Cryptal Language Final Report

Carolina Almirola | Jaewan Bahk | Rahul Kapur | Michail Oikonomou | Sammy Tbeile
ca2636 | jb3621 | rk2749 | mo2617 | st2198

Contents

1 Introduction

Cryptal is a C-like language designed to handle applications of number theory and cryptography. It provides users with a simple and intuitive environment from which to solve modular arithmetic problems. Currently, there are no well-documented or straightforward languages or packages that help alleviate the pains of modular arithmetic and complicated encryption schemes for users. Given the growing demand for more secure systems, a language designed for ease of implementation of encryption schemes is a valuable addition to the field of computer science and security engineering.

Cryptal provides built-in types and operators for use with modular calculations, allowing the user to focus on implementation of encryption capabilities and algorithms. Additionally, it provides basic C functionality and follows C structure and syntax.

1.1 Goals

1. Provide built-in types for modular integers and large numbers.
2. Facilitate modular operations, such as exponential and modular inverse.
3. C-like syntax for ease of use and familiarity.
4. Use built-in types to implement popular encryption algorithms.

2 Language Tutorial

2.1 Setup

OCaml must be installed in order to use the compiler, as well as LLVM v3.6 or higher. They can both be installed using OPAM:

```
$ sudo apt install ocaml llvm llvm-runtime m4 opam
$ opam init
$ opam install llvm.3.8
$ eval `opam config env`
```

Additionally, openSSL should be installed in the user's machine.

2.2 Getting Started

Within the Cryptal directory, type the make command. This creates the Cryptal to LLVM compiler, cryptal.native. Cryptal files use the extension .crp, and they are passed in to the compiler that generates LLVM code. For example, to compile the helloworld.crp

```
$ ./cryptal.native helloworld.crp > helloworld.ll
```

2.3 A basic Cryptal program

It is a requirement that the program have a main() function. Below is a simple example which demonstrates gems, lattice, and printing functions for both.

```
int main()
{
  gem a;
  gem b;
  lat c;

  a = (2, 7);
  b = (3, 7);
  c = "12345678902345678903456789O";

  print("%d\n", b-a);
  print_gem(a);
  print_lat(c);
  return 0;
}
```

This yields the output:

```
>> 1
>> 2
>> 12345678902345678903456789O
```

3 Language Reference Manual

3.1 Types

Types are used to store data in different formats. A variable must contain a type and name. Names can be alphanumeric and can optionally contain the underscore character('_'). Names are also case sensitive. Reserved keywords cannot be used as names (see Keywords section for list)

3.1.1 Basic Data Types

The basic data types are similar to their C equivalents.

int As in C, an **integer** represents a 4-byte (32 bit) signed (two's complement) integer. It can take on values ranging from -2,147,483,648 to 2,147,483,647. If these values are exceeded, the int will overflow; this is undefined behavior.
ints cannot be assigned an initial value when they are declared, these statements must be line-separated. The value of an int is undefined until it is assigned a value.

```
int a;
int a = 2;
```

Integers are closed under addition, multiplication, subtraction, division, power and modulus. For binary operations of an integer and a different number (gem or lattice), refer to the respective sections of this manual.

uint An **unsigned integer** is the unsigned equivalent of an integer. Values can range from 0 to 4,294,967,295. The same rules for an int regarding assignment and overflow apply to an unsigned int.

```
uint b;
uint b = 3;
```

char A **character** represents a 1-byte ASCII character. chars follow the naming convention delineated previously. Additionally, the value of the char is generally given in single quotes. However, a positive number can be assigned to a char provided that it is between 0 and 255 (and vice versa).

```
char a;
a = 'a';
char b = 97;
a == b;     // true
```

void A **void** type is the only basic data type that can't be assigned to. It exists solely to indicate when a function will return nothing.

bool A **boolean** represents a value, either True or False. Booleans and integers, gems and lattices are interchanegable, wherein any nonzero value assigned to a bool would contain the value True, and any zero, False.

```
bool c = True;
int d = 1;
c == d;     // true
```

3.1.2 Cryptographic Types

gem A **gem** is the foundation of this crystalline language. Values range from 0 to the modulus-1. It must be initialized with mutable number and an immutable modulus. Numbers larger than the modulus will be treated as that number modulus the initial modulo. For example,

```
gem g = (3, 7);          // initialize gem with integer 3 and modulus 7
printf(g);               // returns 3
gem h = (10, 4);
printf(h);               // returns 2
```

Gems can be initialized in a variety of ways, namely by any pairwise combination of a number type in Cryptal – integer, gem or lattice. This includes (int, int), (int, lattice), (int, gem), (gem, int), (gem, gem), (gem, lattice), (lattice, int), (lattice, gem), (lattice, lattice). If a gem plays part in the instantiation of another gem, only the value of the gem itself (which will naturally be less than its modulus) will be taken into account.

Gems are closed under the 5 basic arithmetic operations - addition, subtraction, multiplication, division and power - provided that they are of the same moduli. Arithmetic operations of addition, subtraction, multiplication and division with other gems of different moduli will throw an error. However, operations with a gem and a different numeric type such as an integer or a lattice will promote the integer or lattice to a gem.

```
gem g = (4, 7);
gem h = (10, 4)
gem i = g+h;                    // error
gem j = (g, 3);                 // j holds 1
gem k = (3, g);                 // k holds 3
```

Comparisons between gems and other number types (integer, gem and lattices) look at the value of the gems when comparing. Gems are the most restrictive type, meaning that an operation between gems and other data types will result in a gem unless it's a comparison in which it will yield a boolean or the gem is the exponent in an exponential operator.

The modulus must be a a nonnegative number. If either zero or negative, the behavior is undefined.

lat A **lattice** is big integer apt for handling the manipulation of large prime numbers, which is essential for the current cryptographic system, where it banks on the product of large prime numbers computationally infeasible to determine. Lattices operate the same way as integers would, encompassing the latter. While integers can be assigned to lattices, lattices cannot be assigned to integers.

```
lat l;
lat l = "123456789012345";
```

Lattices are closed under the 5 basic arithmetic operations, as well as promotes any integer to a lattice if a a binary operation is enacted upon the two types. When lattices are operated on with gems, the result will be a gem except for the condition in which the exponent is a gem, in which case it will return a lattice.

3.1.3 Grouping

Cryptal supports both pointers and arrays for grouping similar data values similar to how they are used in C.

pointer **Pointers** represent a memory address using 8 bytes. Pointers are generally referred to by the type of the data that the programmer would like them to reference. They can be incremented or decremented to move them to the next or previous block of memory (blocks are grouped based on the pointer's type). Dereferencing a pointer (*) will return the value at the memory address it points to. Similarly, referencing a variable (&) will return a pointer to it.

```
char a = 'a';
char *foo = &a;          //foo now points to a
*foo = 'A'               //a now stores 'A'
```

String A **string** is a concatenation of characters that allows for the storage of texts, and messages. Characters can be stored in Strings as well.

```
String b;
String b = "cryptal";
String c = 'c';
```

3.2 Operators

3.2.1 Unary Operators

These work as in C.

*** expression** The **derefernce** operator is used to dereference a pointer. This returns the object which the pointer points to. (See Data Types: Basic Data Types: pointer for more information).

− expression The **numerical negation operator** returns the additive inverse of a number, e.g. an integer, gem or lattice.

! expression The **logical negation operator** returns 0 if expression is non-zero, non-null and 1 otherwise. This special operation also returns the logical negation of a gem, which is the modular inverse of the gem, returning a gem of the same modulus.

```
gem g = (3, 5);
gem h = !g;                    // h contains (2, 5), which is the modular
                               // inverse of 3 in mod 5
```

If the number stored in the gem and the modulus are not coprime, the behavior is undefined (i.e. in order for this operator to work properly, the user must ensure that the modular inverse is obtainable, which is not the case if the gcd of the number and modulus is greater than 1).

3.2.2 Mathematical Operators

These work similar to C, with special behavior in a modular environment.

expression ∗ expression The **multiplication operator** performs multiplication between the left and right expressions, where expression must be of type int, gem or lattice. All three data types are closed under this operation. If an integer is multiplied with a gem, then the result will be a gem. If an integer is multiplied with a lattice, the result will be a lattice. If a gem is muliplied with a lattice, the result will be a gem. If two gems are of different moduli, it will throw an error.

expression / expression The **division operator** performs division between the left and right expressions, where expression must be of type int, gem or lattice. All three data types are closed under this operation. If an integer is divided a gem, then the result will be a gem (and vice versa). If an integer is divided a lattice, the result will be a lattice (and vice versa). If a gem is divided by a lattice, the result will be a gem (and vice versa). If two gems are of different moduli, it will throw an error.

expression + expression The **addition operator** performs the addition between both expressions, where expression must be of type int, gem or lattice. All three data types are closed under this operation. If an integer is added to a gem, then the result will be a gem. If an integer is added to a lattice, the result will be a lattice. If a gem is added to a lattice, the result will be a gem. If two gems are of different moduli, it will throw an error.

expression − expression The **subtraction operator** performs the subtraction between both expressions, where expression must be of type int, gem or lattice. All three data types are closed under this operation. If an integer is subtracted with a gem (and vice versa), then the result will be a gem. If an integer is subtracted with a lattice, the result will be a lattice (and vice versa). If a gem is subtracted with a lattice, the result will be a gem (and vice versa). If two gems are of different moduli, it will throw an error.

expression ∗∗ expression The **exponential operator** represents an exponentiation which requires the expressions be of type int, gem or lattice. All three data types are closed under this operation. If an integer is added to a gem, then the result will be a gem. If an integer is added to a lattice, the result will be a lattice. If a gem is added to a lattice, the result will be a gem. If two gems are of different moduli, it will throw an error.

expression % expression The **modulus operator** represents the modulus between two numbers, which requires the expressions be of type int, gem, or lattice. This is an operation in which the type of the right hand side may be of any numeric type, however the left hand side must be of either an integer or lattice. Gems are by nature moduli themselves so attempts to perform the modulus operation will throw an error.

3.2.3 Relational Operators

These work as in C. The values of the left hand side and the right hand sides are compared, and will return a boolean (True or False) depending on whether the relational operator evaluates to True or False.

expression > expression The **greater than** operator checks to see if the value of the left hand side is greater than the value of the right hand side.

expression < expression The **less than** operator checks to see if the value of the left hand side is less than the value of the right hand side.

expression >= expression The **greater than or equal to** operator checks to see if the value of the left hand side is greater or equal to than the value of the right hand side.

expression <= expression The **less than or equal to** operator checks to see if the value of the left hand side is less than or equal to the value of the right hand side.

expression == expression The **equality** operator checks to see if the value of the left hand side is equal to the value of the right hand side.

expression != expression The **nonequality** operator checks to see if the value of the left hand side is not equal to the value of the right hand side.

These perform comparisons between the left and right expression if either the expressions are of the same types or the expressions are assignable to each other (see the Assignment Operators section for the full list of assignment types). If expression is an int, gem, or lattice, the numerical value is used as a comparison. Otherwise, the value stored at the memory location to which the expression evaluates to is used. These operators return 0 or 1, the equivalent of False and True respsectively, depending on the result of the comparison.

```
int a = 5;
gem b = (a, 3); /* 2 */
gem c = (6, 4); /* 2 */

a > b;          /* 0 */
b == c;         /* 1 */
```

3.2.4 Logical Operators

expression && expression Evaluates to 1 if both expressions evaluate to 1 and 0 otherwise.

expression || expression Evaluates to 1 if one or both expressions evaluate to 1 and 0 otherwise.

3.2.5 Assignment Operators

lvalue = expression The expression is evaluated and stored in the lvalue. All variables of the same type can be assigned to one another. In this case, it will overwrite any value that it was previously containing. If the assignment regards two different variables, only the following cases will be allowed; else it will throw an error:

1. int ← gem
2. int ← lattice
3. lattice ← int
4. lattice ← string
5. lattice ← gem
6. string ← char
7. int ← char
8. char ← int
9. bool ← int

3.3 Control Flow

Here we specify the order in which computation is performed.

3.3.1 Statements and Blocks

Any expression can become a statement if it is followed by a semicolon. In other words, the semicolon is a statement terminator. Examples are:

```
score = 100;
grade = "A+";
```

Additionally, curly braces ({...}) define compound statements or **blocks**. The latter are groups of declarations and statements, which will be syntactically treated as a single statement. We will see their use bellow in the fundamental control flow schemas.

3.3.2 If-Else

To execute statements conditionally, we use the keywords, **if** and **else**. Their syntax follows the rule:

```
if (expression)
        statement1
else
        statement2
```

The *else* part of the syntax is optional. First the expression is evaluated. If it is true, meaning that it evaluates to a non-zero value, then statement1 is executed. Otherwise, if the expression evaluates to zero (false) and an else clause is defined, statement2 is executed. If-Else statements can nest, and because else is optional, an else part is associated with the closest previous else-less if, just like in C. To associate an else to an if further away, blocks can be used, as in the example below:

```
if (x == y) {
        if (x > 0)
                a = 0;
}
else
        a = 1;
```

3.3.3 While Loop

The syntax of a while loop is:

```
while (expression)
        statement
```

First the *expression* is evaluated and if it is true the *statement* is executed. Then we repeat until the *expression* evaluates to false (0).

3.3.4 For Loop

The syntax for a for statement is:

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

Which is equivalent to:

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

The equivalence is not exact for the use of the continue statement, which we will see later. Any of the expression1, expression2 or expression3 may be missing but not the semicolons. Last, the **for** loop:

```
for (;;)
```

is equivalent to the while loop:

```
while (true)
```

A *continue* statement will skip the rest of the statements of the body of the current iteration. Note, that for for loops specifically, expression3 will still be evaluated before the next iteration of the loop.

3.4 Program Structure

A Cryptal program is composed of global variables and functions that can communicate to each other through those global variables as well as parameters passed to them and/or values returned by them.
Global variables consist of a type and a name. Functions consist of a return type, a name, a list of parameter types and names, and a body. Functions will have local variables that consist of a type and a name.
The only function that is required in a Cryptal program is the **main** function that returns an int. This is the entry point of the program. Our language does not require forward-declaration, so it is not necessary to declare a function before it is used if it is defined below. All other functions besides **main** may have any of our data types as a return type. Functions that return void will simply return a 0.
Duplicate variable and function names are not allowed and will return a compile time error.

3.4.1 Examples

```
int main(arg1, arg2) { .... } // returns int
gem func(arg1, arg2) { .... } //returns gem
void func(arg1, arg2) { .... } // returns void
```

3.5 Built-in Functions

- print_gem(gem g): Prints value of gem g to stdout.
- print_lat(lat l): Prints value of lattice l to stdout.
- printf(char *format_str, ...): Prints formatted string to stdout.
- hash_md5(char *str): Calculates md5 of given string str.

4 Project Plan

4.1 Motivation

The motivation for our project came from our combined interest in the fields of security and cryptography. Our group members have previously taken classes focusing on Security Engineering, Malware Analysis, and Cryptography from both a mathematical and computer science viewpoint, and we saw the need for a language that would facilitate the implementation of encryption schemes.

4.2 Early Stages: Planning

Form the early stages of our project we set milestones for our project which we discussed with our TA, Heather Preslier.

1. Project Proposal
2. Language Reference Manual
3. Implement basic features and built-in data types.
4. get Hello World to work
5. Implement expressions and statements
6. Final stage testing and wrap up

4.3 Work Flow

We decided to use git as our version control system. We created a private repository in GitHub where we pushed our commits. Team communication was done over Messenger, as well as through weekly meetings on Wednesdays after class and on Fridays, following our meetings with our TA.

4.4 Testing

Testing was performed at each step following implementation of basic features. Our first test was to successfully compile and run Hello World. We then added appropriate tests for each expression and built-in type as they were added, as detailed in the test plan in Section 7.

4.5 Division of Labour

Our roles are slightly modified from what was decided on the project proposal. Due to familiarity with one file over another and division of work, below are the roles each team member executed:

- Sammy Tbeile (System Architect): Devised integration with OpenSSL. Did significant work on code generation and architected gems and their behaviors.
- Jaewan Bahk (Language Guru/Tester): Static semantic checking. Devised language grammar including primitive types such as gem and lattices, their interaction with preexisting types and concepts of modular inverses. Wrote test suite for edge cases, and most importantly, came up with the language name and logo.
- Michail Oikonomou (System Architect/Tester): Worked primarily on codegen on the implementation of expressions and statements and built-in functions and on testing for continuous integration. Focused on implementation of operations between mixed cryptographic types and performed thorough testing of such operations.
- Carolina Almirola (Manager): Worked on semantic checking for mathematical expressions and statements and on language documentation and final report. Implemented built-in hashing function for language and contributed to BN integration in to codegen and testing for built-in functions and demos.
- Rahul Kapur (Tester): Worked on test suite. Developed failure edge cases through test programs. Wrote Diffie-Hellman demo.

5 Language Evolution

Our language stems from the desire to make modular cryptography clear and intuitive. As with any, we started out with a basic compiler that compiled parts of C-like language. We decided there were certain operations in modern languages did not offer which could see some benefit in the construction of those variables and operations.

We started with hello world – for us, this meant literally printing the string, "Hello world". This ensured we had basic functionality up and running, such as interpreting the strings (as well as support for the string type) and crafting a draft of our grammar.

The next phase involved thinking of what kind of operations do we really want to support, thinking of specific cryptographic protocols we wanted to implement, including the greatest common divisor via the Extended Euclidean algorithm, the RSA Key Exchage, ElGamal as well as the Diffie-Hellman key exchange. The operations between our built in types such as gems and lattices were carefully thought through, the team having discussed it many times. After we were set on what behaviors every instance and interaction any type would have with another, our language was born.

6 Language Architecture

Our combined architecture is composed of the Parser, Scanner, and AST which together create an AST for a given .crp program. The main program that puts all the pieces together and handles command line flags is cryptal.ml. Lastly, our language's interactions with openSSL can be seen in the two files crypto_arith.c and crypto_types.c.

