Test Lint Not

```
0
1
0
1
```

```
int main()
{
   lint l1;
   lint l2;
   l1 = -11l;
   l2 = !l1;
   printl(l2);
   printl(!l2);
   printl(!-23l);
   printl(!0l);
   return 0;
}
```

Test Lint Or

```
1
```

```
int main()
{
    lint test;
    lint test2;
    test = 1l;
    test2 = 0l;
    if(test || test2) {
  print(1);
    }
    else{
  print(0);
    }

    return 0;
}
```

Test Lint Or 1

```
0
```

```
int main()
{
    lint test;
    lint test2;
    test = 0l;
    test2 = 0l;
    if(test || test2) {
  print(1);
    }
    else{
  print(0);
    }

    return 0;
}
```

Test Mod

```
2
```

```
int main()
{
        int test;
        print((5+25)*3%4);
        return 0;
}
```

Test Ternary

```
5
5
15
6
```

```
int main()
{
   lint l1;
   l1 = 3l ^ 5l @ 17l;
   printl(l1);
   printl(3l ^ 5l @ 17l);
   printl(3l ^ 5l + 1l @ 17l);
   printl(3l ^ 3l ^ 5l + 1l @ 17l @ 17l);
   return 0;
}
```

Test Negative

```
-3
```

```
int main()
{
        int test;
        print(-1*3);
        return 0;
}
```

Test Not

```
0
1
0
1
0
```

```
int main()
{
        int test;
        int test2;
        test2 = 5;
        test = 0;
        print(!1);
        print(!0);
        print(!5);
        print(!test);
        print(!test2);
        return 0;
}
```

Test Operations

```
1
1
1
1
1
1
1
1
```

```
int main()
{
  print(1 == 1);
  print(0 != 1);
  print(1 >= 0);
  print(1 <= 2);
  print(2 > 1);
        print(0 < 1);
```

```
  print(20 && 20);
        print(20 || 0);
        return 0;
}
```

Test Point

```
[1,2] & [(100,200) : 400]
[5,6] & [(500,600) : 700]
```

```
int main()
{
    /* creation and assignment */
    pt x;
    curve mycurve;
    printpt([11, 21] & [(1001,2001) : 4001]);


    mycurve = [(5001,6001) : 7001];
    x = [51, 61] & mycurve;
    printpt(x);
    return 0;
}
```

Test Point Access

```
1
2
```

```
int main()
{
    pt a;
    a = [11, 21] & [(41, 51) : 71];
    printl(a.x);
    printl(a.y);
    /* Will add the poly element soon */
}
```

Test Point Return

```
[0,1] & [(1,2) : 2]
[0,1] & [(1,2) : 2]
```

```
pt retPoint(pt a)
{
    printpt(a);
    return a;
}

int main()
{
    curve mycurve;
    pt b;
    mycurve = [(11, 21) : 21];
    b = [01, 11] & mycurve;
    printpt(retPoint(b));
    return 0;
}
```

Test Polynomial

```
[(1,2) : 3]
[(4,5) : 6]
```

```
int main()
{
    curve x;
    x = [(11, 21) : 31];
    printc(x);
```

```
    printc([(41, 51) : 61]);
    return 0;
}
```

Test Precedence

```
4
```

```
/* Check integer variable assignment and binops */
int main()
{
    print(1 + 2 * 4 / 2 - 1);
    return 0;
}
```

Test Print

```
test
```

```
int main()
{
    int test;
    prints("test");
    return 0;
}
```

Test Print If

```
2
1
0
1
1
```

```
int main()
{
    int test;
    print((1==1) + 1);
    if(1==1){
        print(1==1);
    }
    if(!(1!=1)){
        print(1==2);
    }
    if((1<2)+1){
        print(1<2);
    }
    if(1>=2){
        print(404);
    }
    else{
        print(1);
    }
    return 0;
}
```

Test Point Add

```
[0,5] & [(5,12) : 13]
[0,5] & [(5,12) : 13]
[0,5] & [(5,12) : 13]
```

```
int main()
{
    /* creation and assignment */
    pt p1;
    pt p2;
    pt p3;
    curve crv;
```

```
    crv = [(5l, 12l) : 13l];
    p1 = [2l, 2l] & crv;
    p2 = [7l, 0l] & crv;
    printpt(p2 + p1);
    p3 = p1 + p2;
    printpt(p3);
    printpt([2l, 2l] & [(5l, 12l) : 13l] + [7l, 0l] & [(5l, 12l) : 13l]);

    return 0;
}
```

Test Point Add Inf

```
[7,0] & [(5,12) : 13]
[7,0] & [(5,12) : 13]
[7,0] & [(5,12) : 13]
[7,0] & [(5,12) : 13]
```

```
int main()
{
    /* creation and assignment */
    pt p1;
    pt p2;
    pt p3;
    lint m1;
    curve crv;

    m1 = 1l - 2l;

    crv = [(5l, 12l) : 13l];
    p1 = [m1, m1] & crv;
    p2 = [7l, 0l] & crv;
    printpt(p2 + p1);
    p3 = p2 + p1;
    printpt(p3);
    printpt([m1, m1] & [(5l, 12l) : 13l] + [7l, 0l] & [(5l, 12l) : 13l]);
    printpt([7l, 0l] & [(5l, 12l) : 13l] + [m1, m1] & [(5l, 12l) : 13l]);

    return 0;
}
```

Test Point Add Inv

```
[-1,-1] & [(5,12) : 13]
[-1,-1] & [(5,12) : 13]
[-1,-1] & [(5,12) : 13]
```

```
int main()
{
    /* creation and assignment */
    pt p1;
    pt p2;
    pt p3;
    curve crv;

    crv = [(5l, 12l) : 13l];
    p1 = [2l, 2l] & crv;
    p2 = [2l, 11l] & crv;
    printpt(p2 + p1);
    p3 = p1 + p2;
    printpt(p3);
    printpt([2l, 2l] & [(5l, 12l) : 13l] + [2l, 11l] & [(5l, 12l) : 13l]);

    return 0;
}
```

Test Point Add Same

```
[10,10] & [(5,12) : 13]
[10,10] & [(5,12) : 13]
[10,10] & [(5,12) : 13]
```

```
int main()
{
    /* creation and assignment */
    pt p1;
    pt p2;
    pt p3;
    curve crv;

    crv = [(51, 121) : 131];
    p1 = [21, 111] & crv;
    p2 = [21, 111] & crv;
    printpt(p2 + p1);
    p3 = p1 + p2;
    printpt(p3);
    printpt([21, 111] & [(51, 121) : 131] + [21, 111] & [(51, 121) : 131]);

    return 0;
}
```

Test Point Eq

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

```
int main()
{
    /* creation and assignment */
    pt p1;
    pt p2;
    pt p3;
    curve crv;

    crv = [(51, 121) : 131];
    p1 = [21, 21] & crv;
    p2 = [21, 21] & crv;
    p3 = [21, 111] & crv;
    print(p1 == p2);
    print(p2 == p1);
    print([21, 21] & crv == p1);
    print(p1 == p3);
    print(p3 == p1);
    print([21, 21] & crv == p3);
    return 0;
}
```

Test Point Mul

```
[-1,-1] & [(5,12) : 13]
[-1,-1] & [(5,12) : 13]
```

```
int main()
{
    /* creation and assignment */
    pt p1;
    pt p2;
    curve crv;

    crv = [(51, 121) : 131];
    p1 = [21, 21] & crv;
    printpt(81 * p1);
    printpt(p1 * 81);
    return 0;
}
```

Test Point Neg

```
[2,11] & [(5,12) : 13]
[2,11] & [(5,12) : 13]
[2,11] & [(5,12) : 13]
[-1,-1] & [(5,12) : 13]
```

```
int main()
{
    /* creation and assignment */
    pt p1;
    pt p2;
    pt p3;
    curve crv;

    crv = [(5l, 12l) : 13l];
    p1 = [2l, 2l] & crv;
    p2 = -p1;
    printpt(-p1);
    printpt(p2);
    printpt(-[2l, 2l] & crv);
    printpt(-[-1l, -1l] & crv);
    return 0;
}
```

Test Point Neq

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

```
int main()
{
    /* creation and assignment */
    pt p1;
    pt p2;
    pt p3;
    curve crv;

    crv = [(5l, 12l) : 13l];
    p1 = [2l, 2l] & crv;
    p2 = [2l, 2l] & crv;
    p3 = [2l, 11l] & crv;
    print(p1 != p2);
    print(p2 != p1);
    print([2l, 2l] & crv != p1);
    print(p1 != p3);
    print(p3 != p1);
    print([2l, 2l] & crv != p3);
    return 0;
}
```

Test Random

```
59
3862285939510455010053594385244193234435l5
6098
```

```
int main()
{
   lint l1;
   lint l2;
   lint l3;
   l1 = 1l;
   l2 = 100l;
   l3 = 10000000000000000000000000000000000000000l;
   printl(random(l1,l2));
   printl(random(l1,l3));
```

```
    printl(random(11,100001));
    return 0;
}
```

Test Recursion

```
25
```

```
int gcd(int a, int b)
{
  if(b != 0){
    return gcd(b, a % b);
  }
  else{
    return a;
  }
}

int main()
{
  int a;
  int b;
  int c;
  a = 25;
  b = 100;
  c = gcd(a, b);
  print(c);
  return 0;
}
```

Test Return

```
15
```

```
int main()
{
  int i;

  i = 15;
  print(i);
  return 0;
  /* Notes after a return are ok */
}
```

Test String

```
hi
```

```
int main()
{
    string s;
    s = "hello";
    s = "hi";
    prints(s);
    return 0;

}
```

Test Variable

```
4
```

```
/* Check integer variable assignment and binops */
int main()
{
    int test;
    test = 1;
    test = 1 + 2 * 4 / 2 - 1;
    print(test);
    return 0;
}
```

Test Variable And Or

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

```
int main(){
    int i;
    int j;
    i = 5;
    j = 0;
    print(i && j);
    print(i || j);
    print(0 || j);
    print(i && 1);
    print(i && 0);
    print(j && 1);
    return 0;
}
```

Test While

```
5
```

```
int main()
{
    int test;
    int a;
    a = 0;
    test = 0;
    while (a < 5) {
  test = test + 1;
        a = a + 1;
    }
    print(test);
    return 0;
}
```

7.3 Demonstration

Demonstration code is shown above sample generated LLVM code.

Diffie-Hellman Key Exchange on Elliptic Curves

```
pt alice_cpk(pt q) /* Alice computes public key */
{
  lint alpha;
  alpha = 537l;
  return alpha*q;
}

pt bob_cpk(pt q) /* Bob computes public key */
{
  lint beta;
  beta = 792l;
  return beta*q;
}

int alice_csecret(pt b_public_key) /* Alice computes shared secret */
{
  lint alpha;
  alpha = 537l; /* Alice still has access to alpha */
  prints("Alice's Computed Shared Secret:");
  printpt(alpha*b_public_key);
  return 0;
}

```

```
int bob_csecret(pt a_public_key) /* Bob computes shared secret */
{
  lint beta;
  beta = 7921; /* Bob still has access to beta */
  prints("Bob's Computed Shared Secret:");
  printpt(beta*a_public_key);
  return 0;
}

int main()
{
  /* Diffie-Hellman Key Exchange on Elliptic Curves */
  /* prime number from Microsoft Digital Rights Management
   * As seen in Lecture Slides 13 - MATH UN3025 Prof. Dorial Goldfeld - November, 2020
   *
   * coefficients and point q geneerated on
   * http://www.christelbach.com/ECCalculator.aspx
  */

  /* create variables */

  lint p;
  lint a;
  lint b;

  lint x1;
  lint y1;

  pt q;
  pt a_public_key;
  pt b_public_key;

  curve crv;

  /* create the curve */

  p = 785963102379428822376694789446897396207498568951l;
  a = 317689081251325503476317464138276932727469559271l;
  b = 485714067917757273461840828810056205973454266521l;

  crv = [(a, b) : p];

  /* use subgroup generator q */

  x1 = 771507216262649826170648268565579889907769254176l;
  y1 = 390157510246556628525279459266514995562533196655l;

  prints("Elliptic Curve E:");
  printc(crv);

  /* create the point */

  q = [x1, y1] & crv;
  prints("Point q:");
  printpt(q);

  /* Alice computes a Public Key using private alpha */
  a_public_key = alice_cpk(q);

  /* Bob computes a Public Key using private beta */
  b_public_key = bob_cpk(q);

  /* Alice and Bob compute their shared secret using the public transmission from the
    other */
  alice_csecret(b_public_key);
  bob_csecret(a_public_key);

  return 0;
}
```

```
; ModuleID = 'Prime'
source_filename = "Prime"

```

```
%mpz_t = type { i32, i32, i64* }
%point = type { %mpz_t, %mpz_t, %poly }
%poly = type { %mpz_t, %mpz_t, %mpz_t }

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@string = private unnamed_addr constant [49 x i8] c
    "785963102379428822376694789446897396207498568951\00"
@string.2 = private unnamed_addr constant [48 x i8] c
    "31768908125132550347631746413827693272746955927\00"
@string.3 = private unnamed_addr constant [48 x i8] c
    "48571406791775727346184082881005620597345426652\00"
@string.4 = private unnamed_addr constant [49 x i8] c
    "771507216262649826170648268565579889907769254176\00"
@string.5 = private unnamed_addr constant [49 x i8] c
    "390157510246556628525279459266514995562533196655\00"
@string.6 = private unnamed_addr constant [18 x i8] c"Elliptic Curve E:\00"
@string.7 = private unnamed_addr constant [9 x i8] c"Point q:\00"
@fmt.8 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.9 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@string.10 = private unnamed_addr constant [4 x i8] c"792\00"
@string.11 = private unnamed_addr constant [30 x i8] c"Bob's Computed Shared Secret:\00"
@fmt.12 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.13 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@string.14 = private unnamed_addr constant [4 x i8] c"537\00"
@string.15 = private unnamed_addr constant [32 x i8] c"Alice's Computed Shared Secret
    :\00"
@fmt.16 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.17 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@string.18 = private unnamed_addr constant [4 x i8] c"792\00"
@fmt.19 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.20 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@string.21 = private unnamed_addr constant [4 x i8] c"537\00"

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

declare i32 @__gmpz_init_set_str(%mpz_t*, i8*, i32)

declare i32 @__gmpz_init_set_si(%mpz_t*, i32)

declare i32 @__gmpz_init_set(%mpz_t*, %mpz_t*)

declare i32 @printl(%mpz_t*)

declare i32 @__gmpz_add(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_sub(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_mul(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_tdiv_q(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_tdiv_r(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_pow_ui(%mpz_t*, %mpz_t*, i32)

declare i32 @__gmpz_invert(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_powm(%mpz_t*, %mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_neg(%mpz_t*, %mpz_t*)

declare i32 @lnot_func(%mpz_t*, %mpz_t*)

declare i32 @eq_func(%mpz_t*, %mpz_t*)

declare i32 @neq_func(%mpz_t*, %mpz_t*)

declare i32 @lth_func(%mpz_t*, %mpz_t*)

declare i32 @gth_func(%mpz_t*, %mpz_t*)

declare i32 @leq_func(%mpz_t*, %mpz_t*)
```

```
declare i32 @or_func(%mpz_t*, %mpz_t*)

declare i32 @and_func(%mpz_t*, %mpz_t*)

declare i32 @geq_func(%mpz_t*, %mpz_t*)

declare i32 @rand_func(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @Point(%point*, %mpz_t*, %mpz_t*, %poly*)

declare i32 @printpt(%point*)

declare %point* @ptadd(%point*, %point*)

declare %point* @ptmul(%mpz_t*, %point*)

declare %point* @ptneg(%point*)

declare i32 @pteq(%point*, %point*)

declare i32 @ptneq(%point*, %point*)

declare i32 @Poly(%poly*, %mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @printc(%poly*)

declare i32 @encode(%mpz_t*, i8*)

declare i8* @decode(%mpz_t*)

define i32 @main() {
entry:
  %p = alloca %mpz_t
  %a = alloca %mpz_t
  %b = alloca %mpz_t
  %x1 = alloca %mpz_t
  %y1 = alloca %mpz_t
  %q = alloca %point
  %a_public_key = alloca %point
  %b_public_key = alloca %point
  %crv = alloca %poly
  %0 = alloca %mpz_t
  %1 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %__gmpz_init_set_str = call i32 @__gmpz_init_set_str(%mpz_t* %1, i8* getelementptr
    inbounds ([49 x i8], [49 x i8]* @string, i32 0, i32 0), i32 10)
  %2 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %p1 = getelementptr inbounds %mpz_t, %mpz_t* %p, i32 0
  %3 = call i32 @__gmpz_init_set(%mpz_t* %p1, %mpz_t* %2)
  %4 = alloca %mpz_t
  %5 = getelementptr inbounds %mpz_t, %mpz_t* %4, i32 0
  %__gmpz_init_set_str2 = call i32 @__gmpz_init_set_str(%mpz_t* %5, i8* getelementptr
    inbounds ([48 x i8], [48 x i8]* @string.2, i32 0, i32 0), i32 10)
  %6 = getelementptr inbounds %mpz_t, %mpz_t* %4, i32 0
  %a3 = getelementptr inbounds %mpz_t, %mpz_t* %a, i32 0
  %7 = call i32 @__gmpz_init_set(%mpz_t* %a3, %mpz_t* %6)
  %8 = alloca %mpz_t
  %9 = getelementptr inbounds %mpz_t, %mpz_t* %8, i32 0
  %__gmpz_init_set_str4 = call i32 @__gmpz_init_set_str(%mpz_t* %9, i8* getelementptr
    inbounds ([48 x i8], [48 x i8]* @string.3, i32 0, i32 0), i32 10)
  %10 = getelementptr inbounds %mpz_t, %mpz_t* %8, i32 0
  %b5 = getelementptr inbounds %mpz_t, %mpz_t* %b, i32 0
  %11 = call i32 @__gmpz_init_set(%mpz_t* %b5, %mpz_t* %10)
  %a6 = getelementptr inbounds %mpz_t, %mpz_t* %a, i32 0
  %b7 = getelementptr inbounds %mpz_t, %mpz_t* %b, i32 0
  %p8 = getelementptr inbounds %mpz_t, %mpz_t* %p, i32 0
  %tmp_poly = alloca %poly
  %Poly = call i32 @Poly(%poly* %tmp_poly, %mpz_t* %a6, %mpz_t* %b7, %mpz_t* %p8)
  %12 = getelementptr inbounds %poly, %poly* %tmp_poly, i32 0
  %13 = load %poly, %poly* %12
  store %poly %13, %poly* %crv
  %14 = alloca %mpz_t
  %15 = getelementptr inbounds %mpz_t, %mpz_t* %14, i32 0
```

```
  %__gmpz_init_set_str9 = call i32 @__gmpz_init_set_str(%mpz_t* %15, i8* getelementptr
    inbounds ([49 x i8], [49 x i8]* @string.4, i32 0, i32 0), i32 10)
  %16 = getelementptr inbounds %mpz_t, %mpz_t* %14, i32 0
  %x110 = getelementptr inbounds %mpz_t, %mpz_t* %x1, i32 0
  %17 = call i32 @__gmpz_init_set(%mpz_t* %x110, %mpz_t* %16)
  %18 = alloca %mpz_t
  %19 = getelementptr inbounds %mpz_t, %mpz_t* %18, i32 0
  %__gmpz_init_set_str11 = call i32 @__gmpz_init_set_str(%mpz_t* %19, i8* getelementptr
    inbounds ([49 x i8], [49 x i8]* @string.5, i32 0, i32 0), i32 10)
  %20 = getelementptr inbounds %mpz_t, %mpz_t* %18, i32 0
  %y112 = getelementptr inbounds %mpz_t, %mpz_t* %y1, i32 0
  %21 = call i32 @__gmpz_init_set(%mpz_t* %y112, %mpz_t* %20)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
    @fmt.1, i32 0, i32 0), i8* getelementptr inbounds ([18 x i8], [18 x i8]* @string.6,
    i32 0, i32 0))
  %crv13 = getelementptr inbounds %poly, %poly* %crv, i32 0
  %printc = call i32 @printc(%poly* %crv13)
  %x114 = getelementptr inbounds %mpz_t, %mpz_t* %x1, i32 0
  %y115 = getelementptr inbounds %mpz_t, %mpz_t* %y1, i32 0
  %crv16 = getelementptr inbounds %poly, %poly* %crv, i32 0
  %tmp_pt = alloca %point
  %Point = call i32 @Point(%point* %tmp_pt, %mpz_t* %x114, %mpz_t* %y115, %poly* %crv16)
  %22 = getelementptr inbounds %point, %point* %tmp_pt, i32 0
  %23 = load %point, %point* %22
  store %point %23, %point* %q
  %printf17 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8
    ]* @fmt.1, i32 0, i32 0), i8* getelementptr inbounds ([9 x i8], [9 x i8]* @string.7,
     i32 0, i32 0))
  %q18 = getelementptr inbounds %point, %point* %q, i32 0
  %printpt = call i32 @printpt(%point* %q18)
  %q19 = getelementptr inbounds %point, %point* %q, i32 0
  %pt_param = load %point, %point* %q19
  %sret_space = alloca %point
  %24 = getelementptr inbounds %point, %point* %sret_space, i32 0
  %alice_cpk_result = call %point* @alice_cpk(%point* %24, %point %pt_param)
  %25 = getelementptr inbounds %point, %point* %alice_cpk_result, i32 0
  %26 = load %point, %point* %25
  store %point %26, %point* %a_public_key
  %q20 = getelementptr inbounds %point, %point* %q, i32 0
  %pt_param21 = load %point, %point* %q20
  %sret_space22 = alloca %point
  %27 = getelementptr inbounds %point, %point* %sret_space22, i32 0
  %bob_cpk_result = call %point* @bob_cpk(%point* %27, %point %pt_param21)
  %28 = getelementptr inbounds %point, %point* %bob_cpk_result, i32 0
  %29 = load %point, %point* %28
  store %point %29, %point* %b_public_key
  %b_public_key23 = getelementptr inbounds %point, %point* %b_public_key, i32 0
  %pt_param24 = load %point, %point* %b_public_key23
  %alice_csecret_result = call i32 @alice_csecret(%point %pt_param24)
  %a_public_key25 = getelementptr inbounds %point, %point* %a_public_key, i32 0
  %pt_param26 = load %point, %point* %a_public_key25
  %bob_csecret_result = call i32 @bob_csecret(%point %pt_param26)
  ret i32 0
}

define i32 @bob_csecret(%point %a_public_key) {
entry:
  %a_public_key1 = alloca %point
  store %point %a_public_key, %point* %a_public_key1
  %beta = alloca %mpz_t
  %0 = alloca %mpz_t
  %1 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %__gmpz_init_set_str = call i32 @__gmpz_init_set_str(%mpz_t* %1, i8* getelementptr
    inbounds ([4 x i8], [4 x i8]* @string.10, i32 0, i32 0), i32 10)
  %2 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %beta2 = getelementptr inbounds %mpz_t, %mpz_t* %beta, i32 0
  %3 = call i32 @__gmpz_init_set(%mpz_t* %beta2, %mpz_t* %2)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
    @fmt.9, i32 0, i32 0), i8* getelementptr inbounds ([30 x i8], [30 x i8]* @string.11,
     i32 0, i32 0))
  %beta3 = getelementptr inbounds %mpz_t, %mpz_t* %beta, i32 0
  %a_public_key4 = getelementptr inbounds %point, %point* %a_public_key1, i32 0
  %pt_mul = call %point* @ptmul(%mpz_t* %beta3, %point* %a_public_key4)
```

```
  %printpt = call i32 @printpt(%point* %pt_mul)
  ret i32 0
}

define i32 @alice_csecret(%point %b_public_key) {
entry:
  %b_public_key1 = alloca %point
  store %point %b_public_key, %point* %b_public_key1
  %alpha = alloca %mpz_t
  %0 = alloca %mpz_t
  %1 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %__gmpz_init_set_str = call i32 @__gmpz_init_set_str(%mpz_t* %1, i8* getelementptr inbounds ([4 x i8], [4 x i8]* @string.14, i32 0, i32 0), i32 10)
  %2 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %alpha2 = getelementptr inbounds %mpz_t, %mpz_t* %alpha, i32 0
  %3 = call i32 @__gmpz_init_set(%mpz_t* %alpha2, %mpz_t* %2)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.13, i32 0, i32 0), i8* getelementptr inbounds ([32 x i8], [32 x i8]* @string.15, i32 0, i32 0))
  %alpha3 = getelementptr inbounds %mpz_t, %mpz_t* %alpha, i32 0
  %b_public_key4 = getelementptr inbounds %point, %point* %b_public_key1, i32 0
  %pt_mul = call %point* @ptmul(%mpz_t* %alpha3, %point* %b_public_key4)
  %printpt = call i32 @printpt(%point* %pt_mul)
  ret i32 0
}

define %point* @bob_cpk(%point* %sret, %point %q) {
entry:
  %sret1 = alloca %point*
  store %point* %sret, %point** %sret1
  %q2 = alloca %point
  store %point %q, %point* %q2
  %beta = alloca %mpz_t
  %0 = alloca %mpz_t
  %1 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %__gmpz_init_set_str = call i32 @__gmpz_init_set_str(%mpz_t* %1, i8* getelementptr inbounds ([4 x i8], [4 x i8]* @string.18, i32 0, i32 0), i32 10)
  %2 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %beta3 = getelementptr inbounds %mpz_t, %mpz_t* %beta, i32 0
  %3 = call i32 @__gmpz_init_set(%mpz_t* %beta3, %mpz_t* %2)
  %beta4 = getelementptr inbounds %mpz_t, %mpz_t* %beta, i32 0
  %q5 = getelementptr inbounds %point, %point* %q2, i32 0
  %pt_mul = call %point* @ptmul(%mpz_t* %beta4, %point* %q5)
  %4 = load %point, %point* %pt_mul
  %ret_ptr = load %point*, %point** %sret1
  store %point %4, %point* %ret_ptr
  ret %point* %ret_ptr
}

define %point* @alice_cpk(%point* %sret, %point %q) {
entry:
  %sret1 = alloca %point*
  store %point* %sret, %point** %sret1
  %q2 = alloca %point
  store %point %q, %point* %q2
  %alpha = alloca %mpz_t
  %0 = alloca %mpz_t
  %1 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %__gmpz_init_set_str = call i32 @__gmpz_init_set_str(%mpz_t* %1, i8* getelementptr inbounds ([4 x i8], [4 x i8]* @string.21, i32 0, i32 0), i32 10)
  %2 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %alpha3 = getelementptr inbounds %mpz_t, %mpz_t* %alpha, i32 0
  %3 = call i32 @__gmpz_init_set(%mpz_t* %alpha3, %mpz_t* %2)
  %alpha4 = getelementptr inbounds %mpz_t, %mpz_t* %alpha, i32 0
  %q5 = getelementptr inbounds %point, %point* %q2, i32 0
  %pt_mul = call %point* @ptmul(%mpz_t* %alpha4, %point* %q5)
  %4 = load %point, %point* %pt_mul
  %ret_ptr = load %point*, %point** %sret1
  store %point %4, %point* %ret_ptr
  ret %point* %ret_ptr
}
```

