

VLC: C'est La Vie

Project Report

Diana Valverde-Paniagua, drv2110
Kellie Ren Lu, krl2130
David Chen, dhc2129

Contents

1	Introduction	6
Background	6	
Related Work	6	
Goals	7	
Ease of use	7	
Familiarity	7	
2	Language Tutorial	8
Compiling and running	8	
Hardware and Software Requirements	8	
Software Requirements:	8	
For Ubuntu Linux:	8	
For MacOS:	8	
For Windows	8	
Using the Compiler	9	
Installing and Uninstalling	9	
Running VLC	9	
Basic VLC Tutorial	9	
1: Creating your VLC source file	9	
2: Declaring a main function	9	
3: Declaring and Assigning Primitive Variables	10	
4: Declaring and Assigning Arrays	10	
5: For Loops	10	
6: Defining CPU Functions	10	
7: Defining GPU Functions	10	
8: Printing Results	11	
9: Putting it all together	11	

3 Language Reference Manual	13
Types	13
Primitive Types and Values	13
Non-Primitive Types	14
Lexical Conventions	14
Whitespace	14
Comments	14
Identifiers	14
Keywords	14
Integer Literals	14
Float Literals	15
Boolean Literals	15
String Literals	15
Separators	15
Operators, Precedence and Associativity	16
Functions	17
CPU Functions	17
GPU Functions	18
Map	18
Program Structure	19
Control Flow	19
Scope	21
Grammar	21
4 Architecture	24
Block Diagram	24
Compiler files	24
Interfaces	25
Library files	25
5 Project Plan	26
Planning Process	26
Test Plan	26
Style Guide	26
6 Lessons Learned	59
Kellie	60
David	60
Diana	60

7 Code Listing	61
Compiler	61
scanner.mll	61
parser.mly	67
semant.ml	75
codegen_c.ml	115
codegen_ptx.ml	125
vlc.ml	129
Makefile	130
Interfaces	131
ast.ml	131
sast.ml	134
exceptions.ml	140
utils.ml	141
processor.ml	154
Library files	155
vlc.hpp	155
Tests	163
test.sh	163
test-arithmetic_ops.vlc	166
test-arithmetic_ops.cu	167
test-print_hello_world.vlc	168
test-print_hello_world.cu	168
test-statements.vlc	169
test-statements.cu	169
fail-Already_declared.vlc	170
fail-Already_declared.vlc.err	170
fail-Array_elements_not_all_same_type.vlc	171
fail-Array_elements_not_all_same_type.vlc.err	171
fail-bad_array_initialization.vlc	172
fail-bad_array_initialization.vlc.err	172
fail-bad_return_type.vlc	173
fail-bad_return_type.vlc.err	173
fail-Boolean_condition.vlc	174
fail-Boolean_condition.vlc.err	174
fail-Cannot_perform_operation_on_array.vlc	175
fail-Cannot_perform_operation_on_array.vlc.err	175
fail-Cannot_perform_operation_on_string.vlc	176

fail-Cannot_perform_operation_on_string.vlc.err	176
fail-Constants_missing_in_defg.vlc	177
fail-Constants_missing_in_defg.vlc.err	177
fail-defg_reinitialize.vlc	178
fail-defg_reinitialize.vlc.err	178
fail-Empty_array_access.vlc	179
fail-Empty_array_access.vlc.err	179
fail-Function_already_declared.vlc	180
fail-Function_already_declared.vlc.err	180
fail-Function_not_defined.vlc	181
fail-Function_not_defined.vlc.err	181
fail-Have_statements_after_break.vlc	182
fail-Have_statements_after_break.vlc.err	182
fail-Have_statements_after_return.vlc	183
fail-Have_statements_after_return.vlc.err	183
fail-Higher_order_function_only_takes_defg.vlc	184
fail-Higher_order_function_only_takes_defg.vlc.err	184
fail-Invalid_accessor_value.vlc	185
fail-Invalid_accessor_value.vlc.err	185
fail-multidec.vlc	186
fail-multidec.vlc.err	186
fail-Name_not_found.vlc	187
fail-Name_not_found.vlc.err	187
fail-nomain.vlc	188
fail-nomain.vlc.err	188
fail-No_strings_allowed_in_gdecl.vlc	189
fail-No_strings_allowed_in_gdecl.vlc.err	189
fail-nomain.vlc	190
fail-nomain.vlc.err	190
fail-Nonarray_passed_into_gdecl.vlc	191
fail-Nonarray_passed_into_gdecl.vlc.err	191
fail-nonconstdefg.vlc	192
fail-nonconstdefg.vlc.err	192
fail-Out_of_scope.vlc	193
fail-Out_of_scope.vlc.err	193
fail-predefined_defg.vlc	194
fail-predefined_defg.vlc.err	194
fail-Type_mismatch.vlc	195

fail-Type_mismatch.vlc.err	195
fail-undefined_defg.vlc	196
fail-undefined_defg.vlc.err	196
fail-uninitialized_variable.vlc	197
fail-uninitialized_variable.vlc.err	197
fail-unmatching_args.vlc	198
fail-unmatching_args.vlc.err	198
fail-unsupported_defg_args.vlc	199
fail-unsupported_defg_args.vlc.err	199
fail-Variable_not_declared.vlc	200
fail-Variable_not_declared.vlc.err	200
fail-Variable_not_found_in_scope.vlc	201
fail-Variable_not_found_in_scope.vlc.err	201
fail-Void_type_in_gdecl.vlc	202
fail-Void_type_in_gdecl.vlc.err	202
fail-wrong_array_type2.vlc	203
fail-wrong_array_type2.vlc.err	203
fail-wrong_array_types.vlc	204
fail-wrong_array_types.vlc.err	204

1. INTRODUCTION

VLC is a syntactically Python-like high level language for GPU(Graphical Processing Unit) programming on NVIDIA GPUs. VLC is primarily intended for numerical computation, which can be performed orders of magnitude faster on parallelizable GPU architecture than on traditional x86 architecture. VLC is intended to provide convenient and safe access to the GPUs computational power by abstracting common lower level operations - for example, data transfer between the CPU and the GPU - from the user. Other functionality provided by VLC include built-in higher order map and reduce functions that utilize the parallel capabilities of a GPU.