6.1 Parser, Scanner, and AST

Parser handles the syntax of our language which in turn is used in conjunction with the scanner to tokenize our program for the AST. In the parser, we define language semantics for our native types along with associativity and precedence for our operators, including those applied to gem and lat. This is where the bulk of our syntactical definition of Cryptal lies. As mentioned the scanner is used in conjunction with the parser in order to tokenize keywords of a program file as they are read from stdin. This includes tokens for all of the keywords that our language accepts. Finally, these are used together along with the AST file to a produce a program AST.

6.2 Semantic Check

Semantic checker deals with the verification of types and return types from the abstract syntax tree (making it semantically verified). Our semantic file contains among other things handling for custom print operations with our native types as well handling assignment for our native-types and their inter-operability with C-types.

6.3 Codegen

Codegen handles the bulk of the functionality that makes our code inter-operable with native C-types. This includes building functions for the comparison of lats and ints as well as many arithmetic features for gems, lats, and ints. It also deals with the inner workings of our native gem type by handling the allocation and manipulation of pointers for the values contained in the structure.

6.4 crypto_arith.c, crypto_types.h

The crypto_types.h file defines a struct of type gem with two void * to denote the tuple of values that will fill the gem. The crypto_arith.c file is an interface for our language that adds support for both operators and comparators for gem objects. This file also includes openssl/bn.h in order to deal with big number arithmetic for Cryptal's native types, gem & lat.

7 Test Plan and Scripts

Our testing can be broken down into three sections: namely types & assignment, gem & lat edge cases, and operators tests. Testing was done each time a new feature was added to the compiler, so the test suite includes tests for every operator on every combination of types (gems, integers, and lattices) as well as for built-in functions and assignment.

7.1 Automated Tests

Tests were added each time a new feature was added and the script **testall.sh** was run to ensure that all test cases still passed.

7.1.1 Naming Convention

Test files were named following the convention ¡test/fail¿-¡type name¿-¡operation¿.crp. Tests that are named beginning with "fail" are meant to fail when run, since they are mostly meant to capture cases where undefined or restricted operations are performed.

8 Conclusions

Some lessons we learned as a team were about planning and coordination and about effective communication. Additionally, we learned that implementing cryptographic-oriented types and functions correctly is very difficult, which is why we chose in the end to use openSSL's BIGNUM library for most of our operations and built-in functions.
Overall, we were able to accomplish most of the targets which we had set out to in the beginning of the semester and we were able to work together and divide tasks in a way that was effective and mostly everyone did a good amount of work and specialized in at least one file.
If we were to have to do this assignment again in the future, given all the knowledge we gained, we would most likely be able to divide up the work more appropriately from the beginning and we'd focus on writing a language were we could implement most of the operations on our own.