Elliptic Curve Cryptography

```
int main()
{
    /* creation variables */
    lint p;
    lint a;
    lint b;

    lint x1;
    lint y1;
    lint x2;
    lint y2;
    lint x3;
    lint y3;

    pt q;
    pt r;
    pt s;

    curve crv;

    /* create the curve */

    /* prime number from Microsoft Digital Rights Management
     * As seen in Lecture Slides 13 - MATH UN3025 Prof. Dorial Goldfeld - November, 2020
     *
     * coefficients and point q geneerated on
     * http://www.christelbach.com/ECCalculator.aspx
     */

    p = 785963102379428822376694789446897396207498568951l;
    a = 317689081251325503476317464138276932727469559271l;
    b = 485714067917757273461840828810056205973454266521l;

    crv = [(a, b) : p];

    /* use subgroup generator q */

    x1 = 771507216262649826170648268565579889907769254176l;
    y1 = 390157510246556628525279459266514995562533196655l;

    prints("Elliptic Curve E:");
    printc(crv);

    q = [x1, y1] & crv;
    prints(""); prints("Point q:");
    printpt(q);

    r = q + q;
    prints(""); prints("Point r = q + q");
    printpt(r);

    prints(""); prints("-q:");
    printpt(-q);

    prints(""); prints("q + -q:");
    printpt(q + -q);

    s = 123l*q;
    prints(""); prints("123*q");
    printpt(s);

    return 0;
}
```

```
; ModuleID = 'Prime'
source_filename = "Prime"

%mpz_t = type { i32, i32, i64* }
%point = type { %mpz_t, %mpz_t, %poly }
%poly = type { %mpz_t, %mpz_t, %mpz_t }

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
```

```
@fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@string = private unnamed_addr constant [49 x i8] c
    "785963102379428822376694789446897396207498568951\00"
@string.2 = private unnamed_addr constant [48 x i8] c
    "31768908125132550347631746413827693272746955927\00"
@string.3 = private unnamed_addr constant [48 x i8] c
    "48571406791775727346184082881005620597345426652\00"
@string.4 = private unnamed_addr constant [49 x i8] c
    "771507216262649826170648268565579889907769254176\00"
@string.5 = private unnamed_addr constant [49 x i8] c
    "390157510246556628525279459266514995562533196655\00"
@string.6 = private unnamed_addr constant [18 x i8] c"Elliptic Curve E:\00"
@string.7 = private unnamed_addr constant [9 x i8] c"Point q:\00"
@fmt.8 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.9 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@string.10 = private unnamed_addr constant [4 x i8] c"792\00"
@string.11 = private unnamed_addr constant [30 x i8] c"Bob's Computed Shared Secret:\00"
@fmt.12 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.13 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@string.14 = private unnamed_addr constant [4 x i8] c"537\00"
@string.15 = private unnamed_addr constant [32 x i8] c"Alice's Computed Shared Secret
    :\00"
@fmt.16 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.17 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@string.18 = private unnamed_addr constant [4 x i8] c"792\00"
@fmt.19 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.20 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@string.21 = private unnamed_addr constant [4 x i8] c"537\00"

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

declare i32 @__gmpz_init_set_str(%mpz_t*, i8*, i32)

declare i32 @__gmpz_init_set_si(%mpz_t*, i32)

declare i32 @__gmpz_init_set(%mpz_t*, %mpz_t*)

declare i32 @printl(%mpz_t*)

declare i32 @__gmpz_add(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_sub(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_mul(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_tdiv_q(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_tdiv_r(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_pow_ui(%mpz_t*, %mpz_t*, i32)

declare i32 @__gmpz_invert(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_powm(%mpz_t*, %mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_neg(%mpz_t*, %mpz_t*)

declare i32 @lnot_func(%mpz_t*, %mpz_t*)

declare i32 @eq_func(%mpz_t*, %mpz_t*)

declare i32 @neq_func(%mpz_t*, %mpz_t*)

declare i32 @lth_func(%mpz_t*, %mpz_t*)

declare i32 @gth_func(%mpz_t*, %mpz_t*)

declare i32 @leq_func(%mpz_t*, %mpz_t*)

declare i32 @or_func(%mpz_t*, %mpz_t*)

declare i32 @and_func(%mpz_t*, %mpz_t*)

```

```
declare i32 @geq_func(%mpz_t*, %mpz_t*)

declare i32 @rand_func(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @Point(%point*, %mpz_t*, %mpz_t*, %poly*)

declare i32 @printpt(%point*)

declare %point* @ptadd(%point*, %point*)

declare %point* @ptmul(%mpz_t*, %point*)

declare %point* @ptneg(%point*)

declare i32 @pteq(%point*, %point*)

declare i32 @ptneq(%point*, %point*)

declare i32 @Poly(%poly*, %mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @printc(%poly*)

declare i32 @encode(%mpz_t*, i8*)

declare i8* @decode(%mpz_t*)

define i32 @main() {
entry:
  %p = alloca %mpz_t
  %a = alloca %mpz_t
  %b = alloca %mpz_t
  %x1 = alloca %mpz_t
  %y1 = alloca %mpz_t
  %q = alloca %point
  %a_public_key = alloca %point
  %b_public_key = alloca %point
  %crv = alloca %poly
  %0 = alloca %mpz_t
  %1 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %__gmpz_init_set_str = call i32 @__gmpz_init_set_str(%mpz_t* %1, i8* getelementptr
    inbounds ([49 x i8], [49 x i8]* @string, i32 0, i32 0), i32 10)
  %2 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %p1 = getelementptr inbounds %mpz_t, %mpz_t* %p, i32 0
  %3 = call i32 @__gmpz_init_set(%mpz_t* %p1, %mpz_t* %2)
  %4 = alloca %mpz_t
  %5 = getelementptr inbounds %mpz_t, %mpz_t* %4, i32 0
  %__gmpz_init_set_str2 = call i32 @__gmpz_init_set_str(%mpz_t* %5, i8* getelementptr
    inbounds ([48 x i8], [48 x i8]* @string.2, i32 0, i32 0), i32 10)
  %6 = getelementptr inbounds %mpz_t, %mpz_t* %4, i32 0
  %a3 = getelementptr inbounds %mpz_t, %mpz_t* %a, i32 0
  %7 = call i32 @__gmpz_init_set(%mpz_t* %a3, %mpz_t* %6)
  %8 = alloca %mpz_t
  %9 = getelementptr inbounds %mpz_t, %mpz_t* %8, i32 0
  %__gmpz_init_set_str4 = call i32 @__gmpz_init_set_str(%mpz_t* %9, i8* getelementptr
    inbounds ([48 x i8], [48 x i8]* @string.3, i32 0, i32 0), i32 10)
  %10 = getelementptr inbounds %mpz_t, %mpz_t* %8, i32 0
  %b5 = getelementptr inbounds %mpz_t, %mpz_t* %b, i32 0
  %11 = call i32 @__gmpz_init_set(%mpz_t* %b5, %mpz_t* %10)
  %a6 = getelementptr inbounds %mpz_t, %mpz_t* %a, i32 0
  %b7 = getelementptr inbounds %mpz_t, %mpz_t* %b, i32 0
  %p8 = getelementptr inbounds %mpz_t, %mpz_t* %p, i32 0
  %tmp_poly = alloca %poly
  %Poly = call i32 @Poly(%poly* %tmp_poly, %mpz_t* %a6, %mpz_t* %b7, %mpz_t* %p8)
  %12 = getelementptr inbounds %poly, %poly* %tmp_poly, i32 0
  %13 = load %poly, %poly* %12
  store %poly %13, %poly* %crv
  %14 = alloca %mpz_t
  %15 = getelementptr inbounds %mpz_t, %mpz_t* %14, i32 0
  %__gmpz_init_set_str9 = call i32 @__gmpz_init_set_str(%mpz_t* %15, i8* getelementptr
    inbounds ([49 x i8], [49 x i8]* @string.4, i32 0, i32 0), i32 10)
  %16 = getelementptr inbounds %mpz_t, %mpz_t* %14, i32 0
  %x110 = getelementptr inbounds %mpz_t, %mpz_t* %x1, i32 0
  %17 = call i32 @__gmpz_init_set(%mpz_t* %x110, %mpz_t* %16)
```

```
  %18 = alloca %mpz_t
  %19 = getelementptr inbounds %mpz_t, %mpz_t* %18, i32 0
  %__gmpz_init_set_str11 = call i32 @__gmpz_init_set_str(%mpz_t* %19, i8* getelementptr
    inbounds ([49 x i8], [49 x i8]* @string.5, i32 0, i32 0), i32 10)
  %20 = getelementptr inbounds %mpz_t, %mpz_t* %18, i32 0
  %y112 = getelementptr inbounds %mpz_t, %mpz_t* %y1, i32 0
  %21 = call i32 @__gmpz_init_set(%mpz_t* %y112, %mpz_t* %20)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
    @fmt.1, i32 0, i32 0), i8* getelementptr inbounds ([18 x i8], [18 x i8]* @string.6,
    i32 0, i32 0))
  %crv13 = getelementptr inbounds %poly, %poly* %crv, i32 0
  %printc = call i32 @printc(%poly* %crv13)
  %x114 = getelementptr inbounds %mpz_t, %mpz_t* %x1, i32 0
  %y115 = getelementptr inbounds %mpz_t, %mpz_t* %y1, i32 0
  %crv16 = getelementptr inbounds %poly, %poly* %crv, i32 0
  %tmp_pt = alloca %point
  %Point = call i32 @Point(%point* %tmp_pt, %mpz_t* %x114, %mpz_t* %y115, %poly* %crv16)
  %22 = getelementptr inbounds %point, %point* %tmp_pt, i32 0
  %23 = load %point, %point* %22
  store %point %23, %point* %q
  %printf17 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8
    ]* @fmt.1, i32 0, i32 0), i8* getelementptr inbounds ([9 x i8], [9 x i8]* @string.7,
     i32 0, i32 0))
  %q18 = getelementptr inbounds %point, %point* %q, i32 0
  %printpt = call i32 @printpt(%point* %q18)
  %q19 = getelementptr inbounds %point, %point* %q, i32 0
  %pt_param = load %point, %point* %q19
  %sret_space = alloca %point
  %24 = getelementptr inbounds %point, %point* %sret_space, i32 0
  %alice_cpk_result = call %point* @alice_cpk(%point* %24, %point %pt_param)
  %25 = getelementptr inbounds %point, %point* %alice_cpk_result, i32 0
  %26 = load %point, %point* %25
  store %point %26, %point* %a_public_key
  %q20 = getelementptr inbounds %point, %point* %q, i32 0
  %pt_param21 = load %point, %point* %q20
  %sret_space22 = alloca %point
  %27 = getelementptr inbounds %point, %point* %sret_space22, i32 0
  %bob_cpk_result = call %point* @bob_cpk(%point* %27, %point %pt_param21)
  %28 = getelementptr inbounds %point, %point* %bob_cpk_result, i32 0
  %29 = load %point, %point* %28
  store %point %29, %point* %b_public_key
  %b_public_key23 = getelementptr inbounds %point, %point* %b_public_key, i32 0
  %pt_param24 = load %point, %point* %b_public_key23
  %alice_csecret_result = call i32 @alice_csecret(%point %pt_param24)
  %a_public_key25 = getelementptr inbounds %point, %point* %a_public_key, i32 0
  %pt_param26 = load %point, %point* %a_public_key25
  %bob_csecret_result = call i32 @bob_csecret(%point %pt_param26)
  ret i32 0
}

define i32 @bob_csecret(%point %a_public_key) {
entry:
  %a_public_key1 = alloca %point
  store %point %a_public_key, %point* %a_public_key1
  %beta = alloca %mpz_t
  %0 = alloca %mpz_t
  %1 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %__gmpz_init_set_str = call i32 @__gmpz_init_set_str(%mpz_t* %1, i8* getelementptr
    inbounds ([4 x i8], [4 x i8]* @string.10, i32 0, i32 0), i32 10)
  %2 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %beta2 = getelementptr inbounds %mpz_t, %mpz_t* %beta, i32 0
  %3 = call i32 @__gmpz_init_set(%mpz_t* %beta2, %mpz_t* %2)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
    @fmt.9, i32 0, i32 0), i8* getelementptr inbounds ([30 x i8], [30 x i8]* @string.11,
     i32 0, i32 0))
  %beta3 = getelementptr inbounds %mpz_t, %mpz_t* %beta, i32 0
  %a_public_key4 = getelementptr inbounds %point, %point* %a_public_key1, i32 0
  %pt_mul = call %point* @ptmul(%mpz_t* %beta3, %point* %a_public_key4)
  %printpt = call i32 @printpt(%point* %pt_mul)
  ret i32 0
}

define i32 @alice_csecret(%point %b_public_key) {
```

```
entry:
  %b_public_key1 = alloca %point
  store %point %b_public_key, %point* %b_public_key1
  %alpha = alloca %mpz_t
  %0 = alloca %mpz_t
  %1 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %__gmpz_init_set_str = call i32 @__gmpz_init_set_str(%mpz_t* %1, i8* getelementptr
    inbounds ([4 x i8], [4 x i8]* @string.14, i32 0, i32 0), i32 10)
  %2 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %alpha2 = getelementptr inbounds %mpz_t, %mpz_t* %alpha, i32 0
  %3 = call i32 @__gmpz_init_set(%mpz_t* %alpha2, %mpz_t* %2)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
    @fmt.13, i32 0, i32 0), i8* getelementptr inbounds ([32 x i8], [32 x i8]* @string
    .15, i32 0, i32 0))
  %alpha3 = getelementptr inbounds %mpz_t, %mpz_t* %alpha, i32 0
  %b_public_key4 = getelementptr inbounds %point, %point* %b_public_key1, i32 0
  %pt_mul = call %point* @ptmul(%mpz_t* %alpha3, %point* %b_public_key4)
  %printpt = call i32 @printpt(%point* %pt_mul)
  ret i32 0
}

define %point* @bob_cpk(%point* %sret, %point %q) {
entry:
  %sret1 = alloca %point*
  store %point* %sret, %point** %sret1
  %q2 = alloca %point
  store %point %q, %point* %q2
  %beta = alloca %mpz_t
  %0 = alloca %mpz_t
  %1 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %__gmpz_init_set_str = call i32 @__gmpz_init_set_str(%mpz_t* %1, i8* getelementptr
    inbounds ([4 x i8], [4 x i8]* @string.18, i32 0, i32 0), i32 10)
  %2 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %beta3 = getelementptr inbounds %mpz_t, %mpz_t* %beta, i32 0
  %3 = call i32 @__gmpz_init_set(%mpz_t* %beta3, %mpz_t* %2)
  %beta4 = getelementptr inbounds %mpz_t, %mpz_t* %beta, i32 0
  %q5 = getelementptr inbounds %point, %point* %q2, i32 0
  %pt_mul = call %point* @ptmul(%mpz_t* %beta4, %point* %q5)
  %4 = load %point, %point* %pt_mul
  %ret_ptr = load %point*, %point** %sret1
  store %point %4, %point* %ret_ptr
  ret %point* %ret_ptr
}

define %point* @alice_cpk(%point* %sret, %point %q) {
entry:
  %sret1 = alloca %point*
  store %point* %sret, %point** %sret1
  %q2 = alloca %point
  store %point %q, %point* %q2
  %alpha = alloca %mpz_t
  %0 = alloca %mpz_t
  %1 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %__gmpz_init_set_str = call i32 @__gmpz_init_set_str(%mpz_t* %1, i8* getelementptr
    inbounds ([4 x i8], [4 x i8]* @string.21, i32 0, i32 0), i32 10)
  %2 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %alpha3 = getelementptr inbounds %mpz_t, %mpz_t* %alpha, i32 0
  %3 = call i32 @__gmpz_init_set(%mpz_t* %alpha3, %mpz_t* %2)
  %alpha4 = getelementptr inbounds %mpz_t, %mpz_t* %alpha, i32 0
  %q5 = getelementptr inbounds %point, %point* %q2, i32 0
  %pt_mul = call %point* @ptmul(%mpz_t* %alpha4, %point* %q5)
  %4 = load %point, %point* %pt_mul
  %ret_ptr = load %point*, %point** %sret1
  store %point %4, %point* %ret_ptr
  ret %point* %ret_ptr
}
```