9 Full Code Listing

(Continued on next page)

```
==============================
 ast.ml
==============================

(* Abstract Syntax Tree and functions for printing it *)

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

type uop = Neg | Not

type typ = Int | Bool | Void | Gem | UInt | Char | Lattice | Pointer
of typ | String

type bind = typ * string

type expr =
    Literal of int
  | BoolLit of bool
  | Id of string
  | String of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Gem of expr * expr
  | Lat of expr
  | Assign of string * expr
  | Ch of char
  | Call of string * expr list
  | Noexpr
  | Null

type stmt =
    Block of (bind list * stmt list)
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt

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

type program = bind list * func_decl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
```

```
  | Sub -> "-"
  | Mult -> "*"
  | Power -> "**"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"
  | Mod -> "%"

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

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | Id(s) ->  s
  | String(s) -> s
  | Ch(c) -> String.make 1 c
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr
e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Gem(e1, e2) -> "(" ^ string_of_expr e1 ^ ", " ^ string_of_expr e2
^ ")"
  | Lat(e) -> "L " ^ string_of_expr e
  | Assign(v, e) ->v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""
  | Null -> "NULL"

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 rec string_of_typ = function
```

```
    Int -> "int"
  | Bool -> "bool"
  | Void -> "void"
  | Gem -> "gem"
  | UInt -> "uint"
  | Char -> "char"
  | String -> "string"
  | Lattice -> "lattice"
  | Pointer (_ as t) -> "pointer " ^ string_of_typ(t)

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

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

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

===== end of ast.ml =====


==============================
 codegen.ml
==============================

(* Code generation: translate takes a semantically checked AST and
produces LLVM IR

LLVM tutorial: Make sure to read the OCaml version of the tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)

module L = Llvm
module A = Ast

module StringMap = Map.Make(String)

let translate (globals, functions) =
  let context = L.global_context () in
  let the_module = L.create_module context "Cryptal"
  and i32_t  = L.i32_type  context
```

```
  and i8_t   = L.i8_type   context in
  let i1_t   = L.i1_type   context
  and void_t = L.void_type context
  and str_t = L.pointer_type i8_t in
  let lat_pointer = L.pointer_type (L.i8_type context) in
  let gem_type = L.struct_type context  [| lat_pointer ; lat_pointer
|] in
  let gem_pointer = L.pointer_type gem_type in

  let rec ltype_of_typ = function
      A.Int -> i32_t
    | A.Bool -> i1_t
    | A.Void -> void_t
    | A.Gem -> gem_type
    | A.UInt -> i32_t
    | A.Char -> i8_t
    | A.String -> str_t
    | A.Lattice -> lat_pointer
    | A.Pointer x -> L.pointer_type (ltype_of_typ x) in

  (* Declare each global variable; remember its value in a map *)
  let global_vars =
    let global_var m (t, n) =
      let init = L.const_int (ltype_of_typ t) 0
      in StringMap.add n ((L.define_global n init the_module), (t,0))
m in
    List.fold_left global_var StringMap.empty globals in

  (* Declare printf(), which the print built-in function will call *)
  let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t
|] in
  let printf_func = L.declare_function "printf" printf_t the_module in

  let gem_add_t = L.function_type gem_type [| gem_pointer ;
gem_pointer |] in
  let gem_add_func = L.declare_function "gem_add_func" gem_add_t
the_module in

  let gem_sub_t = L.function_type gem_type [| gem_pointer ;
gem_pointer |] in
  let gem_sub_func = L.declare_function "gem_sub_func" gem_sub_t
the_module in

  let gem_mult_t = L.function_type gem_type [| gem_pointer ;
gem_pointer |] in
  let gem_mult_func = L.declare_function "gem_mult_func" gem_mult_t
the_module in

(* use lat pointers as a general 8 byte pointer *)
  let gem_mod_create_t = L.function_type lat_pointer [| i32_t ; i32_t
|] in
  let gem_mod_create_func = L.declare_function "gem_mod_create_func"
gem_mod_create_t the_module in

  let gem_int_create_t = L.function_type lat_pointer [| gem_pointer ;
```

```
i32_t |] in
  let gem_int_create_func = L.declare_function "gem_int_create_func"
gem_int_create_t the_module in

  let int_gem_create_t = L.function_type lat_pointer [|i32_t ;
gem_pointer |] in
  let int_gem_create_func = L.declare_function "int_gem_create_func"
int_gem_create_t the_module in

  let gem_mod_create_latint_t = L.function_type lat_pointer [|
lat_pointer ; i32_t |] in
  let gem_mod_create_latint_func = L.declare_function
"gem_mod_create_latint_func" gem_mod_create_latint_t the_module in

  let gem_mod_create_intlat_t = L.function_type lat_pointer [| i32_t ;
lat_pointer |] in
  let gem_mod_create_intlat_func = L.declare_function
"gem_mod_create_intlat_func" gem_mod_create_intlat_t the_module in

  let gem_mod_create_latlat_t = L.function_type lat_pointer [|
lat_pointer ; lat_pointer |] in
  let gem_mod_create_latlat_func = L.declare_function
"gem_mod_create_latlat_func" gem_mod_create_latlat_t the_module in

  let print_gem_t = L.function_type lat_pointer [|gem_type |] in
  let print_gem_func = L.declare_function "print_gem_func" print_gem_t
the_module in

  let int_to_bn_t = L.function_type lat_pointer [| i32_t |] in
  let int_to_bn_func = L.declare_function "int_to_bn_func" int_to_bn_t
the_module in
  (*We may not want to do modular division... *)
  let gem_div_t = L.function_type gem_type [| gem_pointer ;
gem_pointer |] in
  let gem_div_func = L.declare_function "gem_div_func" gem_div_t
the_module in

  let gem_pow_t = L.function_type gem_type [| gem_pointer ;
gem_pointer |] in
  let gem_pow_func = L.declare_function "gem_pow_func" gem_pow_t
the_module in

  let gem_eq_t = L.function_type i1_t [| gem_pointer ; gem_pointer |]
in
  let gem_eq_func = L.declare_function "gem_eq_func" gem_eq_t
the_module in

  let gem_neq_t = L.function_type i1_t [| gem_pointer ; gem_pointer |]
in
  let gem_neq_func = L.declare_function "gem_neq_func" gem_neq_t
the_module in

  let gem_le_t = L.function_type i1_t [|gem_pointer ; gem_pointer |]
in
  let gem_le_func = L.declare_function "gem_le_func" gem_le_t
```

```
the_module in

  let gem_leq_t = L.function_type i1_t [|gem_pointer ; gem_pointer |]
in
  let gem_leq_func = L.declare_function "gem_leq_func" gem_leq_t
the_module in

  let gem_ge_t = L.function_type i1_t [|gem_pointer ; gem_pointer |]
in
  let gem_ge_func = L.declare_function "gem_ge_func" gem_ge_t
the_module in

  let gem_geq_t = L.function_type i1_t [| gem_pointer ; gem_pointer |]
in
  let gem_geq_func = L.declare_function "gem_geq_func" gem_geq_t
the_module in

  let gem_inverse_t = L.function_type gem_type [| gem_pointer |] in
  let gem_inverse_func = L.declare_function "gem_inverse_func"
gem_inverse_t the_module in


  (* int,gem ops *)
  let int_gem_add_t = L.function_type gem_type [| lat_pointer ;
gem_pointer |] in
  let int_gem_add_func = L.declare_function "int_gem_add_func"
int_gem_add_t the_module in

  let int_gem_sub_t = L.function_type gem_type [| lat_pointer ;
gem_pointer |] in
  let int_gem_sub_func = L.declare_function "int_gem_sub_func"
int_gem_sub_t the_module in

  let int_gem_mult_t = L.function_type gem_type [| lat_pointer ;
gem_pointer |] in
  let int_gem_mult_func = L.declare_function "int_gem_mult_func"
int_gem_mult_t the_module in

  let int_gem_div_t = L.function_type gem_type [| lat_pointer ;
gem_pointer |] in
  let int_gem_div_func = L.declare_function "int_gem_div_func"
int_gem_div_t the_module in

  let int_gem_pow_t = L.function_type gem_type [| lat_pointer ;
gem_pointer |] in
  let int_gem_pow_func = L.declare_function "int_gem_pow_func"
int_gem_pow_t the_module in

  (*int, gem comparisons *)
  let int_gem_eq_t = L.function_type i1_t [| lat_pointer ; gem_pointer
|] in
  let int_gem_eq_func = L.declare_function "int_gem_eq_func"
int_gem_eq_t the_module in

  let int_gem_neq_t = L.function_type i1_t [| lat_pointer ;
```

```
gem_pointer |] in
  let int_gem_neq_func = L.declare_function "int_gem_neq_func"
int_gem_neq_t the_module in

  let int_gem_less_t = L.function_type i1_t [| lat_pointer ;
gem_pointer |] in
  let int_gem_less_func = L.declare_function "int_gem_less_func"
int_gem_less_t the_module in

  let int_gem_greater_t = L.function_type i1_t [| lat_pointer ;
gem_pointer |] in
  let int_gem_greater_func = L.declare_function "int_gem_greater_func"
int_gem_greater_t the_module in

  let int_gem_leq_t = L.function_type i1_t [| lat_pointer ;
gem_pointer |] in
  let int_gem_leq_func = L.declare_function "int_gem_leq_func"
int_gem_leq_t the_module in

  let int_gem_geq_t = L.function_type i1_t [| lat_pointer ;
gem_pointer |] in
  let int_gem_geq_func = L.declare_function "int_gem_geq_func"
int_gem_geq_t the_module in

  let gem_get_t = L.function_type lat_pointer [| gem_type ; i32_t |]
in
  let gem_get_func = L.declare_function "gem_get_func" gem_get_t
the_module in

  let gem_to_lat_t = L.function_type lat_pointer [| gem_pointer |] in
  let gem_to_lat_func = L.declare_function "gem_to_lat_func"
gem_to_lat_t the_module in

  (* Lattice ops *)
  let lat_str_create_t = L.function_type lat_pointer [| str_t |] in
  let lat_str_create_func = L.declare_function "lat_str_create_func"
lat_str_create_t the_module in

  let print_lat_t = L.function_type lat_pointer [| lat_pointer |] in
  let print_lat_func = L.declare_function "print_lat_func" print_lat_t
the_module in

  (* Lat, Lat ops *)
  let lat_add_t = L.function_type lat_pointer [| lat_pointer ;
lat_pointer |] in
  let lat_add_func = L.declare_function "lat_add_func" lat_add_t
the_module in

  let lat_sub_t = L.function_type lat_pointer [| lat_pointer ;
lat_pointer |] in
  let lat_sub_func = L.declare_function "lat_sub_func" lat_sub_t
the_module in

  let lat_mult_t = L.function_type lat_pointer [| lat_pointer ;
lat_pointer |] in
```

```
  let lat_mult_func = L.declare_function "lat_mult_func" lat_mult_t
the_module in

  let lat_div_t = L.function_type lat_pointer [| lat_pointer ;
lat_pointer |] in
  let lat_div_func = L.declare_function "lat_div_func" lat_div_t
the_module in

  let lat_pow_t = L.function_type lat_pointer [| lat_pointer ;
lat_pointer |] in
  let lat_pow_func = L.declare_function "lat_pow_func" lat_pow_t
the_module in

  let lat_mod_t = L.function_type lat_pointer [| lat_pointer ;
lat_pointer |] in
  let lat_mod_func = L.declare_function "lat_mod_func" lat_mod_t
the_module in

  let lat_eq_t = L.function_type i1_t [| lat_pointer ; lat_pointer |]
in
  let lat_eq_func = L.declare_function "lat_eq_func" lat_eq_t
the_module in

  let lat_neq_t = L.function_type i1_t [| lat_pointer ; lat_pointer |]
in
  let lat_neq_func = L.declare_function "lat_neq_func" lat_neq_t
the_module in

  let lat_le_t = L.function_type i1_t [|lat_pointer ; lat_pointer |]
in
  let lat_le_func = L.declare_function "lat_le_func" lat_le_t
the_module in

  let lat_leq_t = L.function_type i1_t [|lat_pointer ; lat_pointer |]
in
  let lat_leq_func = L.declare_function "lat_leq_func" lat_leq_t
the_module in

  let lat_ge_t = L.function_type i1_t [|lat_pointer ; lat_pointer |]
in
  let lat_ge_func = L.declare_function "lat_ge_func" lat_ge_t
the_module in

  let lat_geq_t = L.function_type i1_t [| lat_pointer ; lat_pointer |]
in
  let lat_geq_func = L.declare_function "lat_geq_func" lat_geq_t
the_module in

  let hash_md5_t = L.function_type str_t [|str_t|] in
  let hash_md5_func = L.declare_function "hash_md5_func" hash_md5_t
the_module in


  (* Declare the built-in printbig() function *)
  let printbig_t = L.function_type i32_t [| i32_t |] in
```

```
  let printbig_func = L.declare_function "printbig" printbig_t
the_module in

  (* Declare the built-in print_string() function *)
  let print_string_t = L.function_type str_t [| str_t |] in
  let print_string_func = L.declare_function "print_string"
print_string_t the_module in

  (* Define each function (arguments and return type) so we can call
it *)
  let function_decls =
    let function_decl m fdecl =
      let name = fdecl.A.fname
      and formal_types =
     Array.of_list (List.map (fun (t,_) -> ltype_of_typ t)
fdecl.A.formals)
          in let ftype = L.function_type (ltype_of_typ fdecl.A.typ)
formal_types in
          StringMap.add name (L.define_function name ftype the_module,
fdecl) m in
        List.fold_left function_decl StringMap.empty functions in

  (* Fill in the body of the given function *)
  let build_function_body fdecl =
    let (the_function, _) = StringMap.find fdecl.A.fname
function_decls in
    let builder = L.builder_at_end context (L.entry_block
the_function) in
    let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder
    and string_format_str = L.build_global_stringptr "%s\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 lookup n tree = try StringMap.find n tree
     with Not_found -> StringMap.find n global_vars
     in


    (* Return the value for a variable or formal argument *)

    (* Construct code for an expression; return its value *)
    let rec expr  tree builder  = function
           A.Literal i -> (L.const_int i32_t i, (A.Int,0))
      | A.BoolLit b -> (L.const_int i1_t (if b then 1 else 0), (A.Int,
0))
      | A.Noexpr -> (L.const_int i32_t 0, (A.Void, 0))
      | A.String s -> (L.build_global_stringptr s "fmt" builder,
(A.Pointer(A.Char), 0))
      (* | A.Null ->  *)
      (* | A.Ch -> *)
      | A.Id s ->
      let binding = lookup s tree in
```

```
        (L.build_load (fst binding) s builder, (fst (snd binding), 1))

      (* Assign Value to Gem *)
      (* | A.Gem (e1, e2) ->
          (let newE1 = expr tree builder e1 and newE2 = expr tree
buider e2 in
          let pointer  = L.build_alloca gem_type "e1e2" builder,
(A.Gem, 0))


      *)
      (* | A.Lattice () *)
      | A.Gem (e1, e2) ->
        let (e1', (t1, _)) = expr tree builder e1
        and (e2', (t2, _)) = expr tree builder e2 in
        let struct_gem = L.undef gem_type in (match  (t1,t2) with
          (A.Int, A.Int) ->
            let modded_val = L.build_call gem_mod_create_func [|e1';
e2' |] "gem_mod_res" builder in
            let mod_val = L.build_call int_to_bn_func [|e2' |]
"int_to_bn_res" builder in
            let struct_gem_final = L.build_insertvalue struct_gem
(modded_val) 0 "gm" builder in
            (L.build_insertvalue struct_gem_final (mod_val) 1 "gm2"
builder)
        | (A.Gem, A.Int) ->
          let pointer1 = L.build_alloca gem_type "e1" builder in
          let _ = L.build_store e1' pointer1 builder in
          let modded_val = L.build_call gem_int_create_func [|
pointer1; e2' |] "gem_int_res" builder in
          let mod_val = L.build_call int_to_bn_func [| e2' |]
"int_to_bn_res" builder in
          let struct_gem_final = L.build_insertvalue struct_gem
(modded_val) 0 "gm" builder in
            (L.build_insertvalue struct_gem_final (mod_val) 1 "gm2"
builder)
        | (A.Int, A.Gem) ->
          let pointer2 = L.build_alloca gem_type "e2" builder in
          let _ = L.build_store e2' pointer2 builder in
          let modded_val = L.build_call int_gem_create_func [| e1' ;
pointer2 |] "int_gem_Res" builder in
          let mod_val = L.build_call int_to_bn_func [| e2' |]
"int_to_bn_res" builder in
          let struct_gem_final = L.build_insertvalue struct_gem
(modded_val) 0 "gm" builder in
            (L.build_insertvalue struct_gem_final (mod_val) 1 "gm2"
builder)
        | (A.Int, A.Lattice) ->
          let modded_val = L.build_call gem_mod_create_intlat_func
[| e1'; e2' |] "gem_mod_create_intlat_res" builder in
          let mod_val = e2' in
          let struct_gem_final = L.build_insertvalue struct_gem
(modded_val) 0 "gm" builder in
          (L.build_insertvalue struct_gem_final (mod_val) 1 "gm2"
builder)
```

```
          | (A.Lattice, A.Int) ->
            let modded_val = L.build_call gem_mod_create_latint_func
[| e1'; e2' |] "gem_mod_create_latint_res" builder in
            let mod_val = L.build_call int_to_bn_func [|e2' |]
"int_to_bn_res" builder in
            let struct_gem_final = L.build_insertvalue struct_gem
(modded_val) 0 "gm" builder in
            (L.build_insertvalue struct_gem_final (mod_val) 1 "gm2"
builder)
          | (A.Lattice, A.Lattice) ->
            let modded_val = L.build_call gem_mod_create_latlat_func
[| e1'; e2' |] "gem_mod_create_latlat_res" builder in
            let mod_val = e2' in
            let struct_gem_final = L.build_insertvalue struct_gem
(modded_val) 0 "gm" builder in
            (L.build_insertvalue struct_gem_final (mod_val) 1 "gm2"
builder)
        ), (A.Gem, 1)
      (* Lattice literal - not used
      | A.Lat e ->
        let (e', (t, _)) = expr tree builder e in
        (match t with
            A.Pointer(A.Char) -> L.build_call lat_str_create_func [|
e' |] "lat_str_res" builder
          | A.Int -> L.build_call int_to_bn_func [| e' |]
"int_to_bn_res" builder
          | ty -> raise(Failure("cannot initialize lattice with " ^
A.string_of_typ ty))
        ), (A.Lattice, 1)
      *)
      | A.Binop (e1, op, e2) ->
      (* (print_string ("e1 is of type: " ^ (A.string_of_typ e1))); *)
        let (e1', (t1, leaf1)) = expr tree builder e1
        and  (e2', (t2, leaf2)) = expr tree builder e2 in
          (match (t1, t2) with
          | (A.Int, A.Int) ->
             ((match op with
               A.Add     -> L.build_add
             | A.Sub     -> L.build_sub
             | A.Mult    -> L.build_mul
              | A.Div     -> L.build_sdiv
             | A.And     -> L.build_and
             | A.Or      -> L.build_or
             | A.Equal   -> L.build_icmp L.Icmp.Eq
             | A.Neq     -> L.build_icmp L.Icmp.Ne
             | A.Less    -> L.build_icmp L.Icmp.Slt
             | A.Leq     -> L.build_icmp L.Icmp.Sle
             | A.Greater -> L.build_icmp L.Icmp.Sgt
             | A.Geq     -> L.build_icmp L.Icmp.Sge
              (* | A.Mod     -> L.  *)
              (* | A.Power ->  *)
             ) e1' e2' "tmp" builder, (A.Int, 0))
          | (A.Int, A.Gem) | (A.Lattice, A.Gem) ->
            let arg = if t1 = A.Int then (L.build_call int_to_bn_func
[| e1' |] "int_to_bn_res1" builder) else e1' in
```

```
            let pointer1 = L.build_alloca gem_type "e2" builder in
            let _ = L.build_store e2' pointer1 builder in
            (match op with
              A.Add ->
                (L.build_call int_gem_add_func [|arg; pointer1 |]
"gem_int_add_res" builder, (A.Gem, 0))
            | A.Sub ->
                (L.build_call int_gem_sub_func [|arg ; pointer1 |]
"gem_int_sub_res" builder, (A.Gem, 0))
            | A.Mult ->
                (L.build_call int_gem_mult_func [|arg ; pointer1 |]
"gem_int_mult_res" builder, (A.Gem, 0))
            | A.Div ->
                (L.build_call int_gem_div_func [|arg ; pointer1 |]
"gem_int_div_res" builder, (A.Gem, 0))
            | A.Power ->
                (L.build_call int_gem_pow_func [|arg ; pointer1 |]
"gem_int_pow_res" builder, (A.Gem, 0))
            | A.Equal ->
                (L.build_call int_gem_eq_func [|arg ; pointer1 |]
"gem_int_add_res" builder, (A.Int, 0))
            | A.Neq ->
                (L.build_call int_gem_neq_func [|arg ; pointer1 |]
"gem_int_neq_res" builder, (A.Int, 0))
            | A.Less ->
                (L.build_call int_gem_less_func [|arg ; pointer1 |]
"gem_int_less_res" builder, (A.Int, 0))
            | A.Leq ->
                (L.build_call int_gem_leq_func [|arg ; pointer1 |]
"gem_int_leq_res" builder, (A.Int, 0))
            | A.Greater ->
                (L.build_call int_gem_greater_func [|arg ; pointer1 |]
"gem_int_greater_res" builder, (A.Int, 0))
            | A.Geq ->
                (L.build_call int_gem_geq_func [|arg ; pointer1 |]
"gem_int_geq_res" builder, (A.Int, 0))
            )
          | (A.Gem, A.Int) | (A.Gem, A.Lattice) ->
            let arg = if t2 = A.Int then (L.build_call int_to_bn_func
[| e2' |] "int_to_bn_res2" builder) else e2' in
            let pointer1 = L.build_alloca gem_type "e1" builder in
            let _ = L.build_store  e1' pointer1 builder in
            (match op with
              A.Add ->
                (L.build_call int_gem_add_func [|arg ; pointer1 |]
"gem_int_add_res" builder, (A.Gem, 0))
            | A.Sub ->
                (L.build_call int_gem_sub_func [|arg ; pointer1 |]
"gem_int_sub_res" builder, (A.Gem, 0))
            | A.Mult ->
                (L.build_call int_gem_mult_func [|arg ; pointer1 |]
"gem_int_mult_res" builder, (A.Gem, 0))
            | A.Div ->
                (L.build_call int_gem_div_func [|arg ; pointer1 |]
"gem_int_div_res" builder, (A.Gem, 0))
```

```
              | A.Power ->
                (L.build_call int_gem_pow_func [|arg ; pointer1 |]
"gem_int_pow_res" builder, (A.Gem, 0))
              | A.Equal ->
                (L.build_call int_gem_eq_func [|arg ; pointer1 |]
"gem_int_add_res" builder, (A.Int, 0))
              | A.Neq ->
                (L.build_call int_gem_neq_func [|arg ; pointer1 |]
"gem_int_neq_res" builder, (A.Int, 0))
              | A.Less ->
                (L.build_call int_gem_less_func [|arg ; pointer1 |]
"gem_int_less_res" builder, (A.Int, 0))
              | A.Leq ->
                (L.build_call int_gem_leq_func [|arg ; pointer1 |]
"gem_int_leq_res" builder, (A.Int, 0))
              | A.Greater ->
                (L.build_call int_gem_greater_func [|arg ; pointer1 |]
"gem_int_greater_res" builder, (A.Int, 0))
              | A.Geq ->
                (L.build_call int_gem_geq_func [|arg ; pointer1 |]
"gem_int_geq_res" builder, (A.Int, 0))
              )

          | (A.Gem, A.Gem) ->
            let pointer1  = L.build_alloca gem_type "e1" builder and
            pointer2 = L.build_alloca gem_type "e2" builder in
            let _ = L.build_store e1' pointer1 builder and
            _ = L.build_store e2' pointer2 builder in
            (match op with
              A.Add ->
               (L.build_call gem_add_func [|pointer1 ; pointer2 |]
"gem_add_res" builder, (A.Gem, 0))
              | A.Sub ->
                (L.build_call gem_sub_func [| pointer1 ; pointer2 |]
"gem_sub_res" builder , (A.Gem, 0))
              | A.Mult ->
                (L.build_call gem_mult_func [| pointer1 ; pointer2 |]
"gem_mult_res" builder, (A.Gem, 0))
              | A.Div ->
                (L.build_call gem_div_func [| pointer1 ; pointer2 |]
"gem_div_res" builder, (A.Gem, 0))
              | A.Power ->
                (L.build_call gem_pow_func [| pointer1 ; pointer2 |]
"gem_pow_res" builder, (A.Gem, 0))
              | A.Equal ->
                (L.build_call gem_eq_func [| pointer1 ; pointer2 |]
"gem_eq_res" builder, (A.Int, 0))
              | A.Neq ->
               (L.build_call gem_neq_func [|pointer1 ; pointer2 |]
"gem_neq_res" builder, (A.Int, 0))
              | A.Less ->
               (L.build_call gem_le_func [|pointer1 ; pointer2 |]
"gem_le_res" builder, (A.Int,0))
              | A.Leq ->
               (L.build_call gem_leq_func [|pointer1 ; pointer2 |]
```

```
"gem_leq_res" builder, (A.Int, 0))
              | A.Greater ->
               (L.build_call gem_ge_func [| pointer1 ; pointer2 |]
"gem_ge_res" builder, (A.Int, 0))
              | A.Geq ->
               (L.build_call gem_geq_func [| pointer1 ; pointer2 |]
"gem_geq_res" builder, (A.Int, 0))
              )

          | (A.Lattice, A.Lattice) | (A.Lattice, A.Int) | (A.Int,
A.Lattice) ->
            (* convert the Int to a Lattice *)
            let arg1 = if t1 = A.Int then (L.build_call int_to_bn_func
[| e1' |] "int_to_bn_res1" builder) else e1' in
            let arg2 = if t2 = A.Int then (L.build_call int_to_bn_func
[| e2' |] "int_to_bn_res2" builder) else e2' in
            (match op with
                A.Add ->
                (L.build_call lat_add_func [| arg1 ; arg2 |]
"lat_add_res" builder, (A.Lattice, 0))
              | A.Sub ->
                (L.build_call lat_sub_func [| arg1 ; arg2 |]
"lat_sub_res" builder , (A.Lattice, 0))
              | A.Mult ->
                (L.build_call lat_mult_func [| arg1 ; arg2 |]
"lat_mult_res" builder, (A.Lattice, 0))
              | A.Div ->
                (L.build_call lat_div_func [| arg1 ; arg2 |]
"lat_div_res" builder, (A.Lattice, 0))
              | A.Power ->
                (L.build_call lat_pow_func [| arg1 ; arg2 |]
"lat_pow_res" builder, (A.Lattice, 0))
              | A.Mod ->
                (L.build_call lat_mod_func [| arg1 ; arg2 |]
"lat_mod_res" builder, (A.Lattice, 0))
              | A.Equal ->
                (L.build_call lat_eq_func [| arg1 ; arg2 |]
"lat_eq_res" builder, (A.Int, 0))
              | A.Neq ->
               (L.build_call lat_neq_func [|arg1 ; arg2 |]
"lat_neq_res" builder, (A.Int, 0))
              | A.Less ->
               (L.build_call lat_le_func [|arg1 ; arg2 |] "lat_le_res"
builder, (A.Int,0))
              | A.Leq ->
               (L.build_call lat_leq_func [|arg1 ; arg2 |]
"lat_leq_res" builder, (A.Int, 0))
              | A.Greater ->
               (L.build_call lat_ge_func [| arg1 ; arg2 |]
"lat_ge_res" builder, (A.Int, 0))
              | A.Geq ->
               (L.build_call lat_geq_func [| arg1 ; arg2 |]
"lat_geq_res" builder, (A.Int, 0))
              )
```

```
        )
      | A.Unop(op, e) ->
        let e', (t, _) = expr tree builder e in
        (( match t with
          A.Gem ->
            let pointer1 = L.build_alloca gem_type "e" builder in
            let _ = L.build_store e' pointer1 builder in
            (match op with
            A.Not -> L.build_call gem_inverse_func [| pointer1 |]
"gem_inverse" builder )

          | _ ->(match op with
            A.Neg     -> L.build_neg e' "tmp" builder
          | A.Not     -> L.build_not e' "tmp" builder
        )) , (t, 0))
      | A.Assign (s, e) -> let (e', (t, _)) = expr tree builder e and
        left_type = (fst (snd (lookup s tree))) in (match (left_type,
t) with
        (* Lattice Initialization cases *)
        | (A.Lattice, A.Pointer(A.Char)) ->
          let pointer =
            L.build_call lat_str_create_func [| e' |] "lat_str_res"
builder in
          ignore(L.build_store pointer (fst (lookup s tree))
builder) ; (pointer, (left_type, 0))
        | (A.Lattice, A.Int) ->
          let pointer =
            L.build_call int_to_bn_func [| e' |] "int_to_bn_res"
builder in
          ignore(L.build_store pointer (fst (lookup s tree))
builder) ; (pointer, (left_type, 0))
        | (A.Lattice, A.Gem) ->
          let pointer =
            let pointer1 = L.build_alloca gem_type "e" builder in
            let _ = L.build_store e' pointer1 builder in
            L.build_call gem_to_lat_func [| pointer1 |]
"gem_to_lat_res" builder in
          ignore(L.build_store pointer (fst (lookup s tree))
builder) ; (pointer, (left_type, 0))
        (* Regular *)
        | _ -> ignore (L.build_store e' (fst (lookup s tree))
builder); ( e', (t, 0)) )

      (* | A.Call ("print", [e; expr_t]) | A.Call ("printb", [e;
expr_t]) ->
        let var = expr builder in_b (e,expr_t) in
          (match expr_t with
            A.Int -> L.build_call printf_func [| int_format_str ;
(var) |]
            | A.String -> L.build_call printf_func [|
string_format_str ; (var)|]
            | _ -> raise(Failure("Couldn't match printf"))) "printf"
builder *)
      (* | A.Call ("print", [e]) | A.Call ("printb", [e]) -> *)
      | A.Call ("gem_get", [e ; index ]) ->
```

```
       let (e', (t,_)) = expr tree builder e and
        (index', (t',_)) = expr tree builder index in
       (L.build_call gem_get_func [|e' ; index' |] "gem_get_func"
builder, (A.Lattice, 0))
      | A.Call ("print_gem", [e] ) ->
      let (e', (t, _)) = expr tree builder e in
      (L.build_call print_gem_func [| e'|] "print_gem_func" builder,
(t, 0))
      | A.Call ("print_lat", [e] ) ->
      let (e', (t, _)) = expr tree builder e in
      (L.build_call print_lat_func [| e'|] "print_lat_func" builder,
(t, 0))
      | A.Call ("print", act) ->
        let actuals, _ = List.split (List.rev (List.map (expr tree
builder)
        (List.rev act))) in
        let result = "0" in
        (L.build_call printf_func (Array.of_list actuals) result
builder,
          (A.Pointer(A.Char), 0))
      (* | A.Call ("print_string", [e]) ->
        (* let actuals *)
        (L.build_call printf_func [| string_format_str ; (expr tree
builder e)|] "print_string" builder,  (A.Pointer(A.Char), 0)) *)
      (* | A.Call ("printbig", [e]) ->
        L.build_call printbig_func [| (expr tree builder e) |]
"printbig" builder, (A.Pointer(A.Char, 0)) *)
      | A.Call ("hash_md5", [e]) ->
        let (e', (t, _)) = expr tree builder e in
        (L.build_call hash_md5_func [| e'|] "hash_md5_func" builder,
(t, 0))
      | A.Call (f, act) ->
        let (fdef, fdecl) = StringMap.find f function_decls in
        let actuals, _ = List.split (List.rev (List.map (expr tree
builder) (List.rev act))) in
        let result =
          (match fdecl.A.typ with
              A.Void -> ""
            | _ -> f ^ "_result" ) in
        (L.build_call fdef (Array.of_list actuals) result builder,
(fdecl.A.typ, 0))
      | e -> (print_string ((A.string_of_expr e) ^ "
failed\n")) ;raise(Failure("illegal expression"))
    in

    (* Invoke "f builder" if the current block doesn't already
       have a terminal (e.g., a branch). *)
    let add_terminal builder f =
      match L.block_terminator (L.insertion_block builder) with
        Some _ -> ()
      | None -> ignore (f builder) in

    (* Build the code for the given statement; return the builder for
       the statement's successor *)
    let rec stmt tree builder = function
```

```
        A.Block (vl, sl) ->
          let new_tree =
            let add_local m (t,n) =
              let new_local = L.build_alloca (ltype_of_typ t) n
builder
              in StringMap.add n (new_local, (t,0)) m
            in List.fold_left add_local tree vl
          in List.fold_left (stmt tree) builder sl

      | A.Expr e -> ignore (expr tree builder e); builder
      | A.Return e -> ignore (match fdecl.A.typ with
          A.Void -> L.build_ret_void builder
        | _ -> L.build_ret (fst (expr tree builder e)) builder);
builder
      | A.If (predicate, then_stmt, else_stmt) ->
        let bool_val = fst( expr tree builder predicate) in
        let merge_bb = L.append_block context "merge" the_function in

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

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

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

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

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

        let pred_builder = L.builder_at_end context pred_bb in
        let bool_val = fst( expr tree pred_builder predicate) in

        let merge_bb = L.append_block context "merge" the_function in
        ignore (L.build_cond_br bool_val body_bb merge_bb
pred_builder);
        L.builder_at_end context merge_bb

      | A.For (e1, e2, e3, body) -> stmt tree builder
        ( A.Block ([], [A.Expr e1 ; A.While (e2, A.Block ([], [body ;
A.Expr e3])) ] ))
    in
    let local_vars =
      let add_formal m (t, n) p = L.set_value_name n p;
```

```
        let local = L.build_alloca (ltype_of_typ t) n builder in
        ignore (L.build_store p local builder);
        StringMap.add n (local, (t,0)) m in

      let add_local m (t, n) =
        let local_var = L.build_alloca (ltype_of_typ t) n builder
        in StringMap.add n (local_var, (t,0)) m in

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

    (* Build the code for each statement in the function *)
    let builder = stmt local_vars builder (A.Block ([], fdecl.A.body))
in

    (* Add a return if the last block falls off the end *)
    add_terminal builder (match fdecl.A.typ with
        A.Void -> L.build_ret_void
      | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in

  List.iter build_function_body functions;
  the_module

===== end of codegen.ml =====


==============================
 cryptal.ml
==============================

(* Top-level of the Cryptal compiler: scan & parse the input,
   check the resulting AST, generate LLVM IR, and dump the module *)

module StringMap = Map.Make(String)

type action = Ast | LLVM_IR | Compile

let _ =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM
IR");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./cryptal.native [-a|-l|-c] [file.mc]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename)
usage_msg;
  let lexbuf = Lexing.from_channel !channel in
```

```
  let ast = Parser.program Scanner.token lexbuf in
  Semant.check ast;
  match !action with
    Ast -> print_string (Ast.string_of_program ast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule
(Codegen.translate ast))
  | Compile -> let m = Codegen.translate ast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)

===== end of cryptal.ml =====


==============================
 parser.mly
==============================

/* Ocamlyacc parser for Cryptal */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA
%token PLUS MINUS TIMES DIVIDE MOD POWER ASSIGN NOT
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token RETURN IF ELSE FOR WHILE INT BOOL VOID
%token GEM UINT CHAR LATTICE
%token <int> LITERAL
%token <string> ID
%token <string> STRING
%token <char> CHARLIT
%token NULL
%token EOF

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

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }
```

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



fdecl:
   typ ID LPAREN formals_opt RPAREN LBRACE  stmt_list RBRACE
     { { typ = $1;
       fname = $2;
       formals = $4;
       locals = List.rev (fst $7);
       body = List.rev (snd $7) } }



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

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

typ:
    INT { Int }
  | BOOL { Bool }
  | VOID { Void }
  | CHAR { Char }
  | GEM { Gem }
  | UINT { UInt }
  | LATTICE { Lattice }


/*vdecl_list:
    { [] }
  | vdecl_list vdecl { $2 :: $1 }*/

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

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

stmt:
    expr SEMI { Expr $1 }
  | RETURN SEMI { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev (fst $2), List.rev (snd
$2)) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([],
[])) }
```

```
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr 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:
    LITERAL          { Literal($1) }
  | STRING           { String($1) }
  | CHARLIT          { Ch($1) }
  | TRUE             { BoolLit(true) }
  | FALSE            { BoolLit(false) }
  | ID               { Id($1) }
  | NULL             { Null }
  | expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | expr AND    expr { Binop($1, And,   $3) }
  | expr OR     expr { Binop($1, Or,    $3) }
  | expr MOD expr{ Binop($1, Mod , $3) }
  | expr POWER expr{ Binop($1, Power , $3) }
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr         { Unop(Not, $2) }
  | ID ASSIGN expr   { Assign($1, $3) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LPAREN expr COMMA expr RPAREN {Gem($2, $4)}
  | LPAREN expr RPAREN { $2 }

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

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

===== end of parser.mly =====


==============================
 scanner.mll
==============================

(* Ocamllex scanner for Cryptal *)
```

```
{ open Parser
  module B = Buffer}
(* Deleted BITXOR, BITOR, BITAND *)
rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| "**"     { POWER    }
| '/'      { DIVIDE }
| '%'      { MOD }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "int"    { INT }
| "gem"    { GEM }
| "uint"   { UINT }
| "lat"    { LATTICE }
| "char"   { CHAR }
| "NULL"   { NULL }
| "bool"   { BOOL }
| "void"   { VOID }
| "true"   { TRUE }
| "false"  { FALSE }
| "'"      { CHARLIT (read_char lexbuf) }
| '"'      { STRING (build_str (B.create 100) lexbuf) }
| ['0'-'9']+ as lxm { LITERAL(int_of_string lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped
char)) }

and comment = parse
  "*/" { token lexbuf }
| _    { comment lexbuf }
```

```
and build_str sb = parse
 | '"'      { B.contents sb }
 | '\\''\\' { B.add_char sb '\\';  build_str sb lexbuf }
 | '\\''"'  { B.add_char sb '"';   build_str sb lexbuf }
 | '\\''\'' { B.add_char sb '\'';  build_str sb lexbuf }
 | '\\''n'  { B.add_char sb '\n';  build_str sb lexbuf }
 | '\\''r'  { B.add_char sb '\r';  build_str sb lexbuf }
 | '\\''t'  { B.add_char sb '\t';  build_str sb lexbuf }
 | _ as t   { B.add_char sb t;     build_str sb lexbuf }

 and read_char = parse
 | '\\''\\' { check_length '\'' lexbuf }
 | '\\''"'  { check_length '"' lexbuf }
 | '\\''\'' { check_length '\'' lexbuf }
 | '\\''n'  { check_length '\n' lexbuf }
 | '\\''r'  { check_length '\r' lexbuf }
 | '\\''t'  { check_length '\t' lexbuf }
 | '\\''0'  { check_length (char_of_int 0) lexbuf } (* '\0' char in C
*)
 (* this should only match characters of length 1, as expected *)
 | (_ as t) { check_length t lexbuf }


and check_length buf = parse
 | '\'' { buf }
 | _ as t { raise( Failure ("illegal char literal " ^ Char.escaped
t)) }

===== end of scanner.mll =====


==============================
 semant.ml
==============================

(* Semantic checking for Cryptal compiler *)

open Ast

module StringMap = Map.Make(String)

(* Semantic checking of a program. Returns void if successful,
   throws an exception if something is wrong.

   Check each global variable, then check each function *)