RSA (Concise Version)

```
lint encrypt(lint n, lint e){
     string plntxt; lint encotxt; lint ciphtxt;

     plntxt = "Hey Professor Edwards";
```

```
    encotxt = encode(plntxt);
    ciphtxt = encotxt ^ e @ n;
    return ciphtxt;
}
int main () { /* Primes taken from RSA Factor Challenge - RSA 250*/
  lint p; lint q; lint n; lint e; lint d; lint phi; lint max; lint ciphtxt; lint encotxt
    ;
  string mess;

  p =
    64135289477071580278790190170577389084825014742943447208116859632024532344630238623598752668347708
    l;
  q =
    33372027594978156556226010605355114227940760344767554666784520987023841729210037080257448673296881
    l;

  n = p * q;

  max = 10000000000000000000000000000000000000000001;
  e = random(10l, max);

  ciphtxt = encrypt(n,e);

  phi = (p-1l)*(q-1l);
  d = e`phi;

  encotxt = ciphtxt ^ d @ n;

  mess = decode(encotxt);
  prints(mess);

  return 0;
}
```

```
; ModuleID = 'Prime'
source_filename = "Prime"

%mpz_t = type { i32, i32, i64* }
%point = type { %mpz_t, %mpz_t, %poly }
%poly = type { %mpz_t, %mpz_t, %mpz_t }

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@string = private unnamed_addr constant [126 x i8] c
    "641352894770715802787901901705773890848250147429434472081168596320245323446302386235987526683477087

@string.2 = private unnamed_addr constant [126 x i8] c
    "333720275949781565562260106053551142279407603447675546667845209870238417292100370802574486732968818

@string.3 = private unnamed_addr constant [2 x i8] c"0\00"
@string.4 = private unnamed_addr constant [44 x i8] c
    "1000000000000000000000000000000000000000000\00"
@string.5 = private unnamed_addr constant [2 x i8] c"0\00"
@string.6 = private unnamed_addr constant [3 x i8] c"10\00"
@string.7 = private unnamed_addr constant [2 x i8] c"1\00"
@string.8 = private unnamed_addr constant [2 x i8] c"0\00"
@string.9 = private unnamed_addr constant [2 x i8] c"1\00"
@string.10 = private unnamed_addr constant [2 x i8] c"0\00"
@string.11 = private unnamed_addr constant [2 x i8] c"0\00"
@string.12 = private unnamed_addr constant [2 x i8] c"0\00"
@string.13 = private unnamed_addr constant [2 x i8] c"0\00"
@fmt.14 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.15 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@string.16 = private unnamed_addr constant [22 x i8] c"Hey Professor Edwards\00"
@string.17 = private unnamed_addr constant [2 x i8] c"0\00"

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

declare i32 @__gmpz_init_set_str(%mpz_t*, i8*, i32)

declare i32 @__gmpz_init_set_si(%mpz_t*, i32)

declare i32 @__gmpz_init_set(%mpz_t*, %mpz_t*)
```

```
declare i32 @printl(%mpz_t*)

declare i32 @__gmpz_add(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_sub(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_mul(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_tdiv_q(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_tdiv_r(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_pow_ui(%mpz_t*, %mpz_t*, i32)

declare i32 @__gmpz_invert(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_powm(%mpz_t*, %mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_neg(%mpz_t*, %mpz_t*)

declare i32 @lnot_func(%mpz_t*, %mpz_t*)

declare i32 @eq_func(%mpz_t*, %mpz_t*)

declare i32 @neq_func(%mpz_t*, %mpz_t*)

declare i32 @lth_func(%mpz_t*, %mpz_t*)

declare i32 @gth_func(%mpz_t*, %mpz_t*)

declare i32 @leq_func(%mpz_t*, %mpz_t*)

declare i32 @or_func(%mpz_t*, %mpz_t*)

declare i32 @and_func(%mpz_t*, %mpz_t*)

declare i32 @geq_func(%mpz_t*, %mpz_t*)

declare i32 @rand_func(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @Point(%point*, %mpz_t*, %mpz_t*, %poly*)

declare i32 @printpt(%point*)

declare %point* @ptadd(%point*, %point*)

declare %point* @ptmul(%mpz_t*, %point*)

declare %point* @ptneg(%point*)

declare i32 @pteq(%point*, %point*)

declare i32 @ptneq(%point*, %point*)

declare i32 @Poly(%poly*, %mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @printc(%poly*)

declare i32 @encode(%mpz_t*, i8*)

declare i8* @decode(%mpz_t*)

define i32 @main() {
entry:
  %p = alloca %mpz_t
  %q = alloca %mpz_t
  %n = alloca %mpz_t
  %e = alloca %mpz_t
  %d = alloca %mpz_t
  %phi = alloca %mpz_t
  %max = alloca %mpz_t
  %ciphtxt = alloca %mpz_t
```

```
%encotxt = alloca %mpz_t
%mess = alloca i8*
%0 = alloca %mpz_t
%1 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
%__gmpz_init_set_str = call i32 @__gmpz_init_set_str(%mpz_t* %1, i8* getelementptr
  inbounds ([126 x i8], [126 x i8]* @string, i32 0, i32 0), i32 10)
%2 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
%p1 = getelementptr inbounds %mpz_t, %mpz_t* %p, i32 0
%3 = call i32 @__gmpz_init_set(%mpz_t* %p1, %mpz_t* %2)
%4 = alloca %mpz_t
%5 = getelementptr inbounds %mpz_t, %mpz_t* %4, i32 0
%__gmpz_init_set_str2 = call i32 @__gmpz_init_set_str(%mpz_t* %5, i8* getelementptr
  inbounds ([126 x i8], [126 x i8]* @string.2, i32 0, i32 0), i32 10)
%6 = getelementptr inbounds %mpz_t, %mpz_t* %4, i32 0
%q3 = getelementptr inbounds %mpz_t, %mpz_t* %q, i32 0
%7 = call i32 @__gmpz_init_set(%mpz_t* %q3, %mpz_t* %6)
%p4 = getelementptr inbounds %mpz_t, %mpz_t* %p, i32 0
%q5 = getelementptr inbounds %mpz_t, %mpz_t* %q, i32 0
%8 = alloca %mpz_t
%9 = getelementptr inbounds %mpz_t, %mpz_t* %8, i32 0
%__gmpz_init_set_str6 = call i32 @__gmpz_init_set_str(%mpz_t* %9, i8* getelementptr
  inbounds ([2 x i8], [2 x i8]* @string.3, i32 0, i32 0), i32 10)
%10 = getelementptr inbounds %mpz_t, %mpz_t* %8, i32 0
%__gmpz_mul = call i32 @__gmpz_mul(%mpz_t* %10, %mpz_t* %p4, %mpz_t* %q5)
%n7 = getelementptr inbounds %mpz_t, %mpz_t* %n, i32 0
%11 = call i32 @__gmpz_init_set(%mpz_t* %n7, %mpz_t* %10)
%12 = alloca %mpz_t
%13 = getelementptr inbounds %mpz_t, %mpz_t* %12, i32 0
%__gmpz_init_set_str8 = call i32 @__gmpz_init_set_str(%mpz_t* %13, i8* getelementptr
  inbounds ([44 x i8], [44 x i8]* @string.4, i32 0, i32 0), i32 10)
%14 = getelementptr inbounds %mpz_t, %mpz_t* %12, i32 0
%max9 = getelementptr inbounds %mpz_t, %mpz_t* %max, i32 0
%15 = call i32 @__gmpz_init_set(%mpz_t* %max9, %mpz_t* %14)
%16 = alloca %mpz_t
%17 = getelementptr inbounds %mpz_t, %mpz_t* %16, i32 0
%__gmpz_init_set_str10 = call i32 @__gmpz_init_set_str(%mpz_t* %17, i8* getelementptr
  inbounds ([2 x i8], [2 x i8]* @string.5, i32 0, i32 0), i32 10)
%18 = getelementptr inbounds %mpz_t, %mpz_t* %16, i32 0
%19 = alloca %mpz_t
%20 = getelementptr inbounds %mpz_t, %mpz_t* %19, i32 0
%__gmpz_init_set_str11 = call i32 @__gmpz_init_set_str(%mpz_t* %20, i8* getelementptr
  inbounds ([3 x i8], [3 x i8]* @string.6, i32 0, i32 0), i32 10)
%21 = getelementptr inbounds %mpz_t, %mpz_t* %19, i32 0
%max12 = getelementptr inbounds %mpz_t, %mpz_t* %max, i32 0
%rand_func = call i32 @rand_func(%mpz_t* %18, %mpz_t* %21, %mpz_t* %max12)
%e13 = getelementptr inbounds %mpz_t, %mpz_t* %e, i32 0
%22 = call i32 @__gmpz_init_set(%mpz_t* %e13, %mpz_t* %18)
%e14 = getelementptr inbounds %mpz_t, %mpz_t* %e, i32 0
%lint_param = load %mpz_t, %mpz_t* %e14
%n15 = getelementptr inbounds %mpz_t, %mpz_t* %n, i32 0
%lint_param16 = load %mpz_t, %mpz_t* %n15
%sret_space = alloca %mpz_t
%23 = getelementptr inbounds %mpz_t, %mpz_t* %sret_space, i32 0
%encrypt_result = call %mpz_t* @encrypt(%mpz_t* %23, %mpz_t %lint_param16, %mpz_t %
  lint_param)
%ciphtxt17 = getelementptr inbounds %mpz_t, %mpz_t* %ciphtxt, i32 0
%24 = call i32 @__gmpz_init_set(%mpz_t* %ciphtxt17, %mpz_t* %encrypt_result)
%p18 = getelementptr inbounds %mpz_t, %mpz_t* %p, i32 0
%25 = alloca %mpz_t
%26 = getelementptr inbounds %mpz_t, %mpz_t* %25, i32 0
%__gmpz_init_set_str19 = call i32 @__gmpz_init_set_str(%mpz_t* %26, i8* getelementptr
  inbounds ([2 x i8], [2 x i8]* @string.7, i32 0, i32 0), i32 10)
%27 = getelementptr inbounds %mpz_t, %mpz_t* %25, i32 0
%28 = alloca %mpz_t
%29 = getelementptr inbounds %mpz_t, %mpz_t* %28, i32 0
%__gmpz_init_set_str20 = call i32 @__gmpz_init_set_str(%mpz_t* %29, i8* getelementptr
  inbounds ([2 x i8], [2 x i8]* @string.8, i32 0, i32 0), i32 10)
%30 = getelementptr inbounds %mpz_t, %mpz_t* %28, i32 0
%__gmpz_sub = call i32 @__gmpz_sub(%mpz_t* %30, %mpz_t* %p18, %mpz_t* %27)
%q21 = getelementptr inbounds %mpz_t, %mpz_t* %q, i32 0
%31 = alloca %mpz_t
%32 = getelementptr inbounds %mpz_t, %mpz_t* %31, i32 0
%__gmpz_init_set_str22 = call i32 @__gmpz_init_set_str(%mpz_t* %32, i8* getelementptr
```

```
    inbounds ([2 x i8], [2 x i8]* @string.9, i32 0, i32 0), i32 10)
  %33 = getelementptr inbounds %mpz_t, %mpz_t* %31, i32 0
  %34 = alloca %mpz_t
  %35 = getelementptr inbounds %mpz_t, %mpz_t* %34, i32 0
  %__gmpz_init_set_str23 = call i32 @__gmpz_init_set_str(%mpz_t* %35, i8* getelementptr
    inbounds ([2 x i8], [2 x i8]* @string.10, i32 0, i32 0), i32 10)
  %36 = getelementptr inbounds %mpz_t, %mpz_t* %34, i32 0
  %__gmpz_sub24 = call i32 @__gmpz_sub(%mpz_t* %36, %mpz_t* %q21, %mpz_t* %33)
  %37 = alloca %mpz_t
  %38 = getelementptr inbounds %mpz_t, %mpz_t* %37, i32 0
  %__gmpz_init_set_str25 = call i32 @__gmpz_init_set_str(%mpz_t* %38, i8* getelementptr
    inbounds ([2 x i8], [2 x i8]* @string.11, i32 0, i32 0), i32 10)
  %39 = getelementptr inbounds %mpz_t, %mpz_t* %37, i32 0
  %__gmpz_mul26 = call i32 @__gmpz_mul(%mpz_t* %39, %mpz_t* %30, %mpz_t* %36)
  %phi27 = getelementptr inbounds %mpz_t, %mpz_t* %phi, i32 0
  %40 = call i32 @__gmpz_init_set(%mpz_t* %phi27, %mpz_t* %39)
  %e28 = getelementptr inbounds %mpz_t, %mpz_t* %e, i32 0
  %phi29 = getelementptr inbounds %mpz_t, %mpz_t* %phi, i32 0
  %41 = alloca %mpz_t
  %42 = getelementptr inbounds %mpz_t, %mpz_t* %41, i32 0
  %__gmpz_init_set_str30 = call i32 @__gmpz_init_set_str(%mpz_t* %42, i8* getelementptr
    inbounds ([2 x i8], [2 x i8]* @string.12, i32 0, i32 0), i32 10)
  %43 = getelementptr inbounds %mpz_t, %mpz_t* %41, i32 0
  %__gmpz_invert = call i32 @__gmpz_invert(%mpz_t* %43, %mpz_t* %e28, %mpz_t* %phi29)
  %d31 = getelementptr inbounds %mpz_t, %mpz_t* %d, i32 0
  %44 = call i32 @__gmpz_init_set(%mpz_t* %d31, %mpz_t* %43)
  %ciphtxt32 = getelementptr inbounds %mpz_t, %mpz_t* %ciphtxt, i32 0
  %d33 = getelementptr inbounds %mpz_t, %mpz_t* %d, i32 0
  %n34 = getelementptr inbounds %mpz_t, %mpz_t* %n, i32 0
  %45 = alloca %mpz_t
  %46 = getelementptr inbounds %mpz_t, %mpz_t* %45, i32 0
  %__gmpz_init_set_str35 = call i32 @__gmpz_init_set_str(%mpz_t* %46, i8* getelementptr
    inbounds ([2 x i8], [2 x i8]* @string.13, i32 0, i32 0), i32 10)
  %47 = getelementptr inbounds %mpz_t, %mpz_t* %45, i32 0
  %__gmpz_powm = call i32 @__gmpz_powm(%mpz_t* %47, %mpz_t* %ciphtxt32, %mpz_t* %d33, %
    mpz_t* %n34)
  %encotxt36 = getelementptr inbounds %mpz_t, %mpz_t* %encotxt, i32 0
  %48 = call i32 @__gmpz_init_set(%mpz_t* %encotxt36, %mpz_t* %47)
  %encotxt37 = getelementptr inbounds %mpz_t, %mpz_t* %encotxt, i32 0
  %decode = call i8* @decode(%mpz_t* %encotxt37)
  store i8* %decode, i8** %mess
  %mess38 = load i8*, i8** %mess
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
    @fmt.1, i32 0, i32 0), i8* %mess38)
  ret i32 0
}

define %mpz_t* @encrypt(%mpz_t* %sret, %mpz_t %n, %mpz_t %e) {
entry:
  %sret1 = alloca %mpz_t*
  store %mpz_t* %sret, %mpz_t** %sret1
  %n2 = alloca %mpz_t
  store %mpz_t %n, %mpz_t* %n2
  %e3 = alloca %mpz_t
  store %mpz_t %e, %mpz_t* %e3
  %plntxt = alloca i8*
  %encotxt = alloca %mpz_t
  %ciphtxt = alloca %mpz_t
  store i8* getelementptr inbounds ([22 x i8], [22 x i8]* @string.16, i32 0, i32 0), i8
    ** %plntxt
  %plntxt4 = load i8*, i8** %plntxt
  %0 = alloca %mpz_t
  %1 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %encode = call i32 @encode(%mpz_t* %1, i8* %plntxt4)
  %encotxt5 = getelementptr inbounds %mpz_t, %mpz_t* %encotxt, i32 0
  %2 = call i32 @__gmpz_init_set(%mpz_t* %encotxt5, %mpz_t* %1)
  %encotxt6 = getelementptr inbounds %mpz_t, %mpz_t* %encotxt, i32 0
  %e7 = getelementptr inbounds %mpz_t, %mpz_t* %e3, i32 0
  %n8 = getelementptr inbounds %mpz_t, %mpz_t* %n2, i32 0
  %3 = alloca %mpz_t
  %4 = getelementptr inbounds %mpz_t, %mpz_t* %3, i32 0
  %__gmpz_init_set_str = call i32 @__gmpz_init_set_str(%mpz_t* %4, i8* getelementptr
    inbounds ([2 x i8], [2 x i8]* @string.17, i32 0, i32 0), i32 10)
```

```
  %5 = getelementptr inbounds %mpz_t, %mpz_t* %3, i32 0
  %__gmpz_powm = call i32 @__gmpz_powm(%mpz_t* %5, %mpz_t* %encotxt6, %mpz_t* %e7, %
    mpz_t* %n8)
  %ciphtxt9 = getelementptr inbounds %mpz_t, %mpz_t* %ciphtxt, i32 0
  %6 = call i32 @__gmpz_init_set(%mpz_t* %ciphtxt9, %mpz_t* %5)
  %ciphtxt10 = getelementptr inbounds %mpz_t, %mpz_t* %ciphtxt, i32 0
  %ret_ptr = load %mpz_t*, %mpz_t** %sret1
  %ret_set = call i32 @__gmpz_init_set(%mpz_t* %ret_ptr, %mpz_t* %ciphtxt10)
  ret %mpz_t* %ret_ptr
}
```

RSA

```
lint encrypt(lint n, lint e)
{
     string plntxt;
     lint encotxt;
     lint ciphtxt;

     plntxt = "Hey Professor Edwards";
     encotxt = encode(plntxt);
     prints(""); prints("Encoded text:");
     printl(encotxt);
     ciphtxt = encotxt ^ e @ n;
     prints(""); prints("Cipher text:");
     printl(ciphtxt);
     return ciphtxt;
}


int main ()
{

  /* RSA Algorithm Demonstration */
  /* Primes taken from RSA Factor Challenge - RSA 250*/

  /* declaration */
  lint p;
  lint q;
  lint n;
  lint e;
  lint d;
  lint phi;
  lint max;
  lint ciphtxt;
  lint decotxt;
  string mess;
  int cp;

  /* Select p and q, compute n = pq, and phi(n)  */

  p =
    641352894770715802787901901705773890848250147429434472081168596320245323446302386235987526683477087
    l;
  q =
    333720275949781565562260106053551142279407603447675546667845209870238417292100370802574486732968818
    l;

  n = p * q;
  phi = (p-1l)*(q-1l);

  /* randomly select e */
  max = 1000000000000000000000000000000000000000001l;
  e = random(10l, max);

  prints("Public key (n, e):");
  printl(n);
  prints(""); printl(e);
  /* n, e are the public key */
  ciphtxt = encrypt(n,e);

  /* find phi(n) and use that to find d */
  d = e`phi;
```

```
  /* decrypt message */
  decotxt = ciphtxt ^ d @ n;
  prints(""); prints("Decrypted enoded text:");
  printl(decotxt);


  /* decode message */
  prints(""); prints("Decoded message:");
  mess = decode(decotxt);
  prints(mess);

  return 0;
}
```

```
; ModuleID = 'Prime'
source_filename = "Prime"

%mpz_t = type { i32, i32, i64* }
%point = type { %mpz_t, %mpz_t, %poly }
%poly = type { %mpz_t, %mpz_t, %mpz_t }

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@string = private unnamed_addr constant [126 x i8] c
    "64135289477071580278790190170577389084825014742943447208116859632024532344630238623598752668347708

@string.2 = private unnamed_addr constant [126 x i8] c
    "33372027594978156556226010605355114227940760344767554666784520987023841729210037080257448673296881

@string.3 = private unnamed_addr constant [2 x i8] c"0\00"
@string.4 = private unnamed_addr constant [2 x i8] c"1\00"
@string.5 = private unnamed_addr constant [2 x i8] c"0\00"
@string.6 = private unnamed_addr constant [2 x i8] c"1\00"
@string.7 = private unnamed_addr constant [2 x i8] c"0\00"
@string.8 = private unnamed_addr constant [2 x i8] c"0\00"
@string.9 = private unnamed_addr constant [44 x i8] c
    "1000000000000000000000000000000000000000000\00"
@string.10 = private unnamed_addr constant [2 x i8] c"0\00"
@string.11 = private unnamed_addr constant [3 x i8] c"10\00"
@string.12 = private unnamed_addr constant [19 x i8] c"Public key (n, e):\00"
@string.13 = private unnamed_addr constant [1 x i8] zeroinitializer
@string.14 = private unnamed_addr constant [2 x i8] c"0\00"
@string.15 = private unnamed_addr constant [2 x i8] c"0\00"
@string.16 = private unnamed_addr constant [1 x i8] zeroinitializer
@string.17 = private unnamed_addr constant [23 x i8] c"Decrypted enoded text:\00"
@string.18 = private unnamed_addr constant [1 x i8] zeroinitializer
@string.19 = private unnamed_addr constant [17 x i8] c"Decoded message:\00"
@fmt.20 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.21 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@string.22 = private unnamed_addr constant [22 x i8] c"Hey Professor Edwards\00"
@string.23 = private unnamed_addr constant [1 x i8] zeroinitializer
@string.24 = private unnamed_addr constant [14 x i8] c"Encoded text:\00"
@string.25 = private unnamed_addr constant [2 x i8] c"0\00"
@string.26 = private unnamed_addr constant [1 x i8] zeroinitializer
@string.27 = private unnamed_addr constant [13 x i8] c"Cipher text:\00"

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

declare i32 @__gmpz_init_set_str(%mpz_t*, i8*, i32)

declare i32 @__gmpz_init_set_si(%mpz_t*, i32)

declare i32 @__gmpz_init_set(%mpz_t*, %mpz_t*)

declare i32 @printl(%mpz_t*)

declare i32 @__gmpz_add(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_sub(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_mul(%mpz_t*, %mpz_t*, %mpz_t*)

```

```
declare i32 @__gmpz_tdiv_q(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_tdiv_r(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_pow_ui(%mpz_t*, %mpz_t*, i32)

declare i32 @__gmpz_invert(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_powm(%mpz_t*, %mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @__gmpz_neg(%mpz_t*, %mpz_t*)

declare i32 @lnot_func(%mpz_t*, %mpz_t*)

declare i32 @eq_func(%mpz_t*, %mpz_t*)

declare i32 @neq_func(%mpz_t*, %mpz_t*)

declare i32 @lth_func(%mpz_t*, %mpz_t*)

declare i32 @gth_func(%mpz_t*, %mpz_t*)

declare i32 @leq_func(%mpz_t*, %mpz_t*)

declare i32 @or_func(%mpz_t*, %mpz_t*)

declare i32 @and_func(%mpz_t*, %mpz_t*)

declare i32 @geq_func(%mpz_t*, %mpz_t*)

declare i32 @rand_func(%mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @Point(%point*, %mpz_t*, %mpz_t*, %poly*)

declare i32 @printpt(%point*)

declare %point* @ptadd(%point*, %point*)

declare %point* @ptmul(%mpz_t*, %point*)

declare %point* @ptneg(%point*)

declare i32 @pteq(%point*, %point*)

declare i32 @ptneq(%point*, %point*)

declare i32 @Poly(%poly*, %mpz_t*, %mpz_t*, %mpz_t*)

declare i32 @printc(%poly*)

declare i32 @encode(%mpz_t*, i8*)

declare i8* @decode(%mpz_t*)

define i32 @main() {
entry:
  %p = alloca %mpz_t
  %q = alloca %mpz_t
  %n = alloca %mpz_t
  %e = alloca %mpz_t
  %d = alloca %mpz_t
  %phi = alloca %mpz_t
  %max = alloca %mpz_t
  %ciphtxt = alloca %mpz_t
  %decotxt = alloca %mpz_t
  %mess = alloca i8*
  %cp = alloca i32
  %0 = alloca %mpz_t
  %1 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %__gmpz_init_set_str = call i32 @__gmpz_init_set_str(%mpz_t* %1, i8* getelementptr
    inbounds ([126 x i8], [126 x i8]* @string, i32 0, i32 0), i32 10)
  %2 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %p1 = getelementptr inbounds %mpz_t, %mpz_t* %p, i32 0
```

```
  %3 = call i32 @__gmpz_init_set(%mpz_t* %p1, %mpz_t* %2)
  %4 = alloca %mpz_t
  %5 = getelementptr inbounds %mpz_t, %mpz_t* %4, i32 0
  %__gmpz_init_set_str2 = call i32 @__gmpz_init_set_str(%mpz_t* %5, i8* getelementptr
    inbounds ([126 x i8], [126 x i8]* @string.2, i32 0, i32 0), i32 10)
  %6 = getelementptr inbounds %mpz_t, %mpz_t* %4, i32 0
  %q3 = getelementptr inbounds %mpz_t, %mpz_t* %q, i32 0
  %7 = call i32 @__gmpz_init_set(%mpz_t* %q3, %mpz_t* %6)
  %p4 = getelementptr inbounds %mpz_t, %mpz_t* %p, i32 0
  %q5 = getelementptr inbounds %mpz_t, %mpz_t* %q, i32 0
  %8 = alloca %mpz_t
  %9 = getelementptr inbounds %mpz_t, %mpz_t* %8, i32 0
  %__gmpz_init_set_str6 = call i32 @__gmpz_init_set_str(%mpz_t* %9, i8* getelementptr
    inbounds ([2 x i8], [2 x i8]* @string.3, i32 0, i32 0), i32 10)
  %10 = getelementptr inbounds %mpz_t, %mpz_t* %8, i32 0
  %__gmpz_mul = call i32 @__gmpz_mul(%mpz_t* %10, %mpz_t* %p4, %mpz_t* %q5)
  %n7 = getelementptr inbounds %mpz_t, %mpz_t* %n, i32 0
  %11 = call i32 @__gmpz_init_set(%mpz_t* %n7, %mpz_t* %10)
  %p8 = getelementptr inbounds %mpz_t, %mpz_t* %p, i32 0
  %12 = alloca %mpz_t
  %13 = getelementptr inbounds %mpz_t, %mpz_t* %12, i32 0
  %__gmpz_init_set_str9 = call i32 @__gmpz_init_set_str(%mpz_t* %13, i8* getelementptr
    inbounds ([2 x i8], [2 x i8]* @string.4, i32 0, i32 0), i32 10)
  %14 = getelementptr inbounds %mpz_t, %mpz_t* %12, i32 0
  %15 = alloca %mpz_t
  %16 = getelementptr inbounds %mpz_t, %mpz_t* %15, i32 0
  %__gmpz_init_set_str10 = call i32 @__gmpz_init_set_str(%mpz_t* %16, i8* getelementptr
    inbounds ([2 x i8], [2 x i8]* @string.5, i32 0, i32 0), i32 10)
  %17 = getelementptr inbounds %mpz_t, %mpz_t* %15, i32 0
  %__gmpz_sub = call i32 @__gmpz_sub(%mpz_t* %17, %mpz_t* %p8, %mpz_t* %14)
  %q11 = getelementptr inbounds %mpz_t, %mpz_t* %q, i32 0
  %18 = alloca %mpz_t
  %19 = getelementptr inbounds %mpz_t, %mpz_t* %18, i32 0
  %__gmpz_init_set_str12 = call i32 @__gmpz_init_set_str(%mpz_t* %19, i8* getelementptr
    inbounds ([2 x i8], [2 x i8]* @string.6, i32 0, i32 0), i32 10)
  %20 = getelementptr inbounds %mpz_t, %mpz_t* %18, i32 0
  %21 = alloca %mpz_t
  %22 = getelementptr inbounds %mpz_t, %mpz_t* %21, i32 0
  %__gmpz_init_set_str13 = call i32 @__gmpz_init_set_str(%mpz_t* %22, i8* getelementptr
    inbounds ([2 x i8], [2 x i8]* @string.7, i32 0, i32 0), i32 10)
  %23 = getelementptr inbounds %mpz_t, %mpz_t* %21, i32 0
  %__gmpz_sub14 = call i32 @__gmpz_sub(%mpz_t* %23, %mpz_t* %q11, %mpz_t* %20)
  %24 = alloca %mpz_t
  %25 = getelementptr inbounds %mpz_t, %mpz_t* %24, i32 0
  %__gmpz_init_set_str15 = call i32 @__gmpz_init_set_str(%mpz_t* %25, i8* getelementptr
    inbounds ([2 x i8], [2 x i8]* @string.8, i32 0, i32 0), i32 10)
  %26 = getelementptr inbounds %mpz_t, %mpz_t* %24, i32 0
  %__gmpz_mul16 = call i32 @__gmpz_mul(%mpz_t* %26, %mpz_t* %17, %mpz_t* %23)
  %phi17 = getelementptr inbounds %mpz_t, %mpz_t* %phi, i32 0
  %27 = call i32 @__gmpz_init_set(%mpz_t* %phi17, %mpz_t* %26)
  %28 = alloca %mpz_t
  %29 = getelementptr inbounds %mpz_t, %mpz_t* %28, i32 0
  %__gmpz_init_set_str18 = call i32 @__gmpz_init_set_str(%mpz_t* %29, i8* getelementptr
    inbounds ([44 x i8], [44 x i8]* @string.9, i32 0, i32 0), i32 10)
  %30 = getelementptr inbounds %mpz_t, %mpz_t* %28, i32 0
  %max19 = getelementptr inbounds %mpz_t, %mpz_t* %max, i32 0
  %31 = call i32 @__gmpz_init_set(%mpz_t* %max19, %mpz_t* %30)
  %32 = alloca %mpz_t
  %33 = getelementptr inbounds %mpz_t, %mpz_t* %32, i32 0
  %__gmpz_init_set_str20 = call i32 @__gmpz_init_set_str(%mpz_t* %33, i8* getelementptr
    inbounds ([2 x i8], [2 x i8]* @string.10, i32 0, i32 0), i32 10)
  %34 = getelementptr inbounds %mpz_t, %mpz_t* %32, i32 0
  %35 = alloca %mpz_t
  %36 = getelementptr inbounds %mpz_t, %mpz_t* %35, i32 0
  %__gmpz_init_set_str21 = call i32 @__gmpz_init_set_str(%mpz_t* %36, i8* getelementptr
    inbounds ([3 x i8], [3 x i8]* @string.11, i32 0, i32 0), i32 10)
  %37 = getelementptr inbounds %mpz_t, %mpz_t* %35, i32 0
  %max22 = getelementptr inbounds %mpz_t, %mpz_t* %max, i32 0
  %rand_func = call i32 @rand_func(%mpz_t* %34, %mpz_t* %37, %mpz_t* %max22)
  %e23 = getelementptr inbounds %mpz_t, %mpz_t* %e, i32 0
  %38 = call i32 @__gmpz_init_set(%mpz_t* %e23, %mpz_t* %34)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
    @fmt.1, i32 0, i32 0), i8* getelementptr inbounds ([19 x i8], [19 x i8]* @string.12,
```

```
    i32 0, i32 0))
  %39 = getelementptr inbounds %mpz_t, %mpz_t* %n, i32 0
  %printl = call i32 @printl(%mpz_t* %39)
  %printf24 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1, i32 0, i32 0), i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.13, i32 0, i32 0))
  %40 = getelementptr inbounds %mpz_t, %mpz_t* %e, i32 0
  %printl25 = call i32 @printl(%mpz_t* %40)
  %e26 = getelementptr inbounds %mpz_t, %mpz_t* %e, i32 0
  %lint_param = load %mpz_t, %mpz_t* %e26
  %n27 = getelementptr inbounds %mpz_t, %mpz_t* %n, i32 0
  %lint_param28 = load %mpz_t, %mpz_t* %n27
  %sret_space = alloca %mpz_t
  %41 = getelementptr inbounds %mpz_t, %mpz_t* %sret_space, i32 0
  %encrypt_result = call %mpz_t* @encrypt(%mpz_t* %41, %mpz_t %lint_param28, %mpz_t %lint_param)
  %ciphtxt29 = getelementptr inbounds %mpz_t, %mpz_t* %ciphtxt, i32 0
  %42 = call i32 @__gmpz_init_set(%mpz_t* %ciphtxt29, %mpz_t* %encrypt_result)
  %e30 = getelementptr inbounds %mpz_t, %mpz_t* %e, i32 0
  %phi31 = getelementptr inbounds %mpz_t, %mpz_t* %phi, i32 0
  %43 = alloca %mpz_t
  %44 = getelementptr inbounds %mpz_t, %mpz_t* %43, i32 0
  %__gmpz_init_set_str32 = call i32 @__gmpz_init_set_str(%mpz_t* %44, i8* getelementptr inbounds ([2 x i8], [2 x i8]* @string.14, i32 0, i32 0), i32 10)
  %45 = getelementptr inbounds %mpz_t, %mpz_t* %43, i32 0
  %__gmpz_invert = call i32 @__gmpz_invert(%mpz_t* %45, %mpz_t* %e30, %mpz_t* %phi31)
  %d33 = getelementptr inbounds %mpz_t, %mpz_t* %d, i32 0
  %46 = call i32 @__gmpz_init_set(%mpz_t* %d33, %mpz_t* %45)
  %ciphtxt34 = getelementptr inbounds %mpz_t, %mpz_t* %ciphtxt, i32 0
  %d35 = getelementptr inbounds %mpz_t, %mpz_t* %d, i32 0
  %n36 = getelementptr inbounds %mpz_t, %mpz_t* %n, i32 0
  %47 = alloca %mpz_t
  %48 = getelementptr inbounds %mpz_t, %mpz_t* %47, i32 0
  %__gmpz_init_set_str37 = call i32 @__gmpz_init_set_str(%mpz_t* %48, i8* getelementptr inbounds ([2 x i8], [2 x i8]* @string.15, i32 0, i32 0), i32 10)
  %49 = getelementptr inbounds %mpz_t, %mpz_t* %47, i32 0
  %__gmpz_powm = call i32 @__gmpz_powm(%mpz_t* %49, %mpz_t* %ciphtxt34, %mpz_t* %d35, %mpz_t* %n36)
  %decotxt38 = getelementptr inbounds %mpz_t, %mpz_t* %decotxt, i32 0
  %50 = call i32 @__gmpz_init_set(%mpz_t* %decotxt38, %mpz_t* %49)
  %printf39 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1, i32 0, i32 0), i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.16, i32 0, i32 0))
  %printf40 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1, i32 0, i32 0), i8* getelementptr inbounds ([23 x i8], [23 x i8]* @string.17, i32 0, i32 0))
  %51 = getelementptr inbounds %mpz_t, %mpz_t* %decotxt, i32 0
  %printl41 = call i32 @printl(%mpz_t* %51)
  %printf42 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1, i32 0, i32 0), i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.18, i32 0, i32 0))
  %printf43 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1, i32 0, i32 0), i8* getelementptr inbounds ([17 x i8], [17 x i8]* @string.19, i32 0, i32 0))
  %decotxt44 = getelementptr inbounds %mpz_t, %mpz_t* %decotxt, i32 0
  %decode = call i8* @decode(%mpz_t* %decotxt44)
  store i8* %decode, i8** %mess
  %mess45 = load i8*, i8** %mess
  %printf46 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1, i32 0, i32 0), i8* %mess45)
  ret i32 0
}

define %mpz_t* @encrypt(%mpz_t* %sret, %mpz_t %n, %mpz_t %e) {
entry:
  %sret1 = alloca %mpz_t*
  store %mpz_t* %sret, %mpz_t** %sret1
  %n2 = alloca %mpz_t
  store %mpz_t %n, %mpz_t* %n2
  %e3 = alloca %mpz_t
  store %mpz_t %e, %mpz_t* %e3
  %plntxt = alloca i8*
  %encotxt = alloca %mpz_t
```

```
  %ciphtxt = alloca %mpz_t
  store i8* getelementptr inbounds ([22 x i8], [22 x i8]* @string.22, i32 0, i32 0), i8
    ** %plntxt
  %plntxt4 = load i8*, i8** %plntxt
  %0 = alloca %mpz_t
  %1 = getelementptr inbounds %mpz_t, %mpz_t* %0, i32 0
  %encode = call i32 @encode(%mpz_t* %1, i8* %plntxt4)
  %encotxt5 = getelementptr inbounds %mpz_t, %mpz_t* %encotxt, i32 0
  %2 = call i32 @__gmpz_init_set(%mpz_t* %encotxt5, %mpz_t* %1)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
    @fmt.21, i32 0, i32 0), i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.23,
    i32 0, i32 0))
  %printf6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
     @fmt.21, i32 0, i32 0), i8* getelementptr inbounds ([14 x i8], [14 x i8]* @string
    .24, i32 0, i32 0))
  %3 = getelementptr inbounds %mpz_t, %mpz_t* %encotxt, i32 0
  %printl = call i32 @printl(%mpz_t* %3)
  %encotxt7 = getelementptr inbounds %mpz_t, %mpz_t* %encotxt, i32 0
  %e8 = getelementptr inbounds %mpz_t, %mpz_t* %e3, i32 0
  %n9 = getelementptr inbounds %mpz_t, %mpz_t* %n2, i32 0
  %4 = alloca %mpz_t
  %5 = getelementptr inbounds %mpz_t, %mpz_t* %4, i32 0
  %__gmpz_init_set_str = call i32 @__gmpz_init_set_str(%mpz_t* %5, i8* getelementptr
    inbounds ([2 x i8], [2 x i8]* @string.25, i32 0, i32 0), i32 10)
  %6 = getelementptr inbounds %mpz_t, %mpz_t* %4, i32 0
  %__gmpz_powm = call i32 @__gmpz_powm(%mpz_t* %6, %mpz_t* %encotxt7, %mpz_t* %e8, %
    mpz_t* %n9)
  %ciphtxt10 = getelementptr inbounds %mpz_t, %mpz_t* %ciphtxt, i32 0
  %7 = call i32 @__gmpz_init_set(%mpz_t* %ciphtxt10, %mpz_t* %6)
  %printf11 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8
    ]* @fmt.21, i32 0, i32 0), i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string
    .26, i32 0, i32 0))
  %printf12 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8
    ]* @fmt.21, i32 0, i32 0), i8* getelementptr inbounds ([13 x i8], [13 x i8]* @string
    .27, i32 0, i32 0))
  %8 = getelementptr inbounds %mpz_t, %mpz_t* %ciphtxt, i32 0
  %printl13 = call i32 @printl(%mpz_t* %8)
  %ciphtxt14 = getelementptr inbounds %mpz_t, %mpz_t* %ciphtxt, i32 0
  %ret_ptr = load %mpz_t*, %mpz_t** %sret1
  %ret_set = call i32 @__gmpz_init_set(%mpz_t* %ret_ptr, %mpz_t* %ciphtxt14)
  ret %mpz_t* %ret_ptr
}
```

8 Lessons Learned

8.1 Team Advice

For future teams, make sure you check in often with your team to get the ball rolling and keep up some momentum. Go to OH as many times as necessary, particularly when getting started so you can be pointed in the right direction, this was an early sticking point for us.