let check (globals, functions) =

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

```
    in helper (List.sort compare list)
  in

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

  (* Raise an exception of the given rvalue type cannot be assigned to
     the given lvalue type *)
  let check_assign lvaluet rvaluet err =
     if lvaluet == rvaluet
     || lvaluet == Int && rvaluet == Gem
     || lvaluet == Int && rvaluet == Lattice
     || lvaluet == Lattice && rvaluet == Int
     || lvaluet == Lattice && rvaluet == String
     || lvaluet == Lattice && rvaluet == Gem
     || lvaluet == String && rvaluet == Char
     || lvaluet == Int && rvaluet == Char
     || lvaluet == Char && rvaluet == Int
     || lvaluet == Bool && rvaluet == Int
     then lvaluet else raise err
  in

  (**** Checking Global Variables ****)

  List.iter (check_not_void (fun n -> "illegal void global " ^ n))
globals;

  report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd
globals);

  (**** Checking Functions ****)

  if List.mem "print" (List.map (fun fd -> fd.fname) functions)
  then raise (Failure ("function print may not be defined")) else ();
  if List.mem "print_gem" (List.map (fun fd -> fd.fname) functions)
  then raise (Failure ("function print_gem may not be defined")) else
();
  if List.mem "print_lat" (List.map (fun fd -> fd.fname) functions)
  then raise (Failure ("function print_lat may not be defined")) else
();
  if List.mem "hash_md5" (List.map (fun fd -> fd.fname) functions)
  then raise (Failure ("function hash_md5 may not be defined")) else
();

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

  (* Function declaration for a named function *)
  let built_in_decls =
        StringMap.add "print"
        { typ = Void; fname = "print"; formals = [(Int, "x")];
       locals = []; body = [] }
```

```
       (StringMap.add "print_gem"
       { typ = Void; fname = "print_gem"; formals = [(Gem, "x")];
       locals = []; body =[] }
       (StringMap.add "hash_md5"
       { typ = String; fname = "hash_md5"; formals = [(String, "x")];
       locals = []; body =[] }
      (StringMap.singleton "print_lat"
       { typ = Void; fname = "print_lat"; formals = [(Lattice, "x")];
       locals = []; body = [] })))
  in

  let function_decls = List.fold_left (fun m fd -> StringMap.add
fd.fname fd m)
                       built_in_decls functions
  in

  let function_decl s = try StringMap.find s function_decls
       with Not_found -> raise (Failure ("unrecognized function " ^
s))
  in

  let _ = function_decl "main" in (* Ensure "main" is defined *)

  let check_function func =

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

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

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

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

    (* Type of each variable (global, formal, or local *)
    let symbols = List.fold_left (fun m (t, n) -> StringMap.add n t m)
      StringMap.empty (globals @ func.formals @ func.locals )
    in

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

    (* Return the type of an expression or throw an exception *)
    let rec expr tree = function
     Literal _ -> Int
      | BoolLit _ -> Bool
      | Ch _ -> Char
      | Null -> Void
```

```
      | String _ -> String
      | Gem (e1, e2) -> let v1 = expr tree e1 and v2 = expr tree e2 in
        (match (v1, v2) with
            (Int, Int) -> Gem
          | (Gem, Int) -> Gem
          | (Lattice, Int) -> Gem
          | (Int, Gem) -> Gem
          | (Lattice, Gem) -> Gem
          | (Int, Lattice) -> Gem
          | (Gem, Lattice) -> Gem
          | (Lattice, Lattice) -> Gem
          | _ -> raise (Failure ("illegal constructor for gem (" ^
string_of_typ v1
        ^ "," ^ string_of_typ v2 ^ ")"))
        )
      | Lat _ -> Lattice
      | Id s -> type_of_identifier s tree
      | Binop(e1, op, e2) as e -> let t1 = expr tree e1 and t2 = expr
tree e2 in
         (match op with
          (* Rules for general C operators *)
          Add | Sub | Mult | Div | Power  when t1 = Int && t2 = Int ->
Int
        | Power                          when t1 = Int && t2 = Gem ->
Int
           | Equal | Neq                    when t1 = t2 || t1 = Int
&& t2 = Char || t1 = Char && t2 = Int || t1 = Int && t2 = Bool || t1 =
Bool && t2 = Int || t1 = String && t2 = Char || t1 = Char && t2 =
String -> Bool
           | Less | Leq | Greater | Geq      when t1 = t2 || t1 = Int
&& t2 = Char || t1 = Char && t2 = Int || t1 = Int && t2 = Bool || t1 =
Bool && t2 = Int || t1 = String && t2 = Char || t1 = Char && t2 =
String -> Bool
           | And | Or                       when t1 = Bool && t2 =
Bool -> Bool
        | Mod                            when t1 = Int -> Int
        (* Rules for gems *)
        | Add | Sub | Mult | Div         when t1 = Gem && t2 = Gem ||
t1 = Gem && t2 = Int || t1 = Int && t2 = Gem -> Gem
        | Equal | Neq                    when t1 = Gem && t2 = Int ||
t1 = Int && t2 = Gem -> Bool
        | Less | Leq | Greater | Geq      when t1 = Gem && t2 = Int ||
t1 = Int && t2 = Gem -> Bool
        | Power                          when t1 = Gem && t2 = Gem ||
t1 = Gem && t2 = Int || t1 = Gem && t2 = Lattice-> Gem
        (* Rules for lattices *)
        | Add | Sub | Mult | Div         when t1 = Lattice && t2 =
Lattice || t1 = Int && t2 = Lattice || t1 = Lattice && t2 = Int->
Lattice
        | Equal | Neq                    when t1 = Lattice && t2 =
Int || t1 = Int && t2 = Lattice -> Bool
        | Less | Leq | Greater | Geq      when t1 = Int && t2 =
Lattice || t1 = Lattice && t2 = Int -> Bool
        | Power                          when t1 = Lattice && t2 =
Lattice || t1 = Lattice && t2 = Int || t1 = Int && t2 = Lattice  ->
```

```
Lattice
        (* Rules for gems and lattices *)
        | Add | Sub | Mult | Div          when t1 = Gem && t2 =
Lattice || t1 = Lattice && t2 = Gem -> Gem
        | Equal | Neq                    when t1 = Gem && t2 =
Lattice || t1 = Lattice && t2 = Gem || t1 = Lattice && t2 = String ||
t1 = String && t2 = Lattice -> Bool
        | Less | Leq | Greater | Geq      when t1 = Gem && t2 =
Lattice || t1 = Lattice && t2 = Gem || t1 = Lattice && t2 = String ||
t1 = String && t2 = Lattice -> Bool
        | Power                          when t1 = Lattice && t2 =
Gem -> Lattice
        | Mod                            when t1 = Lattice -> Lattice
            | _ -> raise (Failure ("illegal binary operator " ^
                  string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                  string_of_typ t2 ^ " in " ^ string_of_expr e))
             )
      | Unop(op, e) as ex -> let t = expr tree e in
           (match op with
           Neg       when t = Int -> Int
         | Neg       when t = Gem -> Gem
         | Neg       when t = Lattice -> Lattice
          | Not       when t = Bool -> Bool
         | Not       when t = Gem -> Gem
         | _ -> raise (Failure ("illegal unary operator " ^
string_of_uop op ^
                string_of_typ t ^ " in " ^ string_of_expr ex)))
      | Noexpr -> Void
      | Assign(var, e) as ex -> let lt = type_of_identifier var tree
                                and rt = expr tree e in
        check_assign lt rt (Failure ("illegal assignment " ^
string_of_typ lt ^
                           " = " ^ string_of_typ rt ^ " in " ^
                           string_of_expr ex))
      | Call(fname, actuals) as call -> let fd = function_decl fname
in
        (* if fname - "printf" *)
        if fname <> "print" then
         if List.length actuals != List.length fd.formals then
           raise (Failure ("expecting " ^ string_of_int
             (List.length fd.formals) ^ " arguments in " ^
string_of_expr call))
         else
           List.iter2 (fun (ft, _) e -> let et = expr tree e in
              ignore (check_assign ft et
                (Failure ("illegal actual argument found " ^
string_of_typ et ^
                 " expected " ^ string_of_typ ft ^ " in " ^
string_of_expr e))))
             fd.formals actuals;
           fd.typ
    in

    let check_num_expr tree e = if expr tree e != Int && expr tree e !
= Gem && expr tree e != Lattice
```

```
      then raise (Failure ("expected Number in " ^ string_of_expr e))
      else (true) in

    let check_bool_expr tree e = if expr tree e != Bool &&
check_num_expr tree e != true
     then raise (Failure ("expected Boolean expression in " ^
string_of_expr e))
     else () in


    (* Verify a statement or throw an exception *)
    let rec stmt tree = function
        Block (_, sl) -> let rec check_block block_tree = function
         [ Return _ as s ] -> stmt block_tree s
         | Return _ :: _ -> raise (Failure "nothing may follow a
return")
         | (Block (_,_) as b) :: ss ->stmt block_tree b; check_block
block_tree ss
         | s :: ss -> stmt block_tree s ; check_block block_tree ss
         | [] -> ()
        in check_block tree sl
      | Expr e -> ignore (expr tree e)
      | Return e -> let t = expr tree e in if t = func.typ then ()
else
         raise (Failure ("return gives " ^ string_of_typ t ^ "
expected " ^
                         string_of_typ func.typ ^ " in " ^
string_of_expr e))

      | If(p, b1, b2) -> check_bool_expr tree p; stmt tree b1; stmt
tree b2
      | For(e1, e2, e3, st) -> ignore (expr tree e1); check_bool_expr
tree e2;
                               ignore (expr tree e3); stmt tree st
      | While(p, s) -> check_bool_expr tree p; stmt tree s
    in

    let tree = List.fold_left (fun m (t,n) -> StringMap.add n t m)
      StringMap.empty (globals @ func.formals @ func.locals) in
    stmt tree (Block ([], func.body))

  in
  List.iter check_function functions

===== end of semant.ml =====


==============================
 crypto_types.h
==============================

/*
* Header file to include types used in cryptographic functions
*/
```

```
struct gem {
  void *value;
  void * mod;
};

===== end of crypto_types.h =====


==============================
 crypto_arith.c
==============================

#include <stdlib.h>
#include "crypto_types.h"
#include <string.h>
#include <openssl/bn.h>
#include <openssl/md5.h>

/* gem functions
* Use the modulus of x
* TODO: We may want to change this
*/
void *int_to_bn_func(int);

struct gem gem_inverse_func(struct gem *x)
{
  BIGNUM * result = BN_new();
  BN_CTX *ctx = BN_CTX_new();
  BN_mod_inverse(result, x->value, x->mod, ctx);
  BN_CTX_free(ctx);
  struct gem res;
  res.value = result;
  res.mod = x->mod;
  return res;
}

struct gem gem_add_func(struct gem* x, struct gem* y)
{
  BIGNUM *result = BN_new();
  BN_CTX *ctx = BN_CTX_new();

  BN_mod_add(result, x->value, y->value, x->mod,ctx);
  BN_CTX_free(ctx);
  struct gem res;
  res.value = result;
  res.mod = x->mod;
  return res;
}

struct gem gem_sub_func(struct gem* x, struct gem* y)
{
  BIGNUM *result = BN_new();
  BN_CTX *ctx = BN_CTX_new();

  BN_mod_sub(result, x->value, y->value, x->mod,ctx);
```

```
  BN_CTX_free(ctx);
  struct gem res;
  res.value = result;
  res.mod = x->mod;
  return res;
}

struct gem gem_mult_func(struct gem* x, struct gem* y)
{
  BIGNUM *result = BN_new();
  BN_CTX *ctx = BN_CTX_new();

  BN_mod_mul(result, x->value, y->value, x->mod,ctx);
  BN_CTX_free(ctx);
  struct gem res;
  res.value = result;
  res.mod = x->mod;
  return res;
}

struct gem gem_div_func(struct gem *x, struct gem *y)
{
  BIGNUM *result =  BN_new();
  // BIGNUM *rem = BN_new();
  BN_CTX *ctx = BN_CTX_new();
  struct gem inv = gem_inverse_func(y);
  BN_mod_mul(result,  x->value, inv.value, x->mod, ctx);
  struct gem res;
  res.value = result;
  res.mod = x->mod;
  return res;
}

struct gem gem_pow_func(struct gem *x, struct gem *y)
{
  BIGNUM *result = BN_new();
  BN_CTX *ctx = BN_CTX_new();
  if (BN_is_negative((BIGNUM *)y->value)) {
      BN_mod_inverse(x->value, x->value, x->mod, ctx);
  }
  BN_mod_exp(result, x->value, y->value, x->mod, ctx);
  BN_CTX_free(ctx);
  struct gem res;
  res.value = result;
  res.mod = x->mod;
  return res;
}

//Gem coparators
int gem_eq_func(struct gem *x, struct gem *y)
{
  return (BN_cmp(x->value, y->value) == 0 ? 1 : 0);
}

int gem_neq_func(struct gem *x, struct gem *y)
```

```
{
  return (BN_cmp(x->value, y->value) == 0 ? 0 : 1);
}

int gem_le_func(struct gem *x, struct gem *y)
{
  // fprintf(stdout, "x val: ");
  // BN_print_fp(stdout, x->value);
  // fprintf(stdout, "\ny val: ");
  // BN_print_fp(stdout, y->value);
  // fprintf(stdout, "\n");
  return (BN_cmp(x->value, y->value) < 0 ? 1 : 0);
}

int gem_ge_func(struct gem *x, struct gem *y)
{
  return (BN_cmp(x->value, y->value) >0 ? 1 : 0);
}

int gem_leq_func(struct gem *x, struct gem *y)
{
  return !gem_ge_func(x,y);
}

int gem_geq_func(struct gem *x, struct gem *y)
{
  return !gem_le_func(x, y);
}



//Int, Gem math
struct gem int_gem_add_func(void *x, struct gem *y)
{
  BIGNUM *result = BN_new();
  BN_CTX *ctx = BN_CTX_new();
  BIGNUM *bn_val = x;
  BN_mod_add(result, bn_val, y->value, y->mod, ctx);
  BN_CTX_free(ctx);
  struct gem res;
  res.value = result;
  res.mod = y->mod;
  return res;
}

struct gem int_gem_sub_func(void *x, struct gem *y)
{
  BIGNUM *result = BN_new();
  BN_CTX *ctx = BN_CTX_new();
  BIGNUM *bn_val = x;
  BN_mod_sub(result, bn_val, y->value, y->mod, ctx);
  BN_CTX_free(ctx);
  struct gem res;
  res.value = result;
  res.mod = y->mod;
```

```
  return res;
}

struct gem int_gem_mult_func(void *x, struct gem *y)
{
  BIGNUM *result = BN_new();
  BN_CTX *ctx = BN_CTX_new();
  BIGNUM *bn_val = x;
  BN_mod_mul(result, bn_val, y->value, y->mod, ctx);
  BN_CTX_free(ctx);
  struct gem res;
  res.value = result;
  res.mod = y->mod;
  return res;
}

struct gem int_gem_div_func(void *x, struct gem *y)
{
  BIGNUM *result = BN_new();
  BIGNUM *rem = BN_new();
  BN_CTX *ctx = BN_CTX_new();
  BIGNUM *bn_val = x;
  BN_div(result, rem, bn_val, y->value, ctx);
  BN_CTX_free(ctx);
  struct gem res;
  res.value = result;
  res.mod = y->mod;
  return res;
}

struct gem int_gem_pow_func(void *x, struct gem *y)
{
  BIGNUM *result = BN_new();
  BN_CTX *ctx = BN_CTX_new();
  BIGNUM *bn_val = x;
  if (BN_is_negative((BIGNUM *)y->value)) {
      BN_mod_inverse(y->value, y->value, y->mod, ctx);
  }
  BN_mod_exp(result, y->value, bn_val, y->mod, ctx);
  BN_CTX_free(ctx);
  struct gem res;
  res.value = result;
  res.mod = y->mod;
  return res;
}

// Int, Gem comparators
int int_gem_eq_func(void *x, struct gem *y)
{
  BIGNUM *bn_val = x;
  return (BN_cmp(bn_val, y->value) == 0 ? 1 : 0);
}

int int_gem_neq_func(void *x, struct gem *y)
{
```

```
  BIGNUM *bn_val = x;
  return (BN_cmp(bn_val, y->value) == 0 ? 0 :1);
}

int int_gem_less_func(void *x, struct gem *y)
{
  BIGNUM *bn_val = x;
  return (BN_cmp(bn_val, y->value) <0 ? 1 :0);
}

int int_gem_greater_func(void *x, struct gem *y)
{
  BIGNUM *bn_val = x;
  return (BN_cmp(bn_val, y->value) > 0 ? 1 :0);
}

int int_gem_leq_func(void *x, struct gem *y)
{
  return !int_gem_greater_func(x, y);
}

int int_gem_geq_func(void *x, struct gem *y)
{
  return !int_gem_less_func(x, y);
}

//Functions to create a new gem
void *int_to_bn_func(int mod)
{
  char buf[25];
  sprintf(buf, "%d", mod);
  BIGNUM *result = BN_new();
  BN_dec2bn(&result, buf);
  // printf("int_to bn: %d\n", mod);
  // BN_print_fp(stdout, result );
  // printf("\n");
  return result;
}

//Int, Int -> Gem
void * gem_mod_create_func(int val, int mod)
{
  BIGNUM *result = BN_new();
  BIGNUM *bn_val = int_to_bn_func(val);
  // printf("bn_val: %d\n", val);
  // BN_print_fp(stdout, bn_val);
  // printf("\n");
  BIGNUM *bn_mod = int_to_bn_func(mod);
  BN_CTX *ctx = BN_CTX_new();
  BN_mod(result, bn_val, bn_mod, ctx);
  if (BN_is_negative(result)) {
    BN_add(result, result, bn_mod);
  }
  // printf("bn_mod: %d\n ", mod);
  // BN_print_fp(stdout, result);
```

```
  // printf("\n");
  BN_CTX_free(ctx);
  return result;
}

//Lat, Int -> Gem
void * gem_mod_create_latint_func(void *lat, int mod)
{
  BIGNUM *result = BN_new();
  BIGNUM *bn_val = lat;
  // printf("bn_val: %d\n", val);
  // BN_print_fp(stdout, bn_val);
  // printf("\n");
  BIGNUM *bn_mod = int_to_bn_func(mod);
  BN_CTX *ctx = BN_CTX_new();
  BN_mod(result, bn_val, bn_mod, ctx);
  if (BN_is_negative(result)) {
    BN_add(result, result, bn_mod);
  }
  BN_CTX_free(ctx);
  return result;
}

//Int, Lat -> Gem
void * gem_mod_create_intlat_func(int val, void *mod)
{
  BIGNUM *result = BN_new();
  BIGNUM *bn_val = int_to_bn_func(val);
  BIGNUM *bn_mod = mod;
  BN_CTX *ctx = BN_CTX_new();
  BN_mod(result, bn_val, bn_mod, ctx);
  if (BN_is_negative(result)) {
    BN_add(result, result, bn_mod);
  }
  BN_CTX_free(ctx);
  return result;
}

//Lat, Lat -> Gem
void * gem_mod_create_latlat_func(void *val, void *mod)
{
  BIGNUM *result = BN_new();
  BIGNUM *bn_val = val;
  BIGNUM *bn_mod = mod;
  BN_CTX *ctx = BN_CTX_new();
  BN_mod(result, bn_val, bn_mod, ctx);
  if (BN_is_negative(result)) {
    BN_add(result, result, bn_mod);
  }
  BN_CTX_free(ctx);
  return result;
}

//Gem, Int -> Gem
void * gem_int_create_func( struct gem *x, int mod)
```

```
{
  BIGNUM *result = BN_new();
  BIGNUM *bn_mod = int_to_bn_func(mod);
  BN_CTX *ctx = BN_CTX_new();
  BN_mod(result, x->value, bn_mod, ctx);
  if (BN_is_negative(result)) {
    BN_add(result, result, bn_mod);
  }
  BN_CTX_free(ctx);
  return result;
}

//Int, Gem -> Gem
void *int_gem_create_func(int val, struct gem *x)
{
  BIGNUM *result = BN_new();
  BIGNUM *bn_val = int_to_bn_func(val);
  BN_CTX *ctx = BN_CTX_new();
  BN_mod(result, bn_val, x->mod, ctx);
  if (BN_is_negative(result)) {
    BN_add(result, result, x->mod);
  }
  BN_CTX_free(ctx);
  return result;
}

//Get gem
void *gem_gem(struct gem g, int index)
{
  return index ? g.value : g.mod;
}


//Printing function
void print_gem_func(struct gem x) {
  printf("%s\n", BN_bn2dec(x.value));

}

// ================== LATTICE OPS ==================

// Gem -> Lattice
void *gem_to_lat_func(struct gem *x)
{
  return x->value;
}

//String -> Lattice
void *lat_str_create_func(char *x)
{
  BIGNUM *result = BN_new();
  BN_dec2bn(&result, x);
  return result;
}
```

```
// (Lat, Lat) ops

void *lat_add_func(void *x, void *y)
{
  BIGNUM *result = BN_new();
  BN_add(result, x, y);
  return result;
}

void *lat_sub_func(void *x, void *y)
{
  BIGNUM *result = BN_new();
  BN_sub(result, x, y);
  return result;
}

void *lat_mult_func(void *x, void *y)
{
  BIGNUM *result = BN_new();
  BN_CTX *ctx = BN_CTX_new();

  BN_mul(result, x, y,ctx);
  BN_CTX_free(ctx);
  return result;
}

void *lat_div_func(void *x, void *y)
{
  BIGNUM *result =  BN_new();
  BIGNUM *rem = BN_new();
  BN_CTX *ctx = BN_CTX_new();

  BN_div(result, rem, x, y, ctx);
  return result;
}

void *lat_pow_func(void *x, void *y)
{
  BIGNUM *result = BN_new();
  BN_CTX *ctx = BN_CTX_new();
  BN_exp(result, x, y, ctx);
  BN_CTX_free(ctx);
  return result;
}

void *lat_mod_func(void *x, void *y)
{
  BIGNUM *result =  BN_new();
  BN_CTX *ctx = BN_CTX_new();

  BN_mod(result, x, y, ctx);
  return result;
}

//Lat coparators
```

```
int lat_eq_func(void *x, void *y)
{
  return (BN_cmp(x, y) == 0 ? 1 : 0);
}

int lat_neq_func(void *x, void *y)
{
  return (BN_cmp(x, y) == 0 ? 0 : 1);
}

int lat_le_func(void *x, void *y)
{
  return (BN_cmp(x, y) < 0 ? 1 : 0);
}

int lat_ge_func(void *x, void *y)
{
  return (BN_cmp(x, y) > 0 ? 1 : 0);
}

int lat_leq_func(void *x, void *y)
{
  return !lat_ge_func(x,y);
}

int lat_geq_func(void *x, void *y)
{
  return !lat_le_func(x, y);
}

// Printing a Lat
void print_lat_func(void *lat) {
  printf("%s\n", BN_bn2dec(lat));
}

/* Hash MD5 function */

char *hash_md5_func(char *str) {
  unsigned char digest[16];
  MD5_CTX mtx;

  MD5_Init(&mtx);
  MD5_Update(&mtx, str, strlen(str));
  MD5_Final(digest, &mtx);

  char mdString[33];
  for (int i = 0; i < 16; i++)
      sprintf(&mdString[i*2], "%02x", (unsigned int)digest[i]);

  BIGNUM * hash = BN_new();
  BN_CTX *ctx = BN_CTX_new();
  BN_hex2bn(&hash, mdString);

  char *ret = BN_bn2dec(hash);
```

```
  return ret;
}

===== end of crypto_arith.c =====


==============================
 Makefile
==============================

# Make sure ocamlbuild can find opam-managed packages: first run
#
# eval `opam config env`

# Easiest way to build: using ocamlbuild, which in turn uses ocamlfind

.PHONY : all
all : cryptal.native crypto_arith.o

.PHONY : cryptal.native
cryptal.native :
     ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis -cflags -w,
+a-4-8-26-27-42 \
          cryptal.native

# "make clean" removes all generated files

.PHONY : clean
clean :
     ocamlbuild -clean
     rm -rf testall.log *.diff cryptal scanner.ml parser.ml parser.mli
     rm -rf *.cmx *.cmi *.cmo *.cmx *.o *.s *.ll *.out *.exe *.err
*tar.gz

crypto_arith.o: crypto_arith.c crypto_types.h
     gcc -c crypto_arith.c -o crypto_arith.o -I /usr/local/opt/
openssl/include
# More detailed: build using ocamlc/ocamlopt + ocamlfind to locate
LLVM

OBJS = ast.cmx codegen.cmx parser.cmx scanner.cmx semant.cmx
cryptal.cmx

cryptal : $(OBJS)
     ocamlfind ocamlopt -linkpkg -package llvm -package llvm.analysis
$(OBJS) -o cryptal

scanner.ml : scanner.mll
     ocamllex scanner.mll

parser.ml parser.mli : parser.mly
     ocamlyacc parser.mly

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

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

%.cmx : %.ml
	ocamlfind ocamlopt -c -package llvm $<


### Generated by "ocamldep *.ml *.mli" after building scanner.ml and
parser.ml
ast.cmo :
ast.cmx :
codegen.cmo : ast.cmo
codegen.cmx : ast.cmx
microc.cmo : semant.cmo scanner.cmo parser.cmi codegen.cmo ast.cmo
microc.cmx : semant.cmx scanner.cmx parser.cmx codegen.cmx ast.cmx
parser.cmo : ast.cmo parser.cmi
parser.cmx : ast.cmx parser.cmi
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx
semant.cmo : ast.cmo
semant.cmx : ast.cmx
parser.cmi : ast.cmo

# Building the tarball

TARFILES = ast.ml codegen.ml Makefile cryptal.ml parser.mly \
        scanner.mll semant.ml testall.sh crypto_types.h crypto_arith.c
\
           tests demos

cryptal-llvm.tar.gz : $(TARFILES)
	cd .. && tar czf cryptal/cryptal-llvm.tar.gz \
	   $(TARFILES:%=cryptal/%)

===== end of Makefile =====


==============================
 testall.sh
==============================

#!/bin/sh

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

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

# Path to the LLVM compiler
LLC="llc"
```

```
# Path to the C compiler
CC="cc"

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

# Set time limit for all operations
ulimit -t 30

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

keep=0

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

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

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

# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    eval $* || {
      SignalError "$1 failed on $*"
      return 1
    }
}
```

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

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

    echo -n "$basename..."

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

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.s $
{basename}.exe ${basename}.out" &&
    Run "$CRYPTAL" "$1" ">" "${basename}.ll" &&
    Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s"  "crypto_arith.o"
"-lcrypto" &&
    Run "./${basename}.exe" > "${basename}.out" &&
    Compare ${basename}.out ${reffile}.out ${basename}.diff