8.2 Alexander Liebeskind

This project was quite demanding, but it's incredibly fulfilling to see it all come together. I learned a lot from PRIME about everything from functional programming to group collaboration.

One of my most significant takeaways from the project is the importance of designing around the user. Some of the ideas we had at the outset made sense from a mathematical perspective, but weren't the most effective way to make the life of the programmer easy. For example, we had initially conceived of rings as their own data type, but incorporating rings into curves facilitated elliptic curve cryptography. Realizing an initial proposal is not final and being willing to rethink ideas during implementation is crucial.

Also, communication with the group is vital. Every part of a language relies on the others, so making sure everyone is on the same page is essential.

8.3 Nikhil Mehta

It was a rough semester and I definitely came to realize the importance of getting enough sleep.

Writing robust test cases for every feature in your language as soon as they're implemented is vital to success. If you implement a feature without testing it in 3 or 5 or 10 different ways it might was well be broken. When we didn't test features as well as we could have, we would later find out that putting in literals would throw weird behavior or that certain expressions would cause a segmentation fault.

In addition, I learned, over time, to appreciate OCaml and functional programming. This required a very different kind of thinking than what has been required in all my prior coursework. Through this project I have come to appreciate functional programming a lot more than I did after the first homework.

Lastly, it's essential to write down and constantly update both your individual tasks and the group's tasks. The team was always handling multiple different tasks across different branches at any given moment over the course of the semester and it became very easy for things to slip through the cracks.

8.4 Thomas Tran

LLVM documentation is awful, so understanding how to read it from the start is crucial. Placing an emphasis on understanding the end to end from scanner to executable is especially important, and developing features end to end helped me to understand what was going on low level. Also, using git best practices and setting up continuous integration helps manage version history for large projects like this. (shoutout pedro)

8.5 Pedro B T Santos

This semester's been difficult. Made more so by the almost intractable nature of LLVM and its sparse documentation.

However, I did learn a lot from experimenting and seeing the project build and take shape was very rewarding. I'd always wanted to learn more functional programming, and was glad to be able to do so with OCaml. I learned the importance of planning and communication when managing a remote work environment. This semester especially when none of us were able to meet one another it was especially important to check in every so often and to remain efficient in what we did.

9 Appendix

Attach a complete code listing of your translator with each module signed by its author
All modules were written and edited by all team members. MicroC was used as a starting point for OCaml files.

9.1 ast.ml

```
(* Create a new operator for assignment and create a new expression*)
(* sequences of expressions *)

type operator = Add | Sub | Mul | Div | Mod | Pow | Beq | Bneq | Leq | Geq | Lth | Gth |
     And | Or | Inv
type eqsign = Eq
type uoperator = Neg | Not
type accessor = Access
type toperator = Lpw | Pmd

type typ = Int | Lint | Chr | Ring | String | Point | Poly | Void
type bind = typ * string

type expr =
    Strlit of string
  | Lit of int
  | Lintlit of string
  | Ptlit of expr * expr * expr
  | Polylit of expr * expr * expr
  | Id of string
  | Binop of expr * operator * expr
```

```
  | Relop of expr * operator * expr
  | Trnop of expr * toperator * expr * toperator * expr
  | Unop of uoperator * expr
  | Assign of string * expr
  | Access of string * string (* we will use the second string to convert to gep*)
  | Call of string * expr list
  | Noexpr

type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt (*need for pretty print, temp*)
  | For of expr * expr * expr * stmt
  | While of expr * stmt

type func_decl = {
  typ : typ;
  name : string;
  params : bind list;
  locals : bind list;
  body : stmt list;
}


(* Essentially means variable declarations followed by function defs *)
type program = bind list * func_decl list


let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mul -> "*"
  | Div -> "/"
  | Mod -> "%"
  | Pow -> "/\\"
  | Inv -> "`"
  | Beq -> "=="
  | Bneq -> "!="
  | Leq -> "<="
  | Geq -> ">="
  | Lth -> "<"
  | Gth -> ">"
  | And -> "&&"
  | Or -> "||"

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

let string_of_top = function
    Lpw -> "^"
  | Pmd -> "@"

let rec string_of_expr = function
    Strlit(l) -> "\"" ^ l ^ "\""
  | Id(s)   -> s
  | Lit(l) -> string_of_int l
  | Lintlit(l) -> l
  | Ptlit(i, j, p) -> "[" ^ string_of_expr i ^ "," ^ string_of_expr j ^ "] & "^ string_of_expr p
  | Polylit(i,j, m) -> "[(" ^ string_of_expr i ^ "," ^ string_of_expr j^ ") : " ^ string_of_expr m ^ "]"
  | Binop(e1, o, e2) ->
        string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Relop(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
  | Trnop(e1, o1, e2, o2, e3) ->
        string_of_expr e1 ^ " " ^ string_of_top o1 ^ " " ^ string_of_expr e2 ^ " " ^
        string_of_top o2 ^ " " ^  string_of_expr e3
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Access(v, s) -> v ^ "." ^ s
```

```
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""

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

let string_of_typ = function
    Int -> "int"
  | String -> "string"
  | Lint -> "lint"
  | Point -> "Point"
  | Poly -> "poly"
  | Void -> "void"
  | _ -> "typ PP not implemented"


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

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.name ^ "(" ^ String.concat ", " (List.map snd fdecl.params) ^
  ")\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)
```

9.2 sast.ml

```
open Ast

(* The key thing for the semantic checked ast *)
type sexpr = typ * sx
and sx =
    SLit of int
  | SStrlit of string
  | SLintlit of string
  | SPtlit of sexpr * sexpr * sexpr
  | SPolylit of sexpr * sexpr * sexpr
  | SAccess of string * int
  | SId of string
  | SBinop of sexpr * operator * sexpr
  | SRelop of sexpr * operator * sexpr
  | SUnop of uoperator * sexpr
  | STrnop of sexpr * toperator * sexpr * toperator * sexpr
  | SAssign of string * sexpr
  | SCall of string * sexpr list
  | SNoexpr

type sstmt =
    SBlock of sstmt list
  | SExpr of sexpr
  | SReturn of sexpr
  | SIf of sexpr * sstmt * sstmt (*just for pretty print for now*)
  | SFor of sexpr * sexpr * sexpr * sstmt
  | SWhile of sexpr * sstmt

type sfunc_decl = {
```

```
    styp : typ;
    sname : string;
    sparams: bind list;
    slocals : bind list;
    sbody : sstmt list;
}

type sprogram = bind list * sfunc_decl list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SStrlit(l) -> "\"" ^ l ^ "\""
  | SLintlit(l) -> l
  | SPtlit(i, j, p) -> "[" ^ string_of_sexpr i ^ "," ^ string_of_sexpr j ^ "] & " ^ string_of_sexpr p
  | SPolylit(i, j, m) -> "[(" ^ string_of_sexpr i ^ "," ^ string_of_sexpr j^ ") : " ^ string_of_sexpr m ^ "]"
  | SAccess(s, i) ->s ^ "." ^ string_of_int i
  | SLit(l) -> string_of_int l
  | SId(s) -> s
  | SBinop(e1, o, e2) ->
        string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
  | SRelop(e1, o, e2) ->
        string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | STrnop(e1, o1, e2, o2, e3) ->
        string_of_sexpr e1 ^ " " ^ string_of_top o1 ^ " " ^ string_of_sexpr e2 ^ " " ^
          string_of_top o2 ^ " " ^ string_of_sexpr e3
  | SCall(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
  | SNoexpr -> ""
        ) ^ ")"

let rec string_of_sstmt = function
    SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
  | SIf(e, s, SBlock([])) ->
      "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
      string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
  | SFor(e1, e2, e3, s) ->
      "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
      string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s

let string_of_sfdecl fdecl =
  string_of_typ fdecl.styp ^ " " ^
  fdecl.sname ^ "(" ^ String.concat ", " (List.map snd fdecl.sparams) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.slocals) ^
  String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
  "}\n"

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

9.3 parser.mly

```
%{ open Ast %}
// Thank you again to Professor Edwards for the MicroC template.
// We have made alterations and additions for our language's functionality

%token SEMI COLON LPAREN RPAREN LBRACE RBRACE RBRACK LBRACK COMMA AMP
%token PLUS MINUS TIMES DIVIDE MOD POWER ASSIGN INVERT
%token PMOD LPOWER
%token BEQ BNEQ LTH GTH GEQ LEQ AND OR NOT
```

```
%token ACCESS
%token RETURN IF ELSE WHILE FOR INT LINT POLY POINT RING CHAR STRING //(*add float/void here if wanted*)
%token <int> LITERAL
%token <string> STRLIT LINTLIT ID
%token EOF

%start program
%type <Ast.program> program //(* Add in later when we define the AST *)

// (*precedence*)
%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left BEQ BNEQ
%left LTH GTH LEQ GEQ
%left MOD //(* mod takes precedence below all arithmetic operators - l.guru approves*)
%left PLUS MINUS
%left TIMES DIVIDE //(* Change this order later if necessary \r moved mod up -l.guru*)
%right INVERT
%right NOT
%right POWER
%nonassoc AMP
%right PMOD
%left LPOWER
%left ACCESS    // Built in access methods

%%

//(* All the semantic action braces will be empty for this submission *)

program:
  decls EOF { $1 }

// fst gets the var decs, snd gets the function decs
decls:
   /* nothing */ { ([], []) } // Building up list of variable decs and list of function decs
  | decls declare_init { (($2 :: fst $1), snd $1) }  //(* No external variables ? or keep as is*)
  | decls fdecl { (fst $1, ($2 :: snd $1)) }

// create the record denoted by AST
// the body will then contain declarations (allowed by expr)
fdecl:
   typ ID LPAREN params_opt RPAREN LBRACE seq_stmts RBRACE
     { { typ = $1;
         name = $2;
         params = List.rev $4;
         locals = List.rev (fst $7);
         body = List.rev (snd $7) (* Might have to split this for hello world *)
        } }

params_opt:
    /* nothing */ { [] }
  | params_list   { $1 }

// Have the lists on the left
params_list:
    typ ID                     { [($1,$2)] }
  | params_list COMMA typ ID { ($3,$4) :: $1 }


typ:
    INT   { Int }
  | LINT  { Lint }
  | POINT { Point }
  | POLY { Poly }
  | STRING { String }

vars:
```

```
  /* nothing */ { [] }
  | vars declare_init  { $2 :: $1 }

declare_init:
  typ declarator SEMI { ($1, $2) }

declarator:
    ID { $1 }

seq_stmts:
    vars stmt_list { ($1, $2) }

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

stmt:
    expr_opt SEMI                           { Expr $1 } //(* Expr-stmt *)
  | RETURN expr_opt SEMI                    { Return $2 } //(* Return stmt *)
  | LBRACE stmt_list RBRACE                 { Block(List.rev $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 SEMI expr SEMI expr_opt RPAREN stmt
                                            { For($3, $5, $7, $9)   }
  | WHILE LPAREN expr RPAREN stmt           { While($3, $5)         }

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

expr:
    ID                { Id($1) }
  | LITERAL           { Lit($1) }
  | STRLIT            { Strlit($1) }
  | LINTLIT           { Lintlit($1) }
  | LBRACK expr COMMA expr RBRACK AMP expr { Ptlit ($2, $4, $7) }
  | LBRACK LPAREN expr COMMA expr RPAREN COLON expr RBRACK {Polylit($3, $5, $8)}
  | expr MOD    expr { Binop($1, Mod, $3) }
  | expr POWER  expr { Binop($1, Pow, $3) }
  | expr PLUS   expr { Binop($1, Add, $3) }
  | expr MINUS  expr { Binop($1, Sub, $3) }
  | expr TIMES  expr { Binop($1, Mul, $3) }
  | expr DIVIDE expr { Binop($1, Div, $3) }
  | expr INVERT expr { Binop($1, Inv, $3) }
  | expr BEQ    expr { Relop($1, Beq, $3) }
  | expr BNEQ   expr { Relop($1, Bneq, $3)}
  | expr LTH    expr { Relop($1, Lth, $3) }
  | expr LEQ    expr { Relop($1, Leq, $3) }
  | expr GTH    expr { Relop($1, Gth, $3) }
  | expr GEQ    expr { Relop($1, Geq, $3) }
  | expr AND    expr { Relop($1, And, $3) }
  | expr OR     expr { Relop($1, Or, $3)  }
  | expr LPOWER   expr PMOD expr { Trnop($1, Lpw, $3, Pmd, $5) }
  | MINUS expr %prec NOT { Unop(Neg, $2)  }
  | NOT expr          { Unop(Not, $2)      }
  | ID ASSIGN expr    { Assign($1, $3)     }
  | ID ACCESS ID      { Access($1, $3) } // will be used for accessor methods
  | ID LPAREN args_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN {  $2  }

args_opt:
    /* nothing */ { [] }
  | args_list  { List.rev $1 }

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

9.4 scanner.mll

```
(* Ocamllex scanner for PRIME
   Many thanks to the MicroC compiler example created by
```

```
   Professor Edwards
   Many of the symbols here are directly from or follow that.
*)

{
  open Parser
}

let digit = ['0' - '9']
let digits = digit+

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf }
| "/*"                  { comment lexbuf }  (* add comments *)
| '('       { LPAREN } (* Grouping operators *)
| ')'       { RPAREN }
| '{'       { LBRACE }
| '}'       { RBRACE }
| '['       { LBRACK }
| ']'       { RBRACK }
| ','       { COMMA  }
| '='       { ASSIGN } (* Binary Operators (semi perhaps not) *)
| ';'       { SEMI }
| ':'       { COLON }
| '+'       { PLUS }
| '-'       { MINUS }
| '*'       { TIMES }
| '/'       { DIVIDE }
| "/\\"     { POWER  }
| '%'       { MOD    }
| '`'       { INVERT }
| '.'       { ACCESS }
| '&'       { AMP }
| '@'       { PMOD   }
| '^'       { LPOWER }
| "=="      { BEQ    } (* Relational Ops (which ones of these do we want?)*)
| "!="      { BNEQ   }
| '<'       { LTH    }
| "<="      { LEQ    }
| ">"       { GTH    }
| ">="      { GEQ    }
| "&&"      { AND    }
| "||"      { OR     }
| "!"       { NOT    }
| "if"      { IF     }   (* Keywords and types *)
| "else"    { ELSE   }
| "for"     { FOR    }
| "while"   { WHILE  }
| "return"  { RETURN }
| "int"     { INT    }
| "char"    { CHAR   }
| "string"  { STRING }
| "lint"    { LINT   }  (* OUR CUSTOM TYPES *)
| "curve"     { POLY    } (*More needs to be done here*)
| "pt"      { POINT  }
| "ring"    { RING   }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as name { ID(name) } (*ids can be alpha followed by alphanum and _*)
| digits as lit { LITERAL(int_of_string lit) }
| (digits as lit)('l')  { LINTLIT(lit) }
| '"'(([' '-'!' '#'-'[' ']'-'~'])* as lit)'"' { STRLIT(lit) }
(*| '"'(_* as lit)'"' { STRLIT(lit) } Make a separate rule for looking through string literals and comment literals *)
| eof       { EOF }
| _  as char      { raise (Failure("Undefined character " ^ Char.escaped char)) } (* any other character is not allowed *)

(* part of rule for ending comments *)
and comment = parse
  "*/" { token lexbuf } (*back to normal scanning *)
| _    { comment lexbuf } (* keep reading comments *)

{
```

```
}
```

9.5 semant.ml

```
(* Semantic checking file *)

open Ast
open Sast

(* Make a map to keep track of globals *)
module StringMap = Map.Make(String)

(* String hashmap for lint string conversion *)
(* e.g. RSA *)

module StringHash = Hashtbl.Make(struct
  type t = string
  let equal x y = x = y
  let hash = Hashtbl.hash
  end);;

let vals : int StringHash.t = StringHash.create 10;;


(* Begin Semantic checking sast if good else error *)

let check (globals, functions) =
  (* Check binds have types and ids are unique *)
  let check_binds (kind : string) (binds : bind list) =
    List.iter (function
    (Void, b) -> raise (Failure ("missing/wrong type in declaration " ^ kind ^ " " ^ b))
      | _ -> ()) binds;
    let rec dups = function
        [] -> () (* No name found here *)
      | ((_, n1) :: (_, n2) :: _) when n1 = n2 -> (* check if same in order because sorted *)
        raise (Failure ("duplicate " ^ " " ^ n1))
      | _ :: t -> dups t (* check the tail of the binds *)
  in dups (List.sort (fun (_,a) (_,b) -> compare a b) binds)
in

(* Now actually perform the checks first variables then functions *)
check_binds "global" globals;

(* Start with function declarations for built-ins (just print for now)*)
(* Just call the formal parameter ID of our inbuit functions x*)
let built_in_decls =
  let add_bind map (name, ty) = StringMap.add name {
    typ = Void; (* Our built in print functions will return string*)
    name = name;
    params = [(ty, "x")];
    locals = []; body = [] (* In-built don't have body. Determine semantics here *)
  } map
  in let void_decls = List.fold_left add_bind StringMap.empty [ ("print", Int);
                                                ("prints", String);
                                                ("printl", Lint);
                                                ("printpt", Point);
                                                ("printc", Poly);]
     and add_cast map (name, ty) = StringMap.add name {
       typ = Lint;
       name = name;
       params = [(ty, "x")];
       locals = []; body = []
     } map
  in let void_decls = List.fold_left add_cast void_decls [ ("tolint", Int) ]
     and add_rand map (name, ty) = StringMap.add name {
         typ = Lint;
         name = name;
         params = [(ty, ("x")); (ty, ("y"))];
         locals = []; body = []
      } map
```

```
  in let void_decls = List.fold_left add_rand void_decls [ ("random", Lint) ]
     and add_decode map (name, ty) = StringMap.add name {
       typ = String;
       name = name;
       params = [(ty, "x")];
       locals = []; body = []
     } map
     and add_encode map (name, _) = StringMap.add name {
       typ = Lint;
       name = name;
       params = [(*(Lint, "x");*) (String, "y")]; (* Don't necessarily have to hard-code this but time is short *)
       locals = []; body = [];
     } map
  in let built_decls = List.fold_left add_decode void_decls [ ("decode", Lint)]
  in List.fold_left add_encode built_decls [ ("encode", (String)) ]
  (* We likely don't need the GMP functions here because they are not called directly (in fact should not be) *)
in

(* Now keep track of these named built-in funcs in the top-level symbol table *)
let add_func map fd =
  (* Define what errors we might have  *)
  let built_in_err = "function " ^ fd.name ^ " not defined"
  and dup_err = "duplicate function found: " ^ fd.name
  and make_err er = raise (Failure er) (* Helper to throw error with msg = er *)
  and n = fd.name
  in match fd with
      _ when StringMap.mem n built_in_decls -> make_err built_in_err
    | _ when StringMap.mem n map -> make_err dup_err
    | _ -> StringMap.add n fd map
in

(* Make the symbol table starting with the built-in functions *)
let function_decls = List.fold_left add_func built_in_decls functions
in

(* Returning the added function *)
let find_func s =
  try StringMap.find s function_decls
  with Not_found -> raise (Failure ("function not found: " ^ s))
in

let _ = find_func "main" (* main must exist as entrypoint *)
in

(* check function bodies *)
let check_function func =
  (* All #TODO: *)
  (* check type and identifiers in formal parameters and local vars *)
  (* check all assignments are valid types. Should we co-erce? *)
  let check_assign lvaltype rvaltype err =
    (* print_string ("param: " ^ (string_of_typ lvaltype) ^ " actual: " ^ (string_of_typ rvaltype) ^ "\n"); *)
    match lvaltype with
      (* Lint -> if rvaltype = String || rvaltype = Lint then lvaltype else raise (Failure err) *)
    | _    -> if lvaltype = rvaltype then lvaltype else raise (Failure err)
    (* if lvaltype is lint and rvaltype is string then lvaltype else raise failure*)
  in
  (* make local symbol table and functions to use it*)

  (* Build local symbol table of variables for this function *)
  let symbols = List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
                      StringMap.empty (globals @ func.params @ func.locals )
  in

  (* Return a variable from our local symbol table *)
  let type_of_identifier s =
      try StringMap.find s symbols
      with Not_found -> raise (Failure ("undeclared identifier " ^ s))
  in