    # Report the status and clean up the generated files

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

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

```
    echo -n "$basename..."

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

    generatedfiles=""

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

    # Report the status and clean up the generated files

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

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

shift `expr $OPTIND - 1`

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

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

if [ ! -f crypto_arith.o ]
then
   echo "Could not find crypto_arith.o"
   echo "Try \"make crypto_arith.o\""
   exit 1
```

```
fi

if [ $# -ge 1 ]
then
    files=$@
else
    files="tests/test-*.crp tests/fail-*.crp"
fi

for file in $files
do
    case $file in
      *test-*)
          Check $file 2>> $globallog
          ;;
      *fail-*)
          CheckFail $file 2>> $globallog
          ;;
      *)
          echo "unknown file type $file"
          globalerror=1
          ;;
    esac
done

exit $globalerror

===== end of testall.sh =====


==============================
 demos/chinese-remainder-theorem.c
==============================

#include <stdio.h>

int main(){
  int coeff_a = 2;
  int a = 5;
  int mod_a = 7;

  int coeff_b = 3;
  int b = 4;
  int mod_b = 8;

  int i = 1;
  int j = 1;
  while ((i*coeff_a)%mod_a != 1)
    ++i;
  while ((j*coeff_b)%mod_b != 1)
    ++j;

  printf("%d mod %d\n", (a*mod_b*i + b*mod_a*j)%(mod_a*mod_b),
mod_a*mod_b);
}
```

```
===== end of demos/chinese-remainder-theorem.c =====


==============================
 demos/chinese-remainder-theorem.crp
==============================

int main(){
  int a;
  int mod_a;
  int b;
  int mod_b;

  gem x;
  lat x_scratch;

  gem y;
  lat y_scratch;

  gem z;

  a = 5;
  mod_a = 7;

  b = 4;
  mod_b = 8;

  x = (2, mod_a);
  y = (3, mod_b);

  x_scratch = !x;
  y_scratch = !y;

  z = (x_scratch * a * mod_b + y_scratch * b * mod_a, (mod_a *
mod_b));
  print_gem(z);
}
===== end of demos/chinese-remainder-theorem.crp =====


==============================
 demos/compile_demo
==============================

#! /bin/bash

../cryptal.native "${1}.crp" > "${1}.ll"
llc "${1}.ll" > "${1}.s"
gcc -o "${1}.exe" "${1}.s" "../crypto_arith.o" -lcrypto


===== end of demos/compile_demo =====
```

```
==============================
 demos/diffie-hellman.crp
==============================

gem sign_alice_exponent(gem a) {
  lat alice_secret_exponent;
  gem alice_message_signed;

  alice_secret_exponent = 3;

  alice_message_signed = a ** alice_secret_exponent;

  return alice_message_signed;
}

gem sign_bob_exponent(gem b) {
  lat bob_secret_exponent;
  gem bob_message_signed;

  bob_secret_exponent = 4;

  bob_message_signed = b ** bob_secret_exponent;

  return bob_message_signed;
}

int main() {
  lat PRIME;
  lat NUM;

  gem alice_message;
  gem bob_message;

  PRIME = 15485863;
  NUM = 32452843;

  alice_message = (NUM, PRIME);
  alice_message = sign_alice_exponent(alice_message);

  bob_message = (NUM, PRIME);
  bob_message = sign_bob_exponent(bob_message);


  if (sign_alice_exponent(bob_message) ==
sign_bob_exponent(alice_message)) {
    print("Diffie-Hellman Key Exchange Successful");
  } else {
    print("Diffie-Hellman Key Exchange Failed");
  }

  return 0;
}

===== end of demos/diffie-hellman.crp =====
```

```
==============================
 demos/diffie-hellman.out
==============================

Diffe-Hellman Key Exchange Successful
===== end of demos/diffie-hellman.out =====


==============================
 demos/euclidean-algorithm.c
==============================

#include <stdio.h>

int gcd(int a, int b){
  int rem = a%b;
  while (rem){
    a = b;
    b = rem;
    rem = a%b;
  }
  return b;
}

int main(){
    int a = 10;
    int b = 50;
    int rem;
    int g = gcd(a,b);
    printf("%d\n",g);
}

===== end of demos/euclidean-algorithm.c =====


==============================
 demos/euclidean-algorithm.crp
==============================

lat gcd(lat a, lat b) {
  gem rem;
  rem = (a, b);
  while (rem != 0) {
    a = b;
    b = rem;
    rem = (a, b);
  }
  return b;
}

int main() {
    lat a;
    lat b;
    lat g;
```

```
    a = 10;
    b = 50;
    g = gcd(a, b);
    print_lat(g);
}
===== end of demos/euclidean-algorithm.crp =====


==============================
 demos/hash_md5.crp
==============================

int main() {
     lat a;

     a = hash_md5("hello_world");
     print_lat(a);
}
===== end of demos/hash_md5.crp =====


==============================
 demos/hello-world.crp
==============================

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

===== end of demos/hello-world.crp =====


==============================
 tests/fail-assign1.crp
==============================

int main()
{
  int i;
  bool b;

  i = 42;
  i = 10;
  b = true;
  b = false;
  i = false; /* Fail: assigning a bool to an integer */
}

===== end of tests/fail-assign1.crp =====


==============================
 tests/fail-assign1.err
```

```
==============================

Fatal error: exception Failure("illegal assignment int = bool in i =
false")

===== end of tests/fail-assign1.err =====


==============================
 tests/fail-assign3.crp
==============================

void myvoid()
{
  return;
}

int main()
{
  int i;

  i = myvoid(); /* Fail: assigning a void to an integer */
}

===== end of tests/fail-assign3.crp =====


==============================
 tests/fail-assign3.err
==============================

Fatal error: exception Failure("illegal assignment int = void in i =
myvoid()")

===== end of tests/fail-assign3.err =====


==============================
 tests/fail-dead1.crp
==============================

int main()
{
  int i;

  i = 15;
  return i;
  i = 32; /* Error: code after a return */
}

===== end of tests/fail-dead1.crp =====


==============================
 tests/fail-dead1.err
```

```
==============================

Fatal error: exception Failure("nothing may follow a return")

===== end of tests/fail-dead1.err =====


==============================
 tests/fail-expr1.crp
==============================

int a;
bool b;

void foo(int c, bool d)
{
  int dd;
  bool e;
  a + c;
  c - a;
  a * 3;
  c / 2;
  d + a; /* Error: bool + int */
}

int main()
{
  return 0;
}

===== end of tests/fail-expr1.crp =====


==============================
 tests/fail-expr1.err
==============================

Fatal error: exception Failure("illegal binary operator bool + int in
d + a")

===== end of tests/fail-expr1.err =====


==============================
 tests/fail-expr2.crp
==============================

int a;
bool b;

void foo(int c, bool d)
{
  int d;
  bool e;
  b + a; /* Error: bool + int */
```

```
}

int main()
{
  return 0;
}

===== end of tests/fail-expr2.crp =====


==============================
 tests/fail-expr2.err
==============================

Fatal error: exception Failure("illegal binary operator bool + int in
b + a")

===== end of tests/fail-expr2.err =====


==============================
 tests/fail-for1.crp
==============================

int main()
{
  int i;
  for ( ; true ; ) {} /* OK: Forever */

  for (i = 0 ; i < 10 ; i = i + 1) {
    if (i == 3) return 42;
  }

  for (j = 0; i < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}

===== end of tests/fail-for1.crp =====


==============================
 tests/fail-for1.err
==============================

Fatal error: exception Failure("undeclared identifier j")

===== end of tests/fail-for1.err =====


==============================
 tests/fail-for2.crp
==============================

int main()
```

```
{
  int i;

  for (i = 0; j < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}

===== end of tests/fail-for2.crp =====


==============================
 tests/fail-for2.err
==============================

Fatal error: exception Failure("undeclared identifier j")

===== end of tests/fail-for2.err =====


==============================
 tests/fail-for4.crp
==============================

int main()
{
  int i;

  for (i = 0; i < 10 ; i = j + 1) {} /* j undefined */

  return 0;
}

===== end of tests/fail-for4.crp =====


==============================
 tests/fail-for4.err
==============================

Fatal error: exception Failure("undeclared identifier j")

===== end of tests/fail-for4.err =====


==============================
 tests/fail-for5.crp
==============================

int main()
{
  int i;

  for (i = 0; i < 10 ; i = i + 1) {
    foo(); /* Error: no function foo */
```

```
  }

  return 0;
}

===== end of tests/fail-for5.crp =====


==============================
 tests/fail-for5.err
==============================

Fatal error: exception Failure("unrecognized function foo")

===== end of tests/fail-for5.err =====


==============================
 tests/fail-func1.crp
==============================

int foo() {}

int bar() {}

int baz() {}

void bar() {} /* Error: duplicate function bar */

int main()
{
  return 0;
}

===== end of tests/fail-func1.crp =====


==============================
 tests/fail-func1.err
==============================

Fatal error: exception Failure("duplicate function bar")

===== end of tests/fail-func1.err =====


==============================
 tests/fail-func2.crp
==============================

int foo(int a, bool b, int c) { }

void bar(int a, bool b, int a) {} /* Error: duplicate formal a in bar
*/
```

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

===== end of tests/fail-func2.crp =====


==============================
 tests/fail-func2.err
==============================

Fatal error: exception Failure("duplicate formal a in bar")

===== end of tests/fail-func2.err =====


==============================
 tests/fail-func3.crp
==============================

int foo(int a, bool b, int c) { }

void bar(int a, void b, int c) {} /* Error: illegal void formal b */

int main()
{
  return 0;
}

===== end of tests/fail-func3.crp =====


==============================
 tests/fail-func3.err
==============================

Fatal error: exception Failure("illegal void formal b in bar")

===== end of tests/fail-func3.err =====


==============================
 tests/fail-func4.crp
==============================

int foo() {}

void bar() {}

int print() {} /* Should not be able to define print */

void baz() {}

int main()
```

```
{
  return 0;
}

===== end of tests/fail-func4.crp =====


==============================
 tests/fail-func4.err
==============================

Fatal error: exception Failure("function print may not be defined")

===== end of tests/fail-func4.err =====


==============================
 tests/fail-func5.crp
==============================

int foo() {}

int bar() {
  int a;
  void b; /* Error: illegal void local b */
  bool c;

  return 0;
}

int main()
{
  return 0;
}

===== end of tests/fail-func5.crp =====


==============================
 tests/fail-func5.err
==============================

Fatal error: exception Failure("illegal void local b in bar")

===== end of tests/fail-func5.err =====


==============================
 tests/fail-func6.crp
==============================

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

```
int main()
{
  foo(42, true);
  foo(42); /* Wrong number of arguments */
}

===== end of tests/fail-func6.crp =====


==============================
 tests/fail-func6.err
==============================

Fatal error: exception Failure("expecting 2 arguments in foo(42)")

===== end of tests/fail-func6.err =====


==============================
 tests/fail-func7.crp
==============================

void foo(int a, bool b)
{
}

int main()
{
  foo(42, true);
  foo(42, true, false); /* Wrong number of arguments */
}

===== end of tests/fail-func7.crp =====


==============================
 tests/fail-func7.err
==============================

Fatal error: exception Failure("expecting 2 arguments in foo(42, true,
false)")

===== end of tests/fail-func7.err =====


==============================
 tests/fail-func8.crp
==============================

void foo(int a, bool b)
{
}

void bar()
{
```

```
}

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

===== end of tests/fail-func8.crp =====


==============================
 tests/fail-func8.err
==============================

Fatal error: exception Failure("illegal actual argument found void
expected bool in bar()")

===== end of tests/fail-func8.err =====


==============================
 tests/fail-global1.crp
==============================

int c;
bool b;
void a; /* global variables should not be void */


int main()
{
  return 0;
}

===== end of tests/fail-global1.crp =====


==============================
 tests/fail-global1.err
==============================

Fatal error: exception Failure("illegal void global a")

===== end of tests/fail-global1.err =====


==============================
 tests/fail-global2.crp
==============================

int b;
bool c;
int a;
int b; /* Duplicate global variable */
```

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

===== end of tests/fail-global2.crp =====


==============================
 tests/fail-global2.err
==============================

Fatal error: exception Failure("duplicate global b")

===== end of tests/fail-global2.err =====


==============================
 tests/fail-if2.crp
==============================

int main()
{
  if (true) {
    foo; /* Error: undeclared variable */
  }
}

===== end of tests/fail-if2.crp =====


==============================
 tests/fail-if2.err
==============================

Fatal error: exception Failure("undeclared identifier foo")

===== end of tests/fail-if2.err =====


==============================
 tests/fail-if3.crp
==============================

int main()
{
  if (true) {
    42;
  } else {
    bar; /* Error: undeclared variable */
  }
}

===== end of tests/fail-if3.crp =====
```

```
==============================
 tests/fail-if3.err
==============================

Fatal error: exception Failure("undeclared identifier bar")

===== end of tests/fail-if3.err =====


==============================
 tests/fail-nomain.crp
==============================


===== end of tests/fail-nomain.crp =====


==============================
 tests/fail-nomain.err
==============================

Fatal error: exception Failure("unrecognized function main")

===== end of tests/fail-nomain.err =====


==============================
 tests/fail-return1.crp
==============================

int main()
{
  return true; /* Should return int */
}

===== end of tests/fail-return1.crp =====


==============================
 tests/fail-return1.err
==============================

Fatal error: exception Failure("return gives bool expected int in
true")

===== end of tests/fail-return1.err =====


==============================
 tests/fail-return2.crp
==============================

void foo()
```

```
{
  if (true) return 42; /* Should return void */
  else return;
}

int main()
{
  return 42;
}

===== end of tests/fail-return2.crp =====


==============================
 tests/fail-return2.err
==============================

Fatal error: exception Failure("return gives int expected void in 42")

===== end of tests/fail-return2.err =====


==============================
 tests/fail-while2.crp
==============================

int main()
{
  int i;

  while (true) {
    i = i + 1;
  }

  while (true) {
    foo(); /* foo undefined */
  }

}

===== end of tests/fail-while2.crp =====


==============================
 tests/fail-while2.err
==============================

Fatal error: exception Failure("unrecognized function foo")

===== end of tests/fail-while2.err =====


==============================
 tests/test-add1.crp
==============================
```

```
int add(int x, int y)
{
  return x + y;
}

int main()
{
  print("%d", add(25,17));
  return 0;
}

===== end of tests/test-add1.crp =====


==============================
 tests/test-add1.out
==============================

42

===== end of tests/test-add1.out =====


==============================
 tests/test-arith1.crp
==============================

int main()
{
  print("%d",39 + 3);
  return 0;
}

===== end of tests/test-arith1.crp =====


==============================
 tests/test-arith1.out
==============================

42

===== end of tests/test-arith1.out =====


==============================
 tests/test-arith2.crp
==============================

int main()
{
  print("%d", 1 + 2 * 3 + 4);
  return 0;
}
```

```
===== end of tests/test-arith2.crp =====


==============================
 tests/test-arith2.out
==============================

11

===== end of tests/test-arith2.out =====


==============================
 tests/test-arith3.crp
==============================

int foo(int a)
{
  return a;
}

int main()
{
  int a;
  a = 42;
  a = a + 5;
  print("%d",a);
  return 0;
}

===== end of tests/test-arith3.crp =====


==============================
 tests/test-arith3.out
==============================

47

===== end of tests/test-arith3.out =====


==============================
 tests/test-demo-crt.crp
==============================

int main(){
  int a;
  int mod_a;
  int b;
  int mod_b;

  gem x;
  lat x_scratch;
```

```
  gem y;
  lat y_scratch;

  gem z;

  a = 5;
  mod_a = 7;

  b = 4;
  mod_b = 8;

  x = (2, mod_a);
  y = (3, mod_b);

  x_scratch = !x;
  y_scratch = !y;

  z = (x_scratch * a * mod_b + y_scratch * b * mod_a, (mod_a *
mod_b));
  print_gem(z);
}
```

```
===== end of tests/test-demo-crt.crp =====


==============================
 tests/test-demo-crt.out
==============================

20

===== end of tests/test-demo-crt.out =====


==============================
 tests/test-demo-diffie-hellman.crp
==============================

gem sign_alice_exponent(gem a) {
  lat alice_secret_exponent;
  gem alice_message_signed;

  alice_secret_exponent = 3;

  alice_message_signed = a ** alice_secret_exponent;

  return alice_message_signed;
}

gem sign_bob_exponent(gem b) {
  lat bob_secret_exponent;
  gem bob_message_signed;

  bob_secret_exponent = 4;
```

```
  bob_message_signed = b ** bob_secret_exponent;

  return bob_message_signed;
}

int main() {
  lat PRIME;
  lat NUM;

  gem alice_message;
  gem bob_message;

  PRIME = 15485863;
  NUM = 32452843;

  alice_message = (NUM, PRIME);
  alice_message = sign_alice_exponent(alice_message);

  bob_message = (NUM, PRIME);
  bob_message = sign_bob_exponent(bob_message);


  if (sign_alice_exponent(bob_message) ==
sign_bob_exponent(alice_message)) {
    print("Diffie-Hellman Key Exchange Successful");
  } else {
    print("Diffie-Hellman Key Exchange Failed");
  }

  return 0;
}

===== end of tests/test-demo-diffie-hellman.crp =====


==============================
 tests/test-demo-diffie-hellman.out
==============================

Diffie-Hellman Key Exchange Successful
===== end of tests/test-demo-diffie-hellman.out =====


==============================
 tests/test-demo-euclidean.crp
==============================

lat gcd(lat a, lat b) {
  gem rem;
  rem = (a, b);
  while (rem != 0) {
    a = b;
    b = rem;
    rem = (a, b);
```

```
  }
  return b;
}

int main() {
    lat a;
    lat b;
    lat g;
    a = 10;
    b = 50;
    g = gcd(a, b);
    print_lat(g);
}
```

===== end of tests/test-demo-euclidean.crp =====

==========
 tests/test-demo-euclidean.out
==========

```
10
```

===== end of tests/test-demo-euclidean.out =====

==========
 tests/test-fib.crp
==========

```
int fib(int x)
{
  if (x < 2) return 1;
  return fib(x-1) + fib(x-2);
}

int main()
{
  print("%d\n",fib(0));
  print("%d\n",fib(1));
  print("%d\n",fib(2));
  print("%d\n",fib(3));
  print("%d\n",fib(4));
  print("%d\n",fib(5));
  return 0;
}
```

===== end of tests/test-fib.crp =====

==========
 tests/test-fib.out
==========

```
1
```

```
1
2
3
5
8

===== end of tests/test-fib.out =====


==============================
 tests/test-for1.crp
==============================

int main()
{
  int i;
  for (i = 0 ; i < 5 ; i = i + 1) {
    print("%d\n", i);
  }
  print("%d\n",42);
  return 0;
}

===== end of tests/test-for1.crp =====


==============================
 tests/test-for1.out
==============================

0
1
2
3
4
42

===== end of tests/test-for1.out =====


==============================
 tests/test-for2.crp
==============================

int main()
{
  int i;
  i = 0;
  for ( ; i < 5; ) {
    print("%d\n",i);
    i = i + 1;
  }
  print("%d\n",42);
  return 0;
}
```

```
===== end of tests/test-for2.crp =====


==============================
 tests/test-for2.out
==============================

0
1
2
3
4
42

===== end of tests/test-for2.out =====


==============================
 tests/test-func1.crp
==============================

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

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

===== end of tests/test-func1.crp =====


==============================
 tests/test-func1.out
==============================

42

===== end of tests/test-func1.out =====


==============================
 tests/test-func2.crp
==============================

/* Bug noticed by Pin-Chin Huang */

int fun(int x, int y)
{
```

```
  return 0;
}

int main()
{
  int i;
  i = 1;

  fun(i = 2, i = i+1);

  print("%d", i);
  return 0;
}
```

===== end of tests/test-func2.crp =====

==========
 tests/test-func2.out
==========

```
2
```

===== end of tests/test-func2.out =====

==========
 tests/test-func3.crp
==========

```
void printem(int a, int b, int c, int d)
{
  print("%d\n",a);
  print("%d\n",b);
  print("%d\n",c);
  print("%d\n",d);
}

int main()
{
  printem(42,17,192,8);
  return 0;
}
```

===== end of tests/test-func3.crp =====

==========
 tests/test-func3.out
==========

```
42
17
192
8
```

```
===== end of tests/test-func3.out =====


==============================
 tests/test-func4.crp
==============================

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

int main()
{
  int d;
  d = add(52, 10);
  print("%d",d);
  return 0;
}

===== end of tests/test-func4.crp =====


==============================
 tests/test-func4.out
==============================

62

===== end of tests/test-func4.out =====


==============================
 tests/test-func5.crp
==============================

int foo(int a)
{
  return a;
}

int main()
{
  return 0;
}

===== end of tests/test-func5.crp =====


==============================
 tests/test-func5.out
==============================
```

```
===== end of tests/test-func5.out =====


==============================
 tests/test-func6.crp
==============================

void foo() {}

int bar(int a, bool b, int c) { return a + c; }

int main()
{
  print("%d",bar(17, false, 25));
  return 0;
}

===== end of tests/test-func6.crp =====


==============================
 tests/test-func6.out
==============================

42

===== end of tests/test-func6.out =====


==============================
 tests/test-func7.crp
==============================

int a;

void foo(int c)
{
  a = c + 42;
}

int main()
{
  foo(73);
  print("%d",a);
  return 0;
}

===== end of tests/test-func7.crp =====


==============================
 tests/test-func7.out
==============================
```

```
115

===== end of tests/test-func7.out =====


==============================
 tests/test-func8.crp
==============================

void foo(int a)
{
  print("%d",a + 3);
}

int main()
{
  foo(40);
  return 0;
}

===== end of tests/test-func8.crp =====


==============================
 tests/test-func8.out
==============================