```

```
(* semantic expression checking *)
let rec expr = function
    Lit l -> (Int, SLit l)
  | Id s -> (type_of_identifier s, SId s)
  | Strlit l -> (String, SStrlit l) (* String literals *)
  | Lintlit l -> (Lint, SLintlit l)
  | Noexpr   -> (Void, SNoexpr)
  | Assign(var, e) as ex ->
          let lt = type_of_identifier var
          and (rt, e') = expr e in
          let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
            string_of_typ rt ^ " in " ^ string_of_expr ex
          in (check_assign lt rt err, SAssign(var, (rt, e')))
  | Ptlit(e1, e2, e3) ->
      let (t1, e1') = expr e1
      and (t2, e2') = expr e2
      and (t3, e3') = expr e3 in
      let ty = match t1, t2, t3 with
      Lint, Lint, Poly -> Point
      | _ -> raise (Failure ("points must have Lint coordinates and be defined under a Poly"))
          in (ty, SPtlit((t1, e1'), (t2, e2'), (t3, e3')))
  | Access(var, e2) as ex -> (* Will give us the right index for gep from string *)
      let lt = type_of_identifier var in
      (match lt with
        Point -> (match e2 with
                    "x"     -> (Lint, SAccess(var, 0))
                  | "y"     -> (Lint, SAccess(var, 1))
                  | _   -> raise (Failure ("invalid access element " ^ e2 ^ " in "
                                                 ^ string_of_expr ex)))
      | _     -> raise (Failure ("cannot access type: " ^ string_of_typ lt
                                  ^ " in " ^ string_of_expr ex)))
  | Polylit(e1, e2, e3) ->
      let (t1, e1') = expr e1
      and (t2, e2') = expr e2
      and (t3, e3') = expr e3 in
      let ty = match t1, t2, t3 with
      Lint, Lint, Lint -> Poly
      | _ -> raise (Failure ("Polynomials must have Lint coefficients and a Lint modulus"))
            in (ty, SPolylit((t1, e1'), (t2, e2'), (t3, e3')))
    | Unop(op, e) as ex ->
            let (t, e') = expr e in
            let ty = match op with
              Neg when t = Int -> Int
            | Not when t = Int -> Int
            | Neg when t = Lint -> Lint
            | Not when t = Lint -> Lint
            | Neg when t = Point -> Point
            | _ -> raise (Failure ("illegal unary operator " ^
                                   string_of_uop op ^ string_of_typ t ^
                                   " in " ^ string_of_expr ex))
            in (ty, SUnop(op, (t, e')))
    | Binop(e1, op, e2) as e ->
            let (t1, e1') = expr e1
            and (t2, e2') = expr e2 in
            (* All binary operators require operands of the same type *)
            let same = t1 = t2 in
            (* Determine expression type based on operator and operand types *)
            let ty = match op with
              Add | Sub | Mul | Div | Mod | Pow when same && t1 = Int -> Int
            | Add | Sub | Mul | Div | Mod | Inv when same && t1 = Lint -> Lint
            | Add                               when same && t1 = Point -> Point
      | Pow                               when t1 = Lint && t2 = Int -> Lint
            | Mul                               when t1 = Lint && t2 = Point -> Point
            | Mul                               when t2 = Lint && t1 = Point -> Point
      | Beq | Bneq | Leq | Geq | Lth | Gth | And | Or when same && t1 = Int -> Int
            | Beq | Bneq | Leq | Geq | Lth | Gth             when same && t1 = Lint -> Int
            | _ -> raise (
                Failure ("illegal binary operator " ^
                        string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                        string_of_typ t2 ^ " in " ^ string_of_expr e))
```

```
            in (ty, SBinop((t1, e1'), op, (t2, e2')))
    | Relop(e1, op, e2) as e ->
            let (t1, e1') = expr e1
            and (t2, e2') = expr e2 in
            let same = t1 = t2 in
            let ty = match op with
            | Beq | Bneq | Leq | Geq | Lth | Gth | And | Or when same && t1 = Int -> Int
            | Beq | Bneq                                 when same && t1 = Point -> Int
            | Beq | Bneq | Leq | Geq | Lth | Gth | And | Or when same && t1 = Lint -> Int
            | _ -> raise (
                Failure ("illegal relational operator " ^
                        string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                        string_of_typ t2 ^ " in " ^ string_of_expr e))
            in (ty, SRelop((t1, e1'), op, (t2, e2')))
    | Trnop(e1, o1, e2, o2, e3) as e ->
            let (t1, e1') = expr e1
            and (t2, e2') = expr e2
            and (t3, e3') = expr e3 in
            let ty = match o1, o2 with
            Lpw, Pmd when t1 = Lint && t2 = Lint && t3 = Lint -> Lint
          | _ -> raise (
              Failure ("illegal ternary operator " ^ string_of_typ t1 ^ " " ^
              string_of_top o1 ^ " " ^ string_of_typ t2 ^ " " ^ string_of_top o2 ^ " " ^
              string_of_typ t3 ^ " in " ^ string_of_expr e))
            in (ty, STrnop((t1, e1'), o1, (t2, e2'), o2, (t3, e3')))
    | Call(name, args) (* as call *) ->
        let fd = find_func name in
        let param_length = List.length fd.params in
        if List.length args != param_length then
          raise (Failure ("expecting " ^ string_of_int param_length ^
                          " arguments in " ^ name))
        else let check_call (param_typ, _) e = (* validate call *)
          let (et, e') = expr e in (* recursively semantic check expr *)
          let err = "illegal argument " ^ string_of_typ et ^
          " expected " ^ string_of_typ param_typ ^ " in " ^ string_of_expr e
          in (check_assign param_typ et err, e')
        in
        let args' = List.map2 check_call fd.params args
        in (fd.typ, SCall(name, args'))
  in

  let check_int_expr e =
      let (t', e') = expr e
      and err = "expected integer expression in " ^ string_of_expr e
      in if t' != Int then raise (Failure err) else (t', e')
    in

  (* Here is where we check statements (only expr and Block for now)*)
  let rec check_stmt = function
    Expr e -> SExpr (expr e) (* recursive check *)
  | Return e -> let (t, e') = expr e in
      if t = func.typ then SReturn (t, e') (* Correct return type for function *)
      else raise (Failure "wrong return type")
  | Block sl ->
      let rec check_stmt_list = function (* Maybe add other return checks here *)
        [Return _ as s] -> [check_stmt s]
      | Return _ :: _   -> raise (Failure "nothing may follow a return")
      | Block sl :: ss  -> check_stmt_list (sl @ ss) (* Flatten blocks *)| s :: ss -> check_stmt s :: check_stmt_list ss (* one statement at a time *)
      | []        -> [] (* done *)
      in SBlock(check_stmt_list sl)
  | If(p, b1, b2) -> SIf(check_int_expr p, check_stmt b1, check_stmt b2)
  | For(e1, e2, e3, st) -> SFor(expr e1, check_int_expr e2, expr e3, check_stmt st)
  | While(p, s) -> SWhile(check_int_expr p, check_stmt s)
  in
  { styp = func.typ;
    sname = func.name;
    sparams = func.params;
    slocals = func.locals;
    sbody = match check_stmt (Block func.body) with
```

```
        SBlock(sl) -> sl
      | _ -> raise (Failure ("blocking failed"))
  }
in (globals, List.map check_function functions)
```