43

===== end of tests/test-func8.out =====


==============================
 tests/test-gcd.crp
==============================

int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}

int main()
{
  print("%d\n", gcd(2,14));
  print("%d\n", gcd(3,15));
  print("%d\n", gcd(99,121));
  return 0;
}

===== end of tests/test-gcd.crp =====
```

```
==============================
 tests/test-gcd.out
==============================

2
3
11

===== end of tests/test-gcd.out =====


==============================
 tests/test-gcd2.crp
==============================

int gcd(int a, int b) {
  while (a != b)
    if (a > b) a = a - b;
    else b = b - a;
  return a;
}

int main()
{
  print("%d\n", gcd(14,21));
  print("%d\n", gcd(8,36));
  print("%d\n", gcd(99,121));
  return 0;
}

===== end of tests/test-gcd2.crp =====


==============================
 tests/test-gcd2.out
==============================

7
4
11

===== end of tests/test-gcd2.out =====


==============================
 tests/test-gem-add0.crp
==============================

int main()
{
  gem a;
  gem b;

  a = (2, 7);
```

```
  b = (3, 7);

  print_gem(a+b);

  return 0;
}

===== end of tests/test-gem-add0.crp =====


==============================
 tests/test-gem-add0.out
==============================

5

===== end of tests/test-gem-add0.out =====


==============================
 tests/test-gem-add1.crp
==============================

int main()
{
  gem a;
  gem b;

  a = (5, 7);
  b = (6, 7);

  print_gem(a+b);
  return 0;
}

===== end of tests/test-gem-add1.crp =====


==============================
 tests/test-gem-add1.out
==============================

4

===== end of tests/test-gem-add1.out =====


==============================
 tests/test-gem-cmp0.crp
==============================

int main(){
  gem a;
  gem b;
```

```
  a = (2,7);
  b = (2,7);

  if (a == b){
    print("%s\n", "True");
  }
  else{
    print("%s\n", "False");
  }
  return 0;
}
```

===== end of tests/test-gem-cmp0.crp =====

==========
 tests/test-gem-cmp0.out
==========

```
True
```

===== end of tests/test-gem-cmp0.out =====

==========
 tests/test-gem-cmp1.crp
==========

```
int main(){
  gem a;
  gem b;

  a = (2,7);
  b = (9,7);

  if (a == b){
    print("%s\n", "True");
  }
  else{
    print("%s\n", "False");
  }
  return 0;
}
```

===== end of tests/test-gem-cmp1.crp =====

==========
 tests/test-gem-cmp1.out
==========

```
True
```

===== end of tests/test-gem-cmp1.out =====

```
==============================
 tests/test-gem-cmp10.crp
==============================

int main(){
  gem a;
  gem b;
  gem c;
  gem d;

  a = (2,7);
  b = (8,7);
  c = (9,7);
  d = (10,7);

  if (a < b){
    print("True\n");
  }
  else{
    print("False\n");
  }
  if (a < c){
    print("True\n");
  }
  else{
    print("False\n");
  }
  if (a < d){
    print("True\n");
  }
  else{
    print("False\n");
  }
  return 0;
}

===== end of tests/test-gem-cmp10.crp =====


==============================
 tests/test-gem-cmp10.out
==============================

False
False
True

===== end of tests/test-gem-cmp10.out =====


==============================
 tests/test-gem-cmp11.crp
==============================
```

```
int main(){
  gem a;
  gem b;
  gem c;
  gem d;

  a = (2,7);
  b = (8,7);
  c = (9,7);
  d = (10,7);

  if (a > b){
    print("True\n");
  }
  else{
    print("False\n");
  }
  if (a > c){
    print("True\n");
  }
  else{
    print("False\n");
  }
  if (a > d){
    print("True\n");
  }
  else{
    print("False\n");
  }
  return 0;
}
```

===== end of tests/test-gem-cmp11.crp =====

==========
 tests/test-gem-cmp11.out
==========

```
True
False
False
```

===== end of tests/test-gem-cmp11.out =====

==========
 tests/test-gem-cmp12.crp
==========

```
int main(){
  gem a;
  gem b;
  gem c;
  gem d;
```

```
  a = (2,7);
  b = (1,7);
  c = (2,7);
  d = (3,7);

  if (a <= b){
    print("True\n");
  }
  else{
    print("False\n");
  }
  if (a <= c){
    print("True\n");
  }
  else{
    print("False\n");
  }
  if (a <= d){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}

===== end of tests/test-gem-cmp12.crp =====


==============================
 tests/test-gem-cmp12.out
==============================

False
True
True

===== end of tests/test-gem-cmp12.out =====


==============================
 tests/test-gem-cmp13.crp
==============================

int main(){
  gem a;
  gem b;
  gem c;
  gem d;

  a = (2,7);
  b = (1,7);
  c = (2,7);
  d = (3,7);
```

```
  if (a >= b){
    print("True\n");
  }
  else{
    print("False\n");
  }
  if (a >= c){
    print("True\n");
  }
  else{
    print("False\n");
  }
  if (a >= d){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}
```

===== end of tests/test-gem-cmp13.crp =====

==========
 tests/test-gem-cmp13.out
==========

```
True
True
False
```

===== end of tests/test-gem-cmp13.out =====

==========
 tests/test-gem-cmp14.crp
==========

```
int main(){
  gem a;
  gem b;
  gem c;
  gem d;

  a = (2,7);
  b = (8,7);
  c = (9,7);
  d = (10,7);

  if (a >= b){
    print("True\n");
  }
  else{
```

```
    print("False\n");
  }
  if (a >= c){
    print("True\n");
  }
  else{
    print("False\n");
  }
  if (a >= d){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}
```

===== end of tests/test-gem-cmp14.crp =====

```
==============================
 tests/test-gem-cmp14.out
==============================

True
True
False
```

===== end of tests/test-gem-cmp14.out =====

```
==============================
 tests/test-gem-cmp15.crp
==============================

int main(){
  gem a;
  gem b;
  gem c;
  gem d;

  a = (2,7);
  b = (8,7);
  c = (9,7);
  d = (10,7);

  if (a <= b){
    print("True\n");
  }
  else{
    print("False\n");
  }
  if (a <= c){
    print("True\n");
  }
```

```
  else{
    print("False\n");
  }
  if (a <= d){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}

===== end of tests/test-gem-cmp15.crp =====


==============================
 tests/test-gem-cmp15.out
==============================

False
True
True

===== end of tests/test-gem-cmp15.out =====


==============================
 tests/test-gem-cmp2.crp
==============================

int main(){
  gem a;
  gem b;

  a = (2,7);
  b = (1,7);

  if (a == b){
    print("%s\n", "True");
  }
  else{
    print("%s\n", "False");
  }
  return 0;
}

===== end of tests/test-gem-cmp2.crp =====


==============================
 tests/test-gem-cmp2.out
==============================

False
```

```
===== end of tests/test-gem-cmp2.out =====


==============================
 tests/test-gem-cmp3.crp
==============================

int main(){
  gem a;
  gem b;

  a = (2,7);
  b = (8,7);

  if (a == b){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}

===== end of tests/test-gem-cmp3.crp =====


==============================
 tests/test-gem-cmp3.out
==============================

False

===== end of tests/test-gem-cmp3.out =====


==============================
 tests/test-gem-cmp4.crp
==============================

int main(){
  gem a;
  gem b;

  a = (2,7);
  b = (2,7);

  if (a != b){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}
```

```
===== end of tests/test-gem-cmp4.crp =====


==============================
 tests/test-gem-cmp4.out
==============================

False

===== end of tests/test-gem-cmp4.out =====


==============================
 tests/test-gem-cmp5.crp
==============================

int main(){
  gem a;
  gem b;

  a = (2,7);
  b = (10,7);

  if (a != b){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}

===== end of tests/test-gem-cmp5.crp =====


==============================
 tests/test-gem-cmp5.out
==============================

True

===== end of tests/test-gem-cmp5.out =====


==============================
 tests/test-gem-cmp6.crp
==============================

int main(){
  gem a;
  gem b;

  a = (2,7);
  b = (9,7);
```

```
  if (a != b){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}
```

```
===== end of tests/test-gem-cmp6.crp =====
```

```
==============================
 tests/test-gem-cmp6.out
==============================
```

```
False
```

```
===== end of tests/test-gem-cmp6.out =====
```

```
==============================
 tests/test-gem-cmp7.crp
==============================
```

```
int main(){
  gem a;
  gem b;

  a = (2,7);
  b = (10,7);

  if (a != b){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}
```

```
===== end of tests/test-gem-cmp7.crp =====
```

```
==============================
 tests/test-gem-cmp7.out
==============================
```

```
True
```

```
===== end of tests/test-gem-cmp7.out =====
```

```
==============================
 tests/test-gem-cmp8.crp
```

```
==============================

int main(){
  gem a;
  gem b;
  gem c;
  gem d;

  a = (2,7);
  b = (1,7);
  c = (2,7);
  d = (3,7);

  if (a < b){
    print("True\n");
  }
  else{
    print("False\n");
  }
  if (a < c){
    print("True\n");
  }
  else{
    print("False\n");
  }
  if (a < d){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}

===== end of tests/test-gem-cmp8.crp =====


==============================
 tests/test-gem-cmp8.out
==============================

False
False
True

===== end of tests/test-gem-cmp8.out =====


==============================
 tests/test-gem-cmp9.crp
==============================

int main(){
  gem a;
  gem b;
```

```
  gem c;
  gem d;

  a = (2,7);
  b = (1,7);
  c = (2,7);
  d = (3,7);

  if (a > b){
    print("True\n");
  }
  else{
    print("False\n");
  }
  if (a > c){
    print("True\n");
  }
  else{
    print("False\n");
  }
  if (a > d){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}
```

===== end of tests/test-gem-cmp9.crp =====

==========
 tests/test-gem-cmp9.out
==========

```
True
False
False
```

===== end of tests/test-gem-cmp9.out =====

==========
 tests/test-gem-div0.crp
==========

```
int main()
{
  gem a;
  gem b;

  a = (2, 7);
  b = (3, 7);
```

```
  print_gem(a/b);
  return 0;
}

===== end of tests/test-gem-div0.crp =====


==============================
 tests/test-gem-div0.out
==============================

3

===== end of tests/test-gem-div0.out =====


==============================
 tests/test-gem-div1.crp
==============================

int main()
{
  gem a;
  gem b;

  a = (2, 7);
  b = (10, 7);

  print_gem( a/b);
  return 0;
}

===== end of tests/test-gem-div1.crp =====


==============================
 tests/test-gem-div1.out
==============================

3

===== end of tests/test-gem-div1.out =====


==============================
 tests/test-gem-int-assign.crp
==============================

int main()
{
  gem a;
  gem b;
  int c;
  int d;
```

```
  c = 3;
  d = 7;
  a = (c, d);
  b = (c, d);

  if (a == b) {
    print("True");
  } else {
    print("False");
  }


  return 0;
}
```

===== end of tests/test-gem-int-assign.crp =====

==========
 tests/test-gem-int-assign.out
==========

```
True
```

===== end of tests/test-gem-int-assign.out =====

==========
 tests/test-gem-lat-assign.crp
==========

```
int main()
{
  gem a;
  gem b;
  lat c;
  lat d;

  c = "12345678912345678912345678912345678 9";
  d = "98765432198765432198765432198765432 1";
  a = (c, d);
  b = (c, d);

  if (a == b) {
    print("True");
  } else {
    print("False");
  }


  return 0;
}
```

===== end of tests/test-gem-lat-assign.crp =====

```
==============================
 tests/test-gem-lat-assign.out
==============================

True

===== end of tests/test-gem-lat-assign.out =====


==============================
 tests/test-gem-mult0.crp
==============================

int main()
{
  gem a;
  gem b;

  a = (2, 7);
  b = (3, 7);

  print_gem( a*b);
  return 0;
}

===== end of tests/test-gem-mult0.crp =====


==============================
 tests/test-gem-mult0.out
==============================

6

===== end of tests/test-gem-mult0.out =====


==============================
 tests/test-gem-mult1.crp
==============================

int main()
{
  gem a;
  gem b;

  a = (5, 7);
  b = (6, 7);

  print_gem(a*b);
  return 0;
}

===== end of tests/test-gem-mult1.crp =====
```

```
==============================
 tests/test-gem-mult1.out
==============================

2

===== end of tests/test-gem-mult1.out =====


==============================
 tests/test-gem-power0.crp
==============================

int main()
{
  gem a;
  gem b;

  a = (2, 7);
  b = (2, 7);

  print_gem(a**b);
  return 0;
}

===== end of tests/test-gem-power0.crp =====


==============================
 tests/test-gem-power0.out
==============================

4

===== end of tests/test-gem-power0.out =====


==============================
 tests/test-gem-power1.crp
==============================

int main()
{
  gem a;
  gem b;

  a = (2, 7);
  b = (3, 7);

  print_gem(a**b);
  return 0;
}
```

```
===== end of tests/test-gem-power1.crp =====


==============================
 tests/test-gem-power1.out
==============================

1

===== end of tests/test-gem-power1.out =====


==============================
 tests/test-gem-sub0.crp
==============================

int main()
{
  gem a;
  gem b;

  a = (2, 7);
  b = (3, 7);

  print_gem(b-a);
  return 0;
}

===== end of tests/test-gem-sub0.crp =====


==============================
 tests/test-gem-sub0.out
==============================

1

===== end of tests/test-gem-sub0.out =====


==============================
 tests/test-gem-sub1.crp
==============================

int main()
{
  gem a;
  gem b;

  a = (2, 7);
  b = (3, 7);

  print_gem(a-b);
  return 0;
}
```

```
===== end of tests/test-gem-sub1.crp =====


==============================
 tests/test-gem-sub1.out
==============================

6

===== end of tests/test-gem-sub1.out =====


==============================
 tests/test-gem0.crp
==============================

int main(){
  gem a;
  a = (2,7);
  print_gem(a);
  return 0;
}

===== end of tests/test-gem0.crp =====


==============================
 tests/test-gem0.out
==============================

2

===== end of tests/test-gem0.out =====


==============================
 tests/test-gem1.crp
==============================

int main(){
  gem a;
  a = (7,3);
  print_gem(a);
  return 0;
}

===== end of tests/test-gem1.crp =====


==============================
 tests/test-gem1.out
==============================

1
```

```
===== end of tests/test-gem1.out =====


==============================
 tests/test-global1.crp
==============================

int x;
int y;

void printx()
{
  print("%d\n",x);
}

void printy()
{
  print("%d\n",y);
}

void incxy()
{
  x = x + 1;
  y = y + 1;
}

int main()
{
  x = 42;
  y = 21;
  printx();
  printy();
  incxy();
  printx();
  printy();
  return 0;
}

===== end of tests/test-global1.crp =====


==============================
 tests/test-global1.out
==============================

42
21
43
22

===== end of tests/test-global1.out =====


==============================
```

```
 tests/test-global2.crp
==============================

bool i;

int main()
{
  int i; /* Should hide the global i */

  i = 42;
  print("%d",i + i);
  return 0;
}

===== end of tests/test-global2.crp =====


==============================
 tests/test-global2.out
==============================

84

===== end of tests/test-global2.out =====


==============================
 tests/test-global3.crp
==============================

int i;
bool b;
int j;

int main()
{
  i = 42;
  j = 10;
  print("%d",i + j);
  return 0;
}

===== end of tests/test-global3.crp =====


==============================
 tests/test-global3.out
==============================

52

===== end of tests/test-global3.out =====


==============================
```

```
 tests/test-hello.crp
==============================

int main()
{
  print("%d\n",42);
  print("%d\n",71);
  print("%d\n",1);
  return 0;
}

===== end of tests/test-hello.crp =====


==============================
 tests/test-hello.out
==============================

42
71
1

===== end of tests/test-hello.out =====


==============================
 tests/test-hello_world.crp
==============================

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

===== end of tests/test-hello_world.crp =====


==============================
 tests/test-hello_world.out
==============================

Hello World


===== end of tests/test-hello_world.out =====


==============================
 tests/test-if1.crp
==============================

int main()
{
  if (true) print("%d\n",42);
```

```
  print("%d\n",17);
  return 0;
}

===== end of tests/test-if1.crp =====


==============================
 tests/test-if1.out
==============================

42
17

===== end of tests/test-if1.out =====


==============================
 tests/test-if2.crp
==============================

int main()
{
  if (true) print("%d\n",42); else print("%d\n",8);
  print("%d\n",17);
  return 0;
}

===== end of tests/test-if2.crp =====


==============================
 tests/test-if2.out
==============================

42
17

===== end of tests/test-if2.out =====


==============================
 tests/test-if3.crp
==============================

int main()
{
  if (false) print("%d\n",42);
  print("%d\n",17);
  return 0;
}

===== end of tests/test-if3.crp =====
```

```
==============================
 tests/test-if3.out
==============================

17

===== end of tests/test-if3.out =====


==============================
 tests/test-if4.crp
==============================

int main()
{
  if (false) print("%d\n",42); else print("%d\n",8);
  print("%d\n",17);
  return 0;
}

===== end of tests/test-if4.crp =====


==============================
 tests/test-if4.out
==============================

8
17

===== end of tests/test-if4.out =====


==============================
 tests/test-if5.crp
==============================

int cond(bool b)
{
  int x;
  if (b)
    x = 42;
  else
    x = 17;
  return x;
}

int main()
{
 print("%d\n",cond(true));
 print("%d\n",cond(false));
 return 0;
}

===== end of tests/test-if5.crp =====
```

```
==============================
 tests/test-if5.out
==============================

42
17

===== end of tests/test-if5.out =====


==============================
 tests/test-lat-add.crp
==============================

int main(){
  lat a;
  lat b;
  lat c;

  a = 12345;
  b = 6789;
  c = a+b;

  if (c == 19134) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}

===== end of tests/test-lat-add.crp =====


==============================
 tests/test-lat-add.out
==============================