9.6 codegen.ml

```
(* This file will be used to get LLVM to work for our compiler as an IR *)

(* 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
open Sast

module StringMap = Map.Make(String)

(* translate : Sast.program -> Llvm.module *)
let translate (globals, functions) =
  let context    = L.global_context () in

  (* Create the LLVM compilation module into which
     we will generate code *)
  let the_module = L.create_module context "Prime" in

  (* Get types from the context *)
  let i32_t      = L.i32_type    context
  and i8_t       = L.i8_type     context
  and void_t     = L.void_type   context in
  let string_t   = L.pointer_type (i8_t)
  (* and mpz_t      = L.struct_type context [| (L.i32_type context); (L.i32_type context
    ); (L.pointer_type (L.i64_type context)) |] *)
(* in *)
  and mpz_t      = L.named_struct_type context "mpz_t"
    in let mpz_t = L.struct_set_body mpz_t [| (L.i32_type context); (L.i32_type context)
    ; L.pointer_type (L.i64_type context) |] false; mpz_t
  in let poly_t = L.named_struct_type context "poly"
    in let poly_t = L.struct_set_body poly_t [| mpz_t ; mpz_t; mpz_t |] false; poly_t
  in let point_t    = L.named_struct_type context "point"
    in let point_t = L.struct_set_body point_t [| mpz_t ; mpz_t; poly_t |] false;
    point_t
  in

  (* Return the LLVM type for a MicroC type *)
  let ltype_of_typ = function
    A.String   -> string_t
  | A.Lint     -> mpz_t
  | A.Point    -> point_t
  | A.Poly     -> poly_t
  | A.Int      -> i32_t
  | A.Void     -> void_t
  | _          -> void_t
  in

  (* Create a map of global variables after creating each *)
  let global_vars : L.llvalue StringMap.t =
    let global_var m (t, n) =
      let init = match t with
        _ -> L.const_int (ltype_of_typ t) 0
      in StringMap.add n (L.define_global n init the_module) m in
    List.fold_left global_var StringMap.empty globals in

  let printf_t : L.lltype =
      L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let printf_func : L.llvalue =
```

```
    L.declare_function "printf" printf_t the_module in

(* Declare our external functions here*)

(* LINTS *)
let linit_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; string_t; i32_t |] in
let linit_func : L.llvalue =
    L.declare_function "__gmpz_init_set_str" linit_t the_module in
let lcast_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; i32_t |] in
let lcast_func : L.llvalue =
    L.declare_function "__gmpz_init_set_si" lcast_t the_module in
let linitdup_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t |] in
let linitdup_func : L.llvalue =
    L.declare_function "__gmpz_init_set" linitdup_t the_module in
    (* The following would be needed in future work to free mpz memory *)
(* let lclear_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t |] in
let lclear_func : L.llvalue = (* free lints - define usage *)
    L.declare_function "__gmpz_clear" lclear_t the_module in *)
(* We don't use the mpz_out_str because FILE* is a pain *)
let lprint_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t |] in
let lprint_func : L.llvalue =
    L.declare_function "printl" lprint_t the_module in
let ladd_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t;
                             L.pointer_type mpz_t |] in
let ladd_func : L.llvalue =
    L.declare_function "__gmpz_add" ladd_t the_module in
let lsub_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t;
                             L.pointer_type mpz_t |] in
let lsub_func : L.llvalue =
    L.declare_function "__gmpz_sub" lsub_t the_module in
let lmul_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t;
                             L.pointer_type mpz_t |] in
let lmul_func : L.llvalue =
    L.declare_function "__gmpz_mul" lmul_t the_module in
let ldiv_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t;
                             L.pointer_type mpz_t |] in
let ldiv_func : L.llvalue =
    L.declare_function "__gmpz_tdiv_q" ldiv_t the_module in
let lmod_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t;
                             L.pointer_type mpz_t |] in
let lmod_func : L.llvalue =
    L.declare_function "__gmpz_tdiv_r" lmod_t the_module in
(* This power function will be used to raise to an unsigned int power *)
let lpow_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t; i32_t |] in
let lpow_func : L.llvalue =
    L.declare_function "__gmpz_pow_ui" lpow_t the_module in
let linv_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t;
                             L.pointer_type mpz_t |] in
let linv_func : L.llvalue =
    L.declare_function "__gmpz_invert" linv_t the_module in
let lpowmod_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t;
                             L.pointer_type mpz_t; L.pointer_type mpz_t |] in
let lpowmod_func : L.llvalue =
    L.declare_function "__gmpz_powm" lpowmod_t the_module in
let lneg_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t; |] in
let lneg_func : L.llvalue =
    L.declare_function "__gmpz_neg" lneg_t the_module in
let lnot_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t; |] in
```

```
let lnot_func : L.llvalue =
     L.declare_function "lnot_func" lnot_t the_module in

(* comparator operators *)
let l_eq_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t |] in
let l_eq_func : L.llvalue =
    L.declare_function "eq_func" l_eq_t the_module in
let l_neq_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t |] in
let l_neq_func : L.llvalue =
    L.declare_function "neq_func" l_neq_t the_module in
let l_lth_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t |] in
let l_lth_func : L.llvalue =
    L.declare_function "lth_func" l_lth_t the_module in
let l_gth_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t |] in
let l_gth_func : L.llvalue =
    L.declare_function "gth_func" l_gth_t the_module in
let l_leq_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t |] in
let l_leq_func : L.llvalue =
    L.declare_function "leq_func" l_leq_t the_module in
let l_or_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t |] in
let l_or_func : L.llvalue =
    L.declare_function "or_func" l_or_t the_module in
let l_and_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t |] in
let l_and_func : L.llvalue =
    L.declare_function "and_func" l_and_t the_module in
let l_geq_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t |] in
let l_geq_func : L.llvalue =
    L.declare_function "geq_func" l_geq_t the_module in
let l_rand_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; L.pointer_type mpz_t;
    L.pointer_type mpz_t |] in
let l_rand_func : L.llvalue =
    L.declare_function "rand_func" l_rand_t the_module in

(*points and printing points*)
let init_lintpoint_t : L.lltype =
    L.function_type i32_t [| L.pointer_type point_t; L.pointer_type mpz_t ;
                             L.pointer_type mpz_t; L.pointer_type poly_t |] in
let init_point_func : L.llvalue =
    L.declare_function "Point" init_lintpoint_t the_module in
let print_point_t : L.lltype =
    L.function_type i32_t [| L.pointer_type point_t |] in
let print_point_func : L.llvalue =
    L.declare_function "printpt" print_point_t the_module in
let pt_add_t : L.lltype =
    L.function_type (L.pointer_type point_t) [| L.pointer_type point_t;
                                               L.pointer_type point_t |] in
let pt_add_func : L.llvalue =
    L.declare_function "ptadd" pt_add_t the_module in
let pt_mul_t : L.lltype  =
    L.function_type (L.pointer_type point_t) [| L.pointer_type mpz_t;
                                               L.pointer_type point_t |] in
let pt_mul_func : L.llvalue = (* pt multiplication - define usage *)
    L.declare_function "ptmul" pt_mul_t the_module in
let pt_neg_t : L.lltype =
    L.function_type (L.pointer_type point_t) [| L.pointer_type point_t |] in
let pt_neg_func : L.llvalue =
    L.declare_function "ptneg" pt_neg_t the_module in
let pt_eq_t : L.lltype =
    L.function_type i32_t [| L.pointer_type point_t; L.pointer_type point_t |] in
let pt_eq_func : L.llvalue =
    L.declare_function "pteq" pt_eq_t the_module in
let pt_neq_t : L.lltype =
    L.function_type i32_t [| L.pointer_type point_t; L.pointer_type point_t |] in
let pt_neq_func : L.llvalue =
```

```
    L.declare_function "ptneq" pt_neq_t the_module in

(*polys and printing polys*)
let init_poly_t : L.lltype =
    L.function_type i32_t [| L.pointer_type poly_t; L.pointer_type mpz_t ; L.pointer_type mpz_t;  L.pointer_type mpz_t |] in
let init_poly_func : L.llvalue =
    L.declare_function "Poly" init_poly_t the_module in
let print_poly_t : L.lltype =
    L.function_type i32_t [| L.pointer_type poly_t |] in
let print_poly_func : L.llvalue =
    L.declare_function "printc" print_poly_t the_module in

(* Encoding and decoding strings for encryption *)
let encode_t : L.lltype =
    L.function_type i32_t [| L.pointer_type mpz_t; string_t |] in
let encode_func : L.llvalue =
    L.declare_function "encode" encode_t the_module in
let decode_t : L.lltype =
    L.function_type string_t [| L.pointer_type mpz_t |] in
let decode_func : L.llvalue =
    L.declare_function "decode" decode_t the_module in

(* Define each function (arguments and return type) so we can
   call it even before we've created its body *)
let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
  let function_decl m fdecl =
    let name = fdecl.sname(*sfname*)
    and formal_types =
Array.of_list (let new_params = (match fdecl.styp with
                                    A.Lint   -> (A.Lint, "sret") :: fdecl.sparams
                                  | A.Point -> (A.Point, "sret") :: fdecl.sparams
                                  | _       -> fdecl.sparams
                                  ) in
                 List.map (fun (t,n) -> match t with
                   A.Lint  when n = "sret" -> L.pointer_type (ltype_of_typ t)
                 | A.Point when n = "sret" -> L.pointer_type (ltype_of_typ t)
                 | _       -> ltype_of_typ t) new_params)
    in let ftype = L.function_type (match fdecl.styp with
                                      A.Lint -> L.pointer_type mpz_t
                                    | A.Point -> L.pointer_type point_t
                                    | _       -> ltype_of_typ fdecl.styp) 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.sname function_decls in
  let builder = L.builder_at_end context (L.entry_block the_function) in
  let new_params = (match fdecl.styp with
                       A.Lint -> (A.Lint, "sret") :: fdecl.sparams
                     | A.Point -> (A.Point, "sret") :: fdecl.sparams
                     | _       -> fdecl.sparams
                     ) in
  let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder
  and string_format_str = L.build_global_stringptr "%s\n" "fmt" builder
  (* and point_format_str = L.build_global_stringptr "[%\n" "fmt" 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 *)
  let local_vars =
    let add_formal m (t, n) p =
      L.set_value_name n p;
    let local = L.build_alloca (match t with
                                  A.Lint when n = "sret" -> L.pointer_type (ltype_of_typ t)
                                | A.Point when n = "sret" -> L.pointer_type (ltype_of_typ t)
                                | _       -> ltype_of_typ t) n builder in
          ignore (L.build_store p local builder);
```

```
      StringMap.add n local m

      (* Allocate space for any locally declared variables and add the
       * resulting registers to our map *)
      and add_local m (t, n) =
  let local_var = L.build_alloca (ltype_of_typ t) n builder
  in StringMap.add n local_var m
      in
      let formals = List.fold_left2 add_formal StringMap.empty new_params (*fdecl.sparams*)
          (Array.to_list (L.params the_function)) in
      List.fold_left add_local formals fdecl.slocals
    in

    (* Return the value for a variable or formal argument.
       Check local names first, then global names *)
    let lookup n = try StringMap.find n local_vars
                   with Not_found -> StringMap.find n global_vars
    in

    (* Helper function to deal with unassigned lint lits
       Returns: mpz_t pointer to be used for function args *)
    let llit_helper i =
      let lstr = L.build_global_stringptr i "string" builder
      and space = L.build_alloca (ltype_of_typ A.Lint) "" builder in
      let calls = ignore(L.build_call linit_func
        [| L.build_in_bounds_gep space [| L.const_int i32_t 0 |] "" builder; lstr; L.const_int i32_t 10 |]
        "__gmpz_init_set_str" builder);
        L.build_in_bounds_gep space [| L.const_int i32_t 0 |] "" builder
      in calls
      (* how to free after done using *)
    in


    (* Helpful when writing geps *)
    let zero = L.const_int i32_t 0
    in

    (* Construct code for an expression; return its value *)
    let rec expr builder ((stype, e) : sexpr) = match e with
        SStrlit i      -> L.build_global_stringptr i "string" builder
      | SLintlit i     -> llit_helper i (* Pointer to new mpz*)
      | SLit i         -> L.const_int i32_t i
      | SPtlit (i, j, p) -> (* call our struct initialiser passing in loc of initialisation *)
          let e1' = expr builder i
          and e2' = expr builder j
          and e3' = expr builder p
          and space = L.build_alloca point_t "tmp_pt" builder
          in ignore(L.build_call init_point_func [| space; e1' ; e2'; e3' |] "Point" builder); space
      | SPolylit (i, j, m) -> (* call our struct initialiser passing in loc of initialisation *)
          let e1' = expr builder i
          and e2' = expr builder j
          and e3' = expr builder m
          and space = L.build_alloca poly_t "tmp_poly" builder
          in ignore(L.build_call init_poly_func [| space; e1' ; e2'; e3' |] "Poly" builder); space
      | SNoexpr        -> L.const_int i32_t 0
      | SId s          -> (match stype with (* Might be better just to have StructType adt? *)
                              A.Lint  -> L.build_in_bounds_gep (lookup s) [| zero |] s builder
                            | A.Point -> L.build_in_bounds_gep (lookup s) [| zero |] s builder
                            | A.Poly -> L.build_in_bounds_gep (lookup s) [| zero |] s builder
                            | _       -> L.build_load (lookup s) s builder)
      | SAssign (s, ((A.Lint, _) as e1)) -> let e1' = expr builder e1 in
                (* Here we have a pointer to mpz val *)
                ignore(L.build_call linitdup_func
```

```
                [| L.build_in_bounds_gep (lookup s) [| zero |] s builder; e1' |] "" builder); e1'
    | SAssign (s, ((A.Point, _) as e1))  ->
        (* For point lits that already have stack allocated, we get element pointer then store *)
        let e1' = expr builder e1 in
        let val_ptr = L.build_in_bounds_gep e1' [| zero |] "" builder in
        let loaded = L.build_load val_ptr "" builder in
        ignore(L.build_store loaded (lookup s) builder); e1'
    | SAssign (s, ((A.Poly, _) as e1))  ->
        let e1' = expr builder e1 in
        let val_ptr = L.build_in_bounds_gep e1' [| zero |] "" builder in
        let loaded = L.build_load val_ptr "" builder in
        ignore(L.build_store loaded (lookup s) builder); e1'
    | SAssign (s, e) -> let e' = expr builder e in
                          ignore(L.build_store e' (lookup s) builder); e'
      (* Will need to separate out the access into one for the different types *)
    | SAccess (s, idx) ->
        let outer_ptr = L.build_in_bounds_gep (lookup s) [| zero; L.const_int i32_t idx |] "outer" builder
        in
        L.build_in_bounds_gep outer_ptr [| zero |] "inner" builder
    | SBinop ((A.Point, _) as e1, operator, e2) ->
             let e1' = expr builder e1
             and e2' = expr builder e2 in
             (match operator with
             A.Add -> (L.build_call pt_add_func [| e1'; e2' |] "pt_add" builder)
           | A.Mul -> L.build_call pt_mul_func [| e2'; e1' |] "pt_mul" builder
           | _ -> raise (Failure "Operator not implemented for Point"))
    (*special binop for lint times pt*)
    | SBinop (((A.Lint, _) as e1), operator, ((A.Point, _) as e2)) ->
            let e1' = expr builder e1
            and e2' = expr builder e2 in
            (match operator with
               A.Mul -> L.build_call pt_mul_func [| e1'; e2' |] "pt_mul" builder
            | _ -> raise (Failure "Operator not implemented for Lint, Point"))

    | SBinop ((A.Lint, _) as e1, operator, e2) ->
    (* for e1, e2 take second argument of the tuple (A.Lint, _) and do what printl does.
       * See if its an id or lintlit. If id get inbounds elt pointer to struct.
       * If its lintlit use helper function to make new mpz_t and get pointer to it
       * Helper function will return a 1d array. Concat 2 1elt array. call OCaml array.append
       * Pass this to Add *)
            let e1' = expr builder e1
            and e2' = expr builder e2
            and tmp = llit_helper "0" in
            ignore((match operator with
                       A.Add -> L.build_call ladd_func [| tmp; e1'; e2' |] "__gmpz_add" builder
                     | A.Sub -> L.build_call lsub_func [| tmp; e1'; e2' |] "__gmpz_sub" builder
                     | A.Mul -> L.build_call lmul_func [| tmp; e1'; e2' |] "__gmpz_mul" builder
                     | A.Div -> L.build_call ldiv_func [| tmp; e1'; e2' |] "__gmpz_tdiv_q"
                                 builder
                     | A.Mod -> L.build_call lmod_func [| tmp; e1'; e2' |] "__gmpz_tdiv_r"
                                 builder
                     | A.Pow -> L.build_call lpow_func [| tmp; e1'; e2' |] "__gmpz_pow_ui"
                                 builder
                     | A.Inv -> L.build_call linv_func [| tmp; e1'; e2' |] "__gmpz_invert"
                                 builder (* add handling for inv does not exist *)
                     | _ -> raise (Failure "Binary operator not implemented for Lint")
                  )); tmp
    | SRelop ((A.Lint, _) as e1, operator, e2) ->
             let e1' = expr builder e1
             and e2' = expr builder e2 in (match operator with
                  A.Beq -> L.build_call l_eq_func [| e1'; e2' |] "eq_func" builder
```

```
                 | A.Bneq -> L.build_call l_neq_func [| e1'; e2' |] "neq_func" builder
                 | A.Lth -> L.build_call l_lth_func [| e1'; e2' |] "lth_func" builder
                 | A.Gth -> L.build_call l_gth_func [| e1'; e2' |] "gth_func" builder
                 | A.Leq -> L.build_call l_leq_func [| e1'; e2' |] "leq_func" builder
                 | A.Geq -> L.build_call l_geq_func [| e1'; e2' |] "geq_func" builder
                 | A.And -> L.build_call l_and_func [| e1'; e2' |] "and_func" builder
                 | A.Or  -> L.build_call l_or_func [| e1'; e2' |] "or_func" builder
                 | _ -> raise (Failure "Relational operator not implemented for Lint")
               )
  | SRelop ((A.Point, _) as e1, operator, e2) ->
          let e1' = expr builder e1
          and e2' = expr builder e2 in
          (match operator with
            A.Beq -> L.build_call pt_eq_func [| e1'; e2' |] "eq_func" builder
          | A.Bneq -> L.build_call pt_neq_func [| e1'; e2' |] "neq_func" builder
          | _ -> raise (Failure "Relational operator not implemented for Point")
          )
  | SRelop (e1, operator, e2) ->
          let e1' = expr builder e1
          and e2' = expr builder e2 in
          (match operator with
            A.And     -> L.build_zext
                                (L.build_and
                                (L.build_icmp L.Icmp.Ne e1' (L.const_int i32_t 0) "tmp" builder)
                                (L.build_icmp L.Icmp.Ne e2' (L.const_int i32_t 0) "tmp" builder)
                                "tmp" builder) i32_t "tmp" builder
          | A.Or    -> L.build_zext
                                (L.build_or
                                (L.build_icmp L.Icmp.Ne e1' (L.const_int i32_t 0) "tmp" builder)
                                (L.build_icmp L.Icmp.Ne e2' (L.const_int i32_t 0) "tmp" builder)
                                "tmp" builder) i32_t "tmp" builder
          | A.Beq     -> L.build_zext (L.build_icmp L.Icmp.Eq e1' e2' "tmp" builder)
                                i32_t "tmp" builder
          | A.Bneq    -> L.build_zext (L.build_icmp L.Icmp.Ne e1' e2' "tmp" builder) i32_t
                                "tmp" builder
          | A.Lth     -> L.build_zext (L.build_icmp L.Icmp.Slt e1' e2' "tmp" builder) i32_t
                                "tmp" builder
          | A.Leq     -> L.build_zext (L.build_icmp L.Icmp.Sle e1' e2' "tmp" builder) i32_t
                                "tmp" builder
          | A.Gth     -> L.build_zext (L.build_icmp L.Icmp.Sgt e1' e2' "tmp" builder) i32_t
                                "tmp" builder
          | A.Geq     -> L.build_zext (L.build_icmp L.Icmp.Sge e1' e2' "tmp" builder) i32_t
                               "tmp" builder
          | _ -> raise (Failure "Relational operator not implemented"))
  | SBinop (e1, operator, e2) ->
          let e1' = expr builder e1
          and e2' = expr builder e2 in
          (match operator with
            A.Add     -> L.build_add e1' e2' "tmp" builder
          | A.Sub     -> L.build_sub e1' e2' "tmp" builder
          | A.Mul     -> L.build_mul e1' e2' "tmp" builder
          | A.Div     -> L.build_sdiv e1' e2' "tmp" builder
          | A.Mod     -> L.build_srem e1' e2' "tmp" builder
          | _ -> raise (Failure "Binary operator not implemented")
          )
  | SUnop(op, ((A.Lint, _) as e)) ->
          let e' = expr builder e
```

```
          and tmp = llit_helper "0" in
          ignore(match op with
            A.Neg -> L.build_call lneg_func [| tmp; e' |] "__gmpz_neg" builder
          | A.Not -> L.build_call lnot_func [| tmp; e' |] "lnot_func" builder
          ); tmp
  | SUnop(op, ((A.Point, _) as e)) ->
          let e' = expr builder e in
          (match op with
             A.Neg -> (L.build_call pt_neg_func [|e'|] "ptneg" builder)
           |  _ -> raise (Failure "Unary operator not implemented")
          )
  | SUnop(op, ((_, _) as e)) ->
          let e' = expr builder e in
          (match op with
            A.Neg     -> L.build_neg  e' "tmp" builder
          | A.Not     -> (L.build_zext
                         (L.build_icmp L.Icmp.Eq e' (L.const_int i32_t 0) "tmp" builder)
                         i32_t "tmp" builder))
  | STrnop(e1, o1, e2, o2, e3) ->
          let e1' = expr builder e1
          and e2' = expr builder e2
          and e3' = expr builder e3
          and out = llit_helper "0" in
          ignore((match o1, o2 with
              A.Lpw, A.Pmd ->
                  L.build_call lpowmod_func [| out; e1'; e2'; e3' |] "__gmpz_powm" builder
              | _ -> raise (Failure "Trinary operator not implemented"))
            );
          out
  | SCall ("print", [e]) -> (*keep print delete printb printf*)
      L.build_call printf_func [| int_format_str ; (expr builder e) |]
      "printf" builder
  | SCall ("prints", [e]) -> (*print string*)
      L.build_call printf_func [| string_format_str ; (expr builder e) |]
      "printf" builder
  | SCall ("printpt", [(_, e) as e1]) -> (* print pt *)
      let e1' = expr builder e1 in
      (match e with
        SPtlit _ -> L.build_call print_point_func [| L.build_in_bounds_gep e1' [| zero |] "" builder |] "printpt" builder
      | _                -> L.build_call print_point_func [| e1' |] "printpt" builder)
  | SCall ("printc", [(_, e) as e1]) -> (* print poly *)
      let e1' = expr builder e1 in
      (match e with
        SPolylit _ -> L.build_call print_poly_func [| L.build_in_bounds_gep e1' [| zero |] "" builder |] "printc" builder
      | _                -> L.build_call print_poly_func [| e1' |] "printc" builder)
  | SCall ("printl", [(_, e) as ptr]) ->
      (* L.build_call lprint_func [| expr builder e |] "printl" builder *)
      L.build_call lprint_func (match e with
        SId s -> [| (L.build_in_bounds_gep (lookup s)
                    [| L.const_int i32_t 0 |] "" builder) |]
      | SLintlit i -> [| llit_helper i |]
      | _      -> [| expr builder ptr |]) "printl" builder
  | SCall ("tolint", [e]) -> (* allocate some lint space and init with value *)
      let space = L.build_alloca mpz_t "tmp_lint" builder in
      let ptr   = L.build_in_bounds_gep space [| zero |] "" builder
      and e'    = expr builder e in
      ignore(L.build_call lcast_func [| ptr; e' |] "__gmpz_init_set_si" builder); ptr
  | SCall ("random", [e1;e2]) ->
      let rnd = llit_helper "0"
      and sed = expr builder e1
      and max = expr builder e2 in
      ignore(L.build_call l_rand_func [| rnd; sed; max |] "rand_func" builder); rnd
  | SCall ("decode", [e]) ->
      let e' = expr builder e in
      L.build_call decode_func [| e' |] "decode" builder
  | SCall ("encode", [e]) ->
      let e' = expr builder e
      and ret_space = L.build_alloca mpz_t "" builder in
```

```
       let ret_ptr = L.build_in_bounds_gep ret_space [| zero |] "" builder in
       ignore(L.build_call encode_func [| ret_ptr; e' |] "encode" builder); ret_ptr
   | SCall (f, args) ->
       let (fdef, fdecl) = StringMap.find f function_decls in
  (* let args = match fdecl.styp with
         A.Lint -> (A.Lint, "sret") :: args
     | _        -> args in *)
let llargs = List.rev (List.map (fun (ty, se) -> match ty with
                           A.Lint -> L.build_load (expr builder (ty, se)) "lint_param" builder
                         | A.Point -> L.build_load (expr builder (ty, se)) "pt_param" builder
                         | _       -> expr builder (ty, se)) (List.rev args)) in
let result = (match fdecl.styp with
                     A.Void -> ""
                   | _ -> f ^ "_result") in
  let llargs = (match fdecl.styp with
                 A.Lint -> let space = L.build_alloca mpz_t "sret_space" builder
                           in
                           L.build_in_bounds_gep space [| zero |] "" builder :: llargs
               | A.Point -> let space = L.build_alloca point_t "sret_space" builder
                           in
                           L.build_in_bounds_gep space [| zero |] "" builder :: llargs
               | _       -> llargs) in
      L.build_call fdef (Array.of_list llargs) result builder
   (* | _ -> L.const_int i32_t 0 *)
 in

 (* LLVM insists each basic block end with exactly one "terminator"
    instruction that transfers control.  This function runs "instr builder"
    if the current block does not already have a terminator.  Used,
    e.g., to handle the "fall off the end of the function" case. *)
 let add_terminal builder instr =
   match L.block_terminator (L.insertion_block builder) with
     Some _ -> ()
   | None -> ignore (instr builder) in

 (* Build the code for the given statement; return the builder for
    the statement's successor (i.e., the next instruction will be built
    after the one generated by this call) *)

 let rec stmt builder = function
     SBlock sl -> List.fold_left stmt builder sl
   | SExpr e -> ignore(expr builder e); builder
   | SReturn e -> ignore(match fdecl.styp with
                           (* Special "return nothing" instr *)
                             A.Void -> L.build_ret_void builder
                           (* Add return statements for structs *)
                           | A.Lint -> (*let local_val = match (snd e) with
                                   SId s  -> L.build_load (expr builder e) "val_ptr" builder
                                 | _       -> expr builder e;*)
                                 let local_val = expr builder e
                                 and loaded = L.build_load (lookup (snd (List.hd new_params))) "ret_ptr" builder in
                                 ignore(L.build_call linitdup_func
                                 [| loaded; local_val |] "ret_set" builder);
                                 L.build_ret loaded builder
                           | A.Point ->
                                 let local_val = expr builder e in
                                 let value = L.build_load local_val "" builder
                                 and loaded = L.build_load (lookup (snd (List.hd new_params))) "ret_ptr" builder in
                                 ignore(L.build_store value loaded builder);
                                 L.build_ret loaded builder
                           (* Build return statement *)
                           | _       -> L.build_ret (expr builder e) builder );
                  builder
   | SIf (predicate, then_stmt, else_stmt) ->
      let int_val = expr builder predicate in
      let bool_val = L.build_icmp L.Icmp.Ne int_val (L.const_int i32_t 0) "tmp"
```

```
    builder in
        (*L.const_int i1_t (* (if int_val = (L.const_int i32_t 0) then 0 else 1)*)
        ignore(match int_val with
          (L.const_int i32_t 0) -> 0
        | (L.const_int i32_t _) -> 1
        | _ -> raise(Failure "case")
        )*)
        let merge_bb = L.append_block context "merge" the_function in
            let build_br_merge = L.build_br merge_bb in (* partial function *)

        let then_bb = L.append_block context "then" the_function in
        add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
          build_br_merge;

        let else_bb = L.append_block context "else" the_function in
        add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
          build_br_merge;

        ignore(L.build_cond_br bool_val then_bb else_bb builder);
        L.builder_at_end context merge_bb

      | SWhile (predicate, body) ->
    let pred_bb = L.append_block context "while" the_function in
    ignore(L.build_br pred_bb builder);

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

    let pred_builder = L.builder_at_end context pred_bb in
          let int_val = expr pred_builder predicate in
    let bool_val = (L.build_icmp L.Icmp.Ne int_val (L.const_int i32_t 0)) "tmp"
    pred_builder 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

      (* Implement for loops as while loops *)
      | SFor (e1, e2, e3, body) -> stmt builder
      ( SBlock [SExpr e1 ; SWhile (e2, SBlock [body ; SExpr e3]) ] )
    in

    (* Build the code for each statement in the function *)
    let builder = stmt builder (SBlock fdecl.sbody) in

    (*#TODO:  We need some code to clear our lints so free all at the end of this
    function
       We will iterate through our locals map and add clears at the end of each of
       them
       have map that contains all the lints (does local vars work for this?) if not
       we need to make new map.
       Function called for each lint, check elt match type with A.Lint getelementptr
    inbounds
       and pass to lclear_t same way we pass stuff to printl.
       *)

    (* Add a return if the last block falls off the end *)
    add_terminal builder (match fdecl.styp 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
```

9.7 gmpfunc.c

```
// This file will be used to interface with OCaml LLVM
#include <stdio.h>
#include <gmp.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <time.h>
#include "structs.h"


void printl(mpz_t n)
{
    mpz_out_str(stdout, 10, n);
    printf("\n");
}

int rand_func(mpz_t rnd, mpz_t seed, mpz_t max)
{
    /*rand() into mpzt*/
    //mpz_t newseed;
    //mpz_init(newseed);
    if(mpz_sgn(seed) == 0)
    {
        srand(time(0));
        mpz_set_ui(seed, rand());
    }

    gmp_randstate_t state; /*intialize state */

    gmp_randinit_mt(state); /* set set state to use the Mersenne Twister Algorithm */
    gmp_randseed(state, seed); /*seed the state using user input*/

    mpz_urandomm(rnd, state, max); /*generate random int*/

    gmp_randclear(state);
    return(0);
}

char *sub(char *left, char *right)
{
    mpz_t n1;
    mpz_t n2;
    mpz_init(n1);
    mpz_init(n2);
    if (mpz_set_str(n1, left, 10) != 0){
        printf("Failed to assign number");
        mpz_clear(n1);
        mpz_clear(n2);
        exit(1);
    }
    if (mpz_set_str(n2, right, 10) != 0) {
        printf("Failed to assign number");
        mpz_clear(n1);
        mpz_clear(n2);
        exit(1);
    }
    mpz_sub(n1, n1, n2);
    char *ret_str = mpz_get_str(NULL, 10, n1);
    mpz_clear(n1);
    mpz_clear(n2);
    return ret_str;
}

int eq_func(mpz_t x, mpz_t y){
    if(mpz_cmp(x, y) == 0){
        return 1;
    }
    else{
        return 0;
    }
}

int neq_func(mpz_t x, mpz_t y){
    if(mpz_cmp(x, y) == 0){
        return 0;
    }
    else{
        return 1;
```

```
    }
}

int lth_func(mpz_t x, mpz_t y){
    if(mpz_cmp(x, y) < 0){
        return 1;
    }
    else{
        return 0;
    }
}

int gth_func(mpz_t x, mpz_t y){
    if(mpz_cmp(x, y) > 0){
        return 1;
    }
    else{
        return 0;
    }
}

int leq_func(mpz_t x, mpz_t y){
    if(mpz_cmp(x, y) <= 0){
        return 1;
    }
    else{
        return 0;
    }
}

int geq_func(mpz_t x, mpz_t y){
    if(mpz_cmp(x, y) >= 0){
        return 1;
    }
    else{
        return 0;
    }
}

int lnot_func(mpz_t out, mpz_t in){
    if(mpz_sgn(in) == 0)
    {
        mpz_set_str(out, "1", 10);
    }
    else
    {
        mpz_set_str(out, "0", 10);
    }
    return 0;
}

int and_func(mpz_t x, mpz_t y){
    if(mpz_get_si(x) == 0 || mpz_get_si(y) == 0)
    {
      return 0;
    }
    else
    {
      return 1;
    }
}

int or_func(mpz_t x, mpz_t y){
    if(mpz_get_si(x) == 0 && mpz_get_si(y) == 0)
    {
      return 0;
    }
    else
    {
      return 1;
    }
}

```

```
#ifdef BUILD_TEST
int main()
{

    // Create a lint through assignment to an id
    // char *id1 = "193475923745892734958723485839572 8";
    // printf("n = ");
    // printl(id1);
    // printf("\n");

    // // Do some operation(s) on lint
    // printf("Squaring:\n");
    // char *fun = pow(id1, 2);
    // printf("%s", fun);
    // printf("\n");


    // printf("Adding\n");
    // char *added = add("403527340975028473502743052893475 0",
    "139487619823469187364916398427");
    // printf("%s", added);
    // printf("\n");

    // // clean up
    // free(fun);
    // free(added);
    return 0;
}
#endif
```

9.8 structs.h

```
#ifndef STRUCTS_H
#define STRUCTS_H


struct poly {
    mpz_t x_coeff;
    mpz_t c;
    mpz_t mod;
};

struct point
{
  mpz_t i;
  mpz_t j;
  struct poly *curve;
};

#endif
```

9.9 structs.c

```
// This file will be used to interface with OCaml LLVM
#include <stdio.h>
#include <gmp.h>
#include <stdlib.h>
#include <string.h>

#include"structs.h"

// POLYS
void Poly(struct poly *p, mpz_t x_coeff, mpz_t c, mpz_t mod)
{
    mpz_init_set(p->x_coeff, x_coeff);
    mpz_init_set(p->c, c);
    mpz_init_set(p->mod, mod);
}

void printc(struct poly *p)
{
```

```c
    printf("[(");
    mpz_out_str(stdout, 10, p->x_coeff);
    printf(",");
    mpz_out_str(stdout, 10, p->c);
    printf(") : ");
    mpz_out_str(stdout, 10, p->mod);
    printf("]\n");
}


void Point(struct point *p, mpz_t i, mpz_t j, struct poly *curve)
{
    mpz_init_set(p->i, i);
    mpz_init_set(p->j, j);
    p->curve = curve;
}

void printpt(struct point *p)
{
    printf("[");
    mpz_out_str(stdout, 10, p->i);
    printf(",");
    mpz_out_str(stdout, 10, p->j);
    printf("] & ");
    printc(p->curve);
}
struct point *ptadd(struct point *p1, struct point *p2);
/*
char *printpt(struct point p){

    printf("[");
    mpz_out_str(stdout, 10, p.i);
    printf(",");
    mpz_out_str(stdout, 10, p.j);
    printf("]");
    printf("\n");
}*/
struct point *ptmul( mpz_t n, struct point *p1)
{
    /*struct poly *curve = (struct poly *)malloc(sizeof(struct poly));
    struct point *sum  = (struct point *)malloc(sizeof(struct point));

    mpz_t xcoeff;
    mpz_init_set(xcoeff, p1->curve->x_coeff);

    mpz_t c;
    mpz_init_set(c, p1->curve->c);

    mpz_t mod;
    mpz_init_set(mod, p1->curve->mod);

    Poly( curve, xcoeff, c, mod);*/

    struct point *product = p1;
    //copy n into new mpz_t
    mpz_t i;
    mpz_init(i);
    mpz_set(i, n);
    mpz_sub_ui( i, i, (unsigned long) 1);
    while( mpz_sgn(i) != 0)
    {
        product = ptadd(product, p1);
        mpz_sub_ui( i, i, (unsigned long) 1);
    }
    return product;
}
/*
struct point *ptadd(struct point *p1, struct point *p2){
    struct poly *curve = (struct poly *)malloc(sizeof(struct poly));
    struct point *sum  = (struct point *)malloc(sizeof(struct point));

    mpz_t xcoeff;
    mpz_init_set(xcoeff, p1->curve->x_coeff);
```

```

    mpz_t c;
    mpz_init_set(c, p1->curve->c);

    mpz_t mod;
    mpz_init_set(mod, p1->curve->mod);

    Poly( curve, xcoeff, c, mod);
    return ptaddhelper( sum, p1, p2);
}*/
int pteq(struct point *p1, struct point *p2)
{
    if(mpz_cmp(p1->i, p2->i) == 0 &&
            mpz_cmp(p1->j, p2->j) == 0 &&
            mpz_cmp(p1->curve->mod, p2->curve->mod) == 0 &&
            mpz_cmp(p1->curve->c, p2->curve->c) == 0 &&
            mpz_cmp(p1->curve->x_coeff, p2->curve->x_coeff) == 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int ptneq(struct point *p1, struct point *p2)
{
    if(mpz_cmp(p1->i, p2->i) != 0 ||
            mpz_cmp(p1->j, p2->j) != 0 ||
            mpz_cmp(p1->curve->mod, p2->curve->mod) != 0 ||
            mpz_cmp(p1->curve->c, p2->curve->c) != 0 ||
            mpz_cmp(p1->curve->x_coeff, p2->curve->x_coeff) != 0)
    {
        return 1;
    }
    else
    {

         return 0;
    }
}

struct point *ptneg(struct point *p1)
{
    struct poly *curve = (struct poly *)malloc(sizeof(struct poly));
    struct point *neg  = (struct point *)malloc(sizeof(struct point));

    mpz_t xcoeff;
    mpz_init_set(xcoeff, p1->curve->x_coeff);

    mpz_t c;
    mpz_init_set(c, p1->curve->c);

    mpz_t mod;
    mpz_init_set(mod, p1->curve->mod);

    Poly( curve, xcoeff, c, mod);
    if( mpz_sgn(p1->i) == -1 && mpz_sgn(p1->j) == -1 ){
        Point( neg, p1->i, p1->j, curve);
    }
    else{
        mpz_t inv;
        mpz_init(inv);
        mpz_neg(inv, p1->j);
        mpz_mod(inv, inv, mod);

        Point( neg, p1->i, inv, curve );

        mpz_clear(inv);
    }

    mpz_clear(xcoeff);
```

```c
    mpz_clear(c);
    mpz_clear(mod);

    return neg;
}

struct point *ptadd(struct point *p1, struct point *p2)
{
    struct poly *curve = (struct poly *)malloc(sizeof(struct poly));
    struct point *sum  = (struct point *)malloc(sizeof(struct point));

    mpz_t xcoeff;
    mpz_init_set(xcoeff, p1->curve->x_coeff);

    mpz_t c;
    mpz_init_set(c, p1->curve->c);

    mpz_t mod;
    mpz_init_set(mod, p1->curve->mod);

    Poly( curve, xcoeff, c, mod);

    mpz_t zero;
    mpz_init_set_str(zero, "0", 10);

    mpz_t p3x;
    mpz_t p3y;

    mpz_init(p3x);
    mpz_init(p3y);

    /* if pt is -1, -1 -> pt at infinity acts as identity element
     * return other point
     */

    if( mpz_sgn(p1->i) == -1 && mpz_sgn(p1->j) == -1 ){
        mpz_set(p3x, p2->i);
        mpz_set(p3y, p2->j);
        Point( sum, p3x, p3y, curve);


    } else if ( mpz_sgn(p2->i) == -1 && mpz_sgn(p2->j) == -1 ){
        mpz_set(p3x, p1->i);
        mpz_set(p3y, p1->j);
        Point( sum, p3x, p3y, curve);

    }
    else{
    /* build local x and y coords */

        mpz_t p1x;
        mpz_t p1y;
        mpz_t p2x;
        mpz_t p2y;

        mpz_init(p1x);
        mpz_init(p1y);
        mpz_init(p2x);
        mpz_init(p2y);

        mpz_mod(p1x, p1->i, mod);
        mpz_mod(p1y, p1->j, mod);
        mpz_mod(p2x, p2->i, mod);
        mpz_mod(p2y, p2->j, mod);

        /* check if they are inverses of one another */

        mpz_t neg;
        mpz_init(neg);
        mpz_neg(neg, p2y);
        if(mpz_congruent_p(p1y, neg, mod))
        {
            mpz_set_str(p3x, "-1", 10);
```

```
            mpz_set_str(p3y, "-1", 10);
            Point( sum, p3x, p3y, curve);

        /*mpz_clear(neg);
        mpz_clear(p1x);
        mpz_clear(p1y);
        mpz_clear(p2x);
        mpz_clear(p2y);*/

        /*mpz_clear(xcoeff);
        mpz_clear(c);
        mpz_clear(z:qero);
        mpz_clear(mod);*/

           // return sum;
        }
        else{

            //slope

            mpz_t m;
            mpz_init(m);

            /* if pts are not the same */
            if(mpz_cmp(p1x, p2x) != 0 || mpz_cmp(p1y, p2y) != 0)
            {
                mpz_t tmpy;
                mpz_t tmpx;
                mpz_init(tmpy);
                mpz_init(tmpx);

                mpz_sub(tmpy, p2y, p1y);
                mpz_sub(tmpy, tmpy, mod);
                mpz_sub(tmpx, p2x, p1x);
                mpz_mod(tmpx, tmpx, mod);

                mpz_invert(tmpx, tmpx, mod);
                mpz_mul(m, tmpy, tmpx);
                mpz_mod(m, m, mod);

                mpz_clear(tmpy);
                mpz_clear(tmpx);
            } else { /* if points are same */
                mpz_t tmpx;
                mpz_t tmpy;
                mpz_init(tmpx);
                mpz_init(tmpy);

                mpz_mul(tmpx, p1x, p1x);
                mpz_mod(tmpx, tmpx, mod);
                mpz_mul_si(tmpx, tmpx, (long) 3);
                mpz_mod(tmpx, tmpx, mod);
                mpz_add(tmpx, tmpx, xcoeff);
                mpz_mul_si(tmpy, p1y, (long) 2);
                mpz_mod(tmpy, tmpy, mod);
                mpz_invert(tmpy, tmpy, mod);
                mpz_mul(m, tmpx, tmpy);
                mpz_mod(m, m, mod);

                mpz_clear(tmpx);
                mpz_clear(tmpy);
            }

            /* find p3x */
            mpz_t tmp;
            mpz_init(tmp);
            mpz_mul(tmp, m, m);
            mpz_mod(tmp, tmp, mod);
            mpz_sub(tmp, tmp, p1x);
            mpz_sub(tmp, tmp, p2x);
            mpz_mod(tmp, tmp, mod);
            mpz_set(p3x, tmp);

```

```
            /* find p3y */

            mpz_sub(tmp, p1x, p3x);
            mpz_mul(tmp, tmp, m);
            mpz_sub(tmp, tmp, p1y);
            mpz_mod(tmp, tmp, mod);
            mpz_set(p3y, tmp);

            /* build pt */

            Point( sum, p3x, p3y, curve );

            mpz_clear(m);
            mpz_clear(tmp);
        }

        mpz_clear(neg);

        mpz_clear(p1x);
        mpz_clear(p1y);
        mpz_clear(p2x);
        mpz_clear(p2y);
    }

    mpz_clear(xcoeff);
    mpz_clear(c);
    mpz_clear(zero);
    mpz_clear(mod);


    mpz_clear(p3x);
    mpz_clear(p3y);


    return sum;
    /*int i, j;
    int m;
    int b = 1;

    if(p1.i == p2.i && p1.j == p2.j){
  m = (3*(p1.i)^2 + b)/(2*p1.j);
    }
    else{
  m = (p2.j-p1.j)/(p2.i-p1.i);
    }

    i = m^2 - p1.i - p2.i;
    j = m*(p1.i - i) - p1.j;

    return Point(i, j);*/
}
```

9.10 input.c

```
#include <stdio.h>
#include <gmp.h>
#include <string.h>
#include <stdlib.h>

// takes in pointer to mpz to update
void encode(mpz_t res, char *in)
{
    // keep output buff that handles padding length and null terminator
    // printf("%s\n", in);
    char outBuf[3 * strlen(in) + 1];
    int i;
    outBuf[0] = '\0';
    for (i = 0; i < strlen(in); i++) {
        int c = (int) in[i];
        char temp[4];
        sprintf(temp, "%03d", c);
        strncat(outBuf, temp, strlen(temp));
```

```c
        // printf("%s\n", outBuf);
    }
    mpz_init_set_str(res, outBuf, 10);
}

char *decode(mpz_t in)
{
    int i;
    char *lintStr = mpz_get_str(NULL, 10, in);
    int padLen = 3 - (strlen(lintStr) % 3);
    char *tmp = (char *) malloc(strlen(lintStr)+padLen+1); // will leak unless freed
    for (i = 0; i < padLen; i++)
        tmp[i] = '0';
    tmp[i] = '\0';

    strncat(tmp, lintStr, strlen(lintStr));
    free(lintStr);

    int newlength = strlen(tmp)/3 + 1;
    char *ret = (char *) malloc(newlength);
    for (i = 0; i < newlength; i++) {
        char buf[4];
        strncpy(buf, tmp+3*i, 3);
        buf[3] = '\0';
        char c = (char) atoi(buf);
        // printf("%c", c);
        ret[i] = c;
    }
    free(tmp);
    return ret;
}

#ifdef BUILD_TEST
int main()
{
    mpz_t res;
    // mpz_init(res);
    char testStr[] = "HelloWorld";
    encode(res, testStr);
    mpz_out_str(stdout, 10, res);
    printf("\n");
    char *retVal = decode(res);
    printf("%s\n", retVal);
    free(retVal);
}
#endif
```

9.11 prime.ml

```ocaml
(* Compiler command centre: tell sequence of actions here *)

type action = Ast | Sast | LLVM_IR | Compile

let () = (* don't care about return type *)
  let action = ref Compile in (* set default? *)
  let set_action a () = action := a in
  let options = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print LLVM");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in (* Only one mode for now *)
  let usage_msg = "usage: ./prime.native [-a|-c] <filename>" in
  let channel = ref stdin in
  (* take the options and a function that takes filename and opens it for reading *)
  Arg.parse options (fun filename -> channel := open_in filename) usage_msg;

  (* Start reading input *)
  let lexbuf = Lexing.from_channel !channel in (* ! operator dereferences *)
  (* Construct AST *)
  let ast = Parser.program Scanner.token lexbuf in
```

```
  match !action with
    Ast -> print_string (Ast.string_of_program ast)
  | _ -> let sast = Semant.check ast in
    match !action with (* add other options to stop at later *)
      Ast     -> ()
    | Sast    -> print_string (Sast.string_of_sprogram sast)
    | LLVM_IR -> let modu = Codegen.translate sast in
          print_string (Llvm.string_of_llmodule modu)
    | Compile -> let modu =
        Codegen.translate sast in
          Llvm_analysis.assert_valid_module modu;
          print_string (Llvm.string_of_llmodule modu)
```

9.12 Makefile

```
.PHONY : test
test : all test_all.sh
	./test_all.sh

.PHONY : all
all : clean gmp prime.native gmpfunc.o structs.o

# this will serve to install the GNU multiple precision library onto our system
.PHONY : gmp
gmp:
	apt install -y libgmp-dev

# We will now make the compiler
prime.native : codegen.ml sast.ml ast.ml semant.ml scanner.mll parser.mly
	opam config exec -- \
	ocamlbuild -use-ocamlfind prime.native

# Test the GMP calls we build
gmpfunc: gmp gmpfunc.c
	cc -o gmpfunc -DBUILD_TEST gmpfunc.c -lgmp

gmpfunc.o: gmp gmpfunc.c
	cc -c gmpfunc.c

structs: structs.c
	cc -o structs -DBUILD_TEST structs.c -lgmp

structs.o: structs.c
	cc -c structs.c

input: gmp input.c
	cc -o input -DBUILD_TEST input.c -lgmp

input.o: input.c
	cc -c input.c


# Some old stuff:
prime : parser.cmo scanner.cmo prime.cmo
	ocamlc -o prime $^

%.cmo : %.ml
	ocamlc -c $<

%.cmi : %.mli
	ocamlc -c $<

scanner.ml : scanner.mll
	ocamllex $^

parser.ml parser.mli : parser.mly
	ocamlyacc $^

# run the tests (without outputting to file)
prime.out: prime prime.tb
	./prime < prime.tb

```

```
# Depedencies from ocamldep
prime.cmo : scanner.cmo parser.cmi ast.cmi
prime.cmx : scanner.cmx parser.cmx ast.cmi
parser.cmo : ast.cmi parser.cmi
parser.cmx : ast.cmi parser.cmi
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx


##############################

# TARFILES = README Makefile scanner.mll ast.mli parser.mly prime.ml prime.tb

# hw1.tar.gz : $(TARFILES)
#   cd .. && tar zcf hw1/hw1.tar.gz $(TARFILES:%=hw1/%)

.PHONY : clean
clean :
  rm -rf *.cmi *.cmo parser.ml parser.mli scanner.ml prime.out prime
  rm -rf *.exe *.ll *.s *.test *.diff a.out gmpfunc gmpfunc.o structs structs.o input
    input.o
  opam config exec -- \
  ocamlbuild -clean
```

9.13 test_file.sh

```
#!/bin/bash
./prime.native $1.pr > $1.ll
llc -relocation-model=pic $1.ll > $1.s
cc -c gmpfunc.c
cc -c structs.c
gcc -o $1.exe $1.s gmpfunc.o structs.o -lgmp
./$1.exe
rm $1.ll;
rm $1.s;
rm $1.exe;
```

9.14 test_all.sh

```
#!/bin/bash
# time limit on operations
ulimit -t 30
logfile=tests.log
rm -rf $logfile
error=0
exitcode=0

IsError() {
    if [ $error -eq 0 ] ; then
    echo "FAILED"
    error=1
    fi
    # print out what we failed
    echo " $1"
}

Difference() {
    echo diff -b -q $1 $2 ">" $logfile 1>&2
    diff -b "$1" "$2" > "$1.diff" 2>&1 || {
    IsError "Difference in $1"
    }
}

# Run a command retaining error code
Run() {
    echo $* 1>&2
    eval $* || {
        IsError "$1 Failed (cmd: $*)"
        return 1
    }
}
```

```


Test() {
    error=0
    # extracting filename seen here: https://stackoverflow.com/questions/965053/extract-filename-and-extension-in-bash?page=1&tab=votes#tab-top
    filename=$(basename -- "$1")
    filename="${filename%.*}"

    echo -n "Test: $filename "
    # newline between tests
    echo 1>&2
    echo "#### Testing $1 ####" 1>&2

    # Run the various compilation parts
    Run "./prime.native" "$1" ">" "$filename.ll" &&
    Run "llc" "-relocation-model=pic" "$filename.ll" ">" "$filename.s" &&
    Run "cc" "-o" "$filename.exe" "$filename.s" "gmpfunc.o" "structs.o" "input.o" "-lgmp" &&
    Run "./$filename.exe" > "$filename.test" &&
    Difference $filename.test ./tests/$filename.out

    if [ $error -eq 0 ] ; then
        echo "OK"
        echo "#### Success" 1>&2
    else
        echo "#### FAIL" 1>&2
        exitcode=$error
    fi
}

RunFail() {
    echo $* 1>&2
    # Use short circuit && operator
    eval $* && {
        IsError "failed: $* did not show error"
        return 1
    }
    return 0
}

TestFail() {
    error=0
    # extracting filename seen here: https://stackoverflow.com/questions/965053/extract-filename-and-extension-in-bash?page=1&tab=votes#tab-top
    filename=$(basename -- "$1")
    filename="${filename%.*}"

    echo -n "Test: $filename "
    # newline between tests
    echo 1>&2
    echo "#### Testing $1 ####" 1>&2

    # This is a fila case so should not get past the compiler
    RunFail "./prime.native" "<" $1 "2>" "$filename.test" ">>" $logfile &&
    Difference $filename.test ./tests/$filename.out

    if [ $error -eq 0 ] ; then
        echo "OK"
        echo "#### Success" 1>&2
    else
        echo "#### FAIL" 1>&2
        exitcode=$error
    fi
}

# make sure C files ready
# Compile/link in gmpfunc file
cc -c gmpfunc.c
cc -c structs.c
cc -c input.c

# Run test_hello.pr
```

```
# check if specific files to test
if [ $# -ge 1 ]
then
    # provided specific files to test
    files=$@
else
    files="tests/*.pr"
fi

# run positive tests for now
for file in $files
do
    if [[ $file != *fail*.pr ]] ;
    then
        Test $file 2>> $logfile
    else
        TestFail $file 2>> $logfile
    fi
done

# clean up ()
# rm -rf *.exe *.test *.ll *.s

# print out so we can see return at the end
cat $logfile
exit $exitcode
```