True

===== end of tests/test-lat-add.out =====


==============================
 tests/test-lat-assign.crp
==============================

int main(){
  lat a;
  lat b;

  a = 8589934592;
```

```
  b = 8589934593;

  if (a == 8589934592){
    print("True\n");
  }
  else{
    print("False\n");
  }

  if (b == 8589934593){
    print("True\n");
  }
  else{
    print("False\n");
  }


  return 0;
}
```

===== end of tests/test-lat-assign.crp =====

==========
 tests/test-lat-assign.out
==========

```
True
True
```

===== end of tests/test-lat-assign.out =====

==========
 tests/test-lat-big-add.crp
==========

```
int main()
{
  lat a;
  lat b;
  lat c;

  b = "8589934592";
  a = "8589934592";

  c = "17179869184";

  if ((a + b) == c) {
    print("True");
  } else {
    print("False");
  }

  return 0;
```

```
}

===== end of tests/test-lat-big-add.crp =====


==============================
 tests/test-lat-big-add.out
==============================

True

===== end of tests/test-lat-big-add.out =====


==============================
 tests/test-lat-big-div.crp
==============================

int main()
{
  lat a;
  lat b;

  b = "88888888888888";
  a = "88888888888888";

  if (b / a == 1) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}

===== end of tests/test-lat-big-div.crp =====


==============================
 tests/test-lat-big-div.out
==============================

True

===== end of tests/test-lat-big-div.out =====


==============================
 tests/test-lat-big-mod.crp
==============================

int main()
{
  lat a;
  int b;
```

```
  a = 8589934592;
  b = 8589934593;

  if (a % b == 8589934592) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}
```

===== end of tests/test-lat-big-mod.crp =====

==========
 tests/test-lat-big-mod.out
==========

```
True
```

===== end of tests/test-lat-big-mod.out =====

==========
 tests/test-lat-big-mult.crp
==========

```
int main()
{
  lat a;
  lat b;
  lat c;

  b = 2;
  a = "8589934592";


  c = "17179869184";

  if (a * b == c) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}
```

===== end of tests/test-lat-big-mult.crp =====

==========
 tests/test-lat-big-mult.out

```
==============================

True

===== end of tests/test-lat-big-mult.out =====


==============================
 tests/test-lat-big-pow.crp
==============================

int main()
{
  lat a;
  lat b;
  lat c;

  b = 2;
  a = 100000;

  c = "10000000000";

  if (a ** b == c) {
    print("True");
  } else {
    print("True");
  }

  return 0;
}

===== end of tests/test-lat-big-pow.crp =====


==============================
 tests/test-lat-big-pow.out
==============================

True

===== end of tests/test-lat-big-pow.out =====


==============================
 tests/test-lat-big-sub.crp
==============================

int main()
{
  lat a;
  lat b;

  b = 8589934592;
  a = 8589934593;
```

```
  if (a - b == 1) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}
```

===== end of tests/test-lat-big-sub.crp =====

==========
 tests/test-lat-big-sub.out
==========

```
True
```

===== end of tests/test-lat-big-sub.out =====

==========
 tests/test-lat-cmp1.crp
==========

```
int main(){
  lat a;
  lat b;

  a = 123456789;
  b = 123456789;

  if (a == b){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}
```

===== end of tests/test-lat-cmp1.crp =====

==========
 tests/test-lat-cmp1.out
==========

```
True
```

===== end of tests/test-lat-cmp1.out =====

==========
 tests/test-lat-cmp2.crp

```
==============================

int main(){
  lat a;
  lat b;

  a = 123456789;
  b = 123456789;

  if (a >= b){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}

===== end of tests/test-lat-cmp2.crp =====


==============================
 tests/test-lat-cmp2.out
==============================

True

===== end of tests/test-lat-cmp2.out =====


==============================
 tests/test-lat-cmp3.crp
==============================

int main(){
  lat a;
  lat b;

  a = 123456789;
  b = 12345678;

  if (a <= b){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}

===== end of tests/test-lat-cmp3.crp =====


==============================
 tests/test-lat-cmp3.out
```

```
==============================

False

===== end of tests/test-lat-cmp3.out =====


==============================
 tests/test-lat-cmp4.crp
==============================

int main(){
  lat a;
  lat b;

  a = 123456789;
  b = 12345678;

  if (a != b){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}

===== end of tests/test-lat-cmp4.crp =====


==============================
 tests/test-lat-cmp4.out
==============================

True

===== end of tests/test-lat-cmp4.out =====


==============================
 tests/test-lat-cmp5.crp
==============================

int main(){
  lat a;
  lat b;

  a = 123456789;
  b = 12345678;

  if (a < b){
    print("True");
  }
  else{
    print("False");
```

```
  }
  return 0;
}
```

===== end of tests/test-lat-cmp5.crp =====

```
==============================
 tests/test-lat-cmp5.out
==============================

False
```

===== end of tests/test-lat-cmp5.out =====

```
==============================
 tests/test-lat-cmp6.crp
==============================

int main(){
  lat a;
  lat b;

  a = 123456789;
  b = 12345678;

  if (a > b){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}
```

===== end of tests/test-lat-cmp6.crp =====

```
==============================
 tests/test-lat-cmp6.out
==============================

True
```

===== end of tests/test-lat-cmp6.out =====

```
==============================
 tests/test-lat-cmp7.crp
==============================

int main(){
  lat a;
  lat b;
```

```
  lat c;

  a = 123456789;
  b = 12345678;
  c = a % b;

  if (c == 9){
    print("True");
  }
  else{
    print("False");
  }
  return 0;
}
```

===== end of tests/test-lat-cmp7.crp =====

==========
 tests/test-lat-cmp7.out
==========

```
True
```

===== end of tests/test-lat-cmp7.out =====

==========
 tests/test-lat-cmp8.crp
==========

```
int main(){
  lat a;
  lat b;
  lat c;

  a = 123456789;
  b = 12345678;
  c = a % b;

  if (a > b){
    print("True \n");
  }
  else{
    print("False \n");
  }

  if (c > b){
    print("True \n");
  }
  else{
    print("False \n");
  }

  return 0;
```

```
}
```

```
===== end of tests/test-lat-cmp8.crp =====
```

```
==============================
 tests/test-lat-cmp8.out
==============================
```

```
True
False
```

```
===== end of tests/test-lat-cmp8.out =====
```

```
==============================
 tests/test-lat-cmp9.crp
==============================
```

```
int main(){
  lat a;
  lat b;
  lat c;

  a = 123456789;
  b = 12345678;
  c = a % b;

  if (b > c){
    print("True \n");
  }
  else{
    print("False \n");
  }

  if (a > a%b){
    print("True \n");
  }
  else{
    print("False \n");
  }

  return 0;
}
```

```
===== end of tests/test-lat-cmp9.crp =====
```

```
==============================
 tests/test-lat-cmp9.out
==============================
```

```
True
True
```

```
===== end of tests/test-lat-cmp9.out =====


==============================
 tests/test-lat-define.crp
==============================

int main(){
  lat a;
  lat b;

  print("True");

  return 0;
}

===== end of tests/test-lat-define.crp =====


==============================
 tests/test-lat-define.out
==============================

True

===== end of tests/test-lat-define.out =====


==============================
 tests/test-lat-div.crp
==============================

int main(){
  lat a;
  lat b;
  lat c;

  a = 123456;
  b = 61728;
  c = a/b;

  if (c == 2) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}

===== end of tests/test-lat-div.crp =====


==============================
 tests/test-lat-div.out
```

```
==============================

True

===== end of tests/test-lat-div.out =====


==============================
 tests/test-lat-int-add.crp
==============================

int main()
{
  lat a;
  int b;

  b = 10;
  a = 123456789;

  if (a + b == 123456799) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}

===== end of tests/test-lat-int-add.crp =====


==============================
 tests/test-lat-int-add.out
==============================

True

===== end of tests/test-lat-int-add.out =====


==============================
 tests/test-lat-int-assign.crp
==============================

int main()
{
  lat a;
  int b;

  b = 10;
  a = b;

  if (a == b) {
    print("True");
  } else {
```

```
    print("False");
  }

  return 0;
}

===== end of tests/test-lat-int-assign.crp =====


==============================
 tests/test-lat-int-assign.out
==============================

True

===== end of tests/test-lat-int-assign.out =====


==============================
 tests/test-lat-int-cmp1.crp
==============================

int main()
{
  lat a;
  int b;

  b = 10;
  a = 123456789;

  if (a != b) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}

===== end of tests/test-lat-int-cmp1.crp =====


==============================
 tests/test-lat-int-cmp1.out
==============================

True

===== end of tests/test-lat-int-cmp1.out =====


==============================
 tests/test-lat-int-cmp2.crp
==============================
```

```
int main()
{
  lat a;
  int b;

  b = 10;
  a = 123456789;

  if (a == b) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}
```

===== end of tests/test-lat-int-cmp2.crp =====

==============================
 tests/test-lat-int-cmp2.out
==============================

```
False
```

===== end of tests/test-lat-int-cmp2.out =====

==============================
 tests/test-lat-int-cmp3.crp
==============================

```
int main()
{
  lat a;
  int b;

  b = 10;
  a = 123456789;

  if (a >= b) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}
```

===== end of tests/test-lat-int-cmp3.crp =====

==============================
 tests/test-lat-int-cmp3.out

```
==============================

True

===== end of tests/test-lat-int-cmp3.out =====


==============================
 tests/test-lat-int-cmp4.crp
==============================

int main()
{
  lat a;
  int b;

  b = 10;
  a = 123456789;

  if (a <= b) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}

===== end of tests/test-lat-int-cmp4.crp =====


==============================
 tests/test-lat-int-cmp4.out
==============================

False

===== end of tests/test-lat-int-cmp4.out =====


==============================
 tests/test-lat-int-cmp5.crp
==============================

int main()
{
  lat a;
  int b;

  b = 10;
  a = 123456789;

  if (a < b) {
    print("True");
  } else {
```

```
    print("False");
  }

  return 0;
}
```

===== end of tests/test-lat-int-cmp5.crp =====

==========
 tests/test-lat-int-cmp5.out
==========

```
False
```

===== end of tests/test-lat-int-cmp5.out =====

==========
 tests/test-lat-int-cmp6.crp
==========

```
int main()
{
  lat a;
  int b;

  b = 10;
  a = 123456789;

  if (a > b) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}
```

===== end of tests/test-lat-int-cmp6.crp =====

==========
 tests/test-lat-int-cmp6.out
==========

```
True
```

===== end of tests/test-lat-int-cmp6.out =====

==========
 tests/test-lat-int-div.crp
==========

```
int main()
{
  lat a;
  int b;

  b = 10;
  a = "1234567890";

  if (a/b == 123456789) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}
```

===== end of tests/test-lat-int-div.crp =====

==========
 tests/test-lat-int-div.out
==========

```
True
```

===== end of tests/test-lat-int-div.out =====

==========
 tests/test-lat-int-mod.crp
==========

```
int main()
{
  lat a;
  int b;

  b = 2;
  a = 123456789;

  if (a%b == 1) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}
```

===== end of tests/test-lat-int-mod.crp =====

==========
 tests/test-lat-int-mod.out

```
==============================

True

===== end of tests/test-lat-int-mod.out =====


==============================
 tests/test-lat-int-mult.crp
==============================

int main()
{
  lat a;
  int b;
  lat c;

  b = 10;
  a = 123456789;

  c = "1234567890";

  if (a * b == c) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}

===== end of tests/test-lat-int-mult.crp =====


==============================
 tests/test-lat-int-mult.out
==============================

True

===== end of tests/test-lat-int-mult.out =====


==============================
 tests/test-lat-int-pow.crp
==============================

int main()
{
  lat a;
  int b;
  lat c;

  b = 2;
  a = 123456;
```

```
  c = "15241383936";

  if (a**b == c) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}
```

===== end of tests/test-lat-int-pow.crp =====

==========
 tests/test-lat-int-pow.out
==========

```
True
```

===== end of tests/test-lat-int-pow.out =====

==========
 tests/test-lat-int-sub.crp
==========

```
int main()
{
  lat a;
  int b;

  b = 10;
  a = 123456789;

  if (a - b == 123456779) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}
```

===== end of tests/test-lat-int-sub.crp =====

==========
 tests/test-lat-int-sub.out
==========

```
True
```

===== end of tests/test-lat-int-sub.out =====

```
==============================
 tests/test-lat-mod.crp
==============================

int main(){
  lat a;
  lat b;
  lat c;

  a = 123456;
  b = 6789;
  c = a%b;

  if (c == 1254) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}

===== end of tests/test-lat-mod.crp =====


==============================
 tests/test-lat-mod.out
==============================

True

===== end of tests/test-lat-mod.out =====


==============================
 tests/test-lat-mult.crp
==============================

int main(){
  lat a;
  lat b;
  lat c;

  a = 12345;
  b = 6789;
  c = a*b;

  if (c == 83810205) {
    print("True");
  } else {
    print("False");
  }
```

```
  return 0;
}
```

```
===== end of tests/test-lat-mult.crp =====
```

```
==============================
 tests/test-lat-mult.out
==============================
```

```
True
```

```
===== end of tests/test-lat-mult.out =====
```

```
==============================
 tests/test-lat-pow.crp
==============================
```

```
int main(){
  lat a;
  lat b;
  lat c;
  lat d;

  a = 123456;
  b = 2;
  c = a**b;

  d = "15241383936";

  if (c == c) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}
```

```
===== end of tests/test-lat-pow.crp =====
```

```
==============================
 tests/test-lat-pow.out
==============================
```

```
True
```

```
===== end of tests/test-lat-pow.out =====
```

```
==============================
 tests/test-lat-sub.crp
==============================
```

```
int main(){
  lat a;
  lat b;
  lat c;

  a = 12345;
  b = 6789;
  c = a-b;

  if (c == 5556) {
    print("True");
  } else {
    print("False");
  }

  return 0;
}

===== end of tests/test-lat-sub.crp =====


==============================
 tests/test-lat-sub.out
==============================

True

===== end of tests/test-lat-sub.out =====


==============================
 tests/test-local1.crp
==============================

void foo(bool i)
{
  int i; /* Should hide the formal i */

  i = 42;
  print("%d\n",i + i);
}

int main()
{
  foo(true);
  return 0;
}

===== end of tests/test-local1.crp =====


==============================
 tests/test-local1.out
==============================
```

```
84

===== end of tests/test-local1.out =====


==============================
 tests/test-local2.crp
==============================

int foo(int a, bool b)
{
  int c;
  bool d;

  c = a;

  return c + 10;
}

int main() {
 print("%d\n",foo(37, false));
 return 0;
}

===== end of tests/test-local2.crp =====


==============================
 tests/test-local2.out
==============================

47

===== end of tests/test-local2.out =====


==============================
 tests/test-md5.crp
==============================

int main() {
      lat a;

      a = hash_md5("hello_world");
      print_lat(a);
}
===== end of tests/test-md5.crp =====


==============================
 tests/test-md5.out
==============================

204296103248535380412210094235203684887
```

```
===== end of tests/test-md5.out =====


==============================
 tests/test-ops1.crp
==============================

int main()
{
  print("%d\n",1 + 2);
  print("%d\n",1 - 2);
  print("%d\n",1 * 2);
  print("%d\n",100 / 2);
  print("%d\n",99);
  print("%d\n",1 == 2);
  print("%d\n",1 == 1);
  print("%d\n",99);
  print("%d\n",1 != 2);
  print("%d\n",1 != 1);
  print("%d\n",99);
  print("%d\n",1 < 2);
  print("%d\n",2 < 1);
  print("%d\n",99);
  print("%d\n",1 <= 2);
  print("%d\n",1 <= 1);
  print("%d\n",2 <= 1);
  print("%d\n",99);
  print("%d\n",1 > 2);
  print("%d\n",2 > 1);
  print("%d\n",99);
  print("%d\n",1 >= 2);
  print("%d\n",1 >= 1);
  print("%d\n",2 >= 1);
  return 0;
}

===== end of tests/test-ops1.crp =====


==============================
 tests/test-ops1.out
==============================

3
-1
2
50
99
0
1
99
1
0
99
```

```
1
0
99
1
1
0
99
0
1
99
0
1
1
```

```
===== end of tests/test-ops1.out =====
```

```
==============================
 tests/test-ops2.crp
==============================
```

```
int main()
{
  print("%d\n",true);
  print("%d\n",false);
  print("%d\n",true && true);
  print("%d\n",true && false);
  print("%d\n",false && true);
  print("%d\n",false && false);
  print("%d\n",true || true);
  print("%d\n",true || false);
  print("%d\n",false || true);
  print("%d\n",false || false);
  print("%d\n",!false);
  print("%d\n",!true);
  print("%d\n",-10);
  print("%d\n",--42);
}
```

```
===== end of tests/test-ops2.crp =====
```

```
==============================
 tests/test-ops2.out
==============================
```

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

```
0
1
0
-10
42

===== end of tests/test-ops2.out =====


==============================
 tests/test-var1.crp
==============================

int main()
{
  int a;
  a = 42;
  print("%d\n",a);
  return 0;
}

===== end of tests/test-var1.crp =====


==============================
 tests/test-var1.out
==============================

42

===== end of tests/test-var1.out =====


==============================
 tests/test-var2.crp
==============================

int a;

void foo(int c)
{
  a = c + 42;
}

int main()
{
  foo(73);
  print("%d\n",a);
  return 0;
}

===== end of tests/test-var2.crp =====


==============================
```

```
 tests/test-var2.out
==============================

115

===== end of tests/test-var2.out =====


==============================
 tests/test-while1.crp
==============================

int main()
{
  int i;
  i = 5;
  while (i > 0) {
    print("%d\n",i);
    i = i - 1;
  }
  print("%d\n",42);
  return 0;
}

===== end of tests/test-while1.crp =====


==============================
 tests/test-while1.out
==============================

5
4
3
2
1
42

===== end of tests/test-while1.out =====


==============================
 tests/test-while2.crp
==============================

int foo(int a)
{
  int j;
  j = 0;
  while (a > 0) {
    j = j + 2;
    a = a - 1;
  }
  return j;
}
```

```
int main()
{
  print("%d\n",foo(7));
  return 0;
}

===== end of tests/test-while2.crp =====


==============================
 tests/test-while2.out
==============================

14

===== end of tests/test-while2.out =====


==============================
 tests/testall.log
==============================

/usr/local/opt/llvm/bin/lli

===== end of tests/testall.log =====
```