Background

GPUs are specialized processors which are composed of hundreds or thousands of small computing units that work in parallel. In the past, GPUs have been used primarily in computer graphics, but recently the capabilities of the GPU are being applied more broadly to computationally heavy applications that benefit from data-parallel acceleration. GPUs operate as a coprocessor to the main CPU, which off-loads some of its computations to the GPU.

On a GPU, the same program is executed on many data elements in parallel. For NVIDIA GPUs, high level language compilers such as CUDA generate virtual PTX (Parallel Thread Execution) instructions. These instructions are then optimized and translated to the native target hardware instruction set.

Related Work

VLC is modeled after the NVIDIA CUDA framework, which allows programmers to utilize both the CPU and GPU to execute programs. CUDA has simplified parallel programming by allowing C, C++, Fortran, and a variety of other languages to compile straight to the GPU without the need for learning assembly or special

tricks for representing general calculations in polygon-based graphics APIs. High level programs can be written in CUDA to get the best features of both worlds: advanced cache mechanisms from the CPU and multi-threaded data-parallelism from the GPU.

Goals

Ease of use

Most GPU programming languages currently require users to be familiar with the complex memory hierarchy and parallel threading models of a GPU, making it inaccessible for programmers who are not familiar with parallel programming concepts or specifics about GPU hardware. VLC vastly simplifies this allowing the user to execute GPU code through two high level functions called *map* and *reduce*. These functions allow the user to define a simple function and apply it to an array of values. Our unique implementation of these higher order functions will allow programmers to execute all possible GPU programs, at the cost of trading efficiency for ease-of-use. This accomplishes two tasks: the core concept of GPU programming (data-level parallelism) is put at the forefront and the programmer will not need to manage any memory transfers to the GPU.

Familiarity

VLC incorporates basic datatypes and a python-like syntax. This allows for programmers who are familiar with C++ or python to easily pick it up.

2. LANGUAGE TUTORIAL

Compiling and running

Hardware and Software Requirements

On the hardware end, a functioning NVIDIA GPU is required. On the software end, OCaml and the CUDA Nvidia Toolkit are required.

Software Requirements:

Ocaml, CUDA, Nvidia Toolkit

For Ubuntu Linux:

```
$ sudo apt-get install ocaml  
$ sudo dpkg -i cuda-repo-ubuntu-1404_7.5-18_amd64.deb  
$ sudo apt-get update  
$ sudo apt-get install cuda
```

For MacOS:

Follow the download instructions for the Nvidia Toolkit for Mac found here¹. If you do not have Homebrew, install it by running the script:

```
$ /usr/bin/ruby -e "$(curl -fsSL  
→ https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Then run:

```
$ brew install ocaml
```

For Windows

Follow the download instructions for the CUDA Nvidia Toolkit for Windows here²

Once everything is installed, clone the git repository to your desired directory:

¹<https://developer.nvidia.com/cuda-downloads>

²<https://developer.nvidia.com/cuda-downloads>

```
$ cd PATH
$ git clone https://github.com/Wumpkins/vlc.git
```

Using the Compiler

Installing and Uninstalling

Change the directory on your terminal or console to PATH/vlc_folder

```
$ make install
```

To uninstall, run:

```
$ make uninstall
```

Running VLC

```
$ vlc [mode] <source_file>
mode:
  -r: compiles and runs source_file
  -c compiles source_file down to CUDA and PTX files in
  ↵ current directory
  -s: prints sast (semantically analyzed abstract syntax
  ↵ tree) to console
  -a: prints ast (abstract syntax tree) to console
  -t: prints tokens read in by scanner
```

Basic VLC Tutorial

In this section, we walk you through creating your first VLC program

1: Creating your VLC source file

Create a new file called `tutorial.vlc` in any directory and open it up.

2: Declaring a main function

All VLC programs must contain a `vlc()` function. The `vlc()` is the first function that gets called by the CPU and determines the rest of the program execution.

```
int def vlc():
```

3: Declaring and Assigning Primitive Variables

In VLC, you declare variables by writing a datatype, followed by an alphanumeric string that begins with an alphabetic letter. The basic data types are `string`, `int`, `float`, `bool`.

```
int def vlc():
    string hello = "Hello World!"
```

4: Declaring and Assigning Arrays

Arrays are declared with the datatype, the size of the array and the identifier for the array. Arrays are then assigned with curly braces surrounding the elements contained in the array.

```
int[5] a = {1, 2, 3, 4, 5}
int[5] b = {1, 2, 3, 4, 5}
```

5: For Loops

A for loop is declared with the keyword `for` followed by parenthesis containing a list of loop iteration parameters and a colon. The first parameter in the loop is the loop iteration variable, the second is the loop termination condition, and the third is the loop afterthought. The afterthought is performed exactly once every time the loop ends, then repeats.

```
/* Sequential add */
for (int i = 0, i < 5, i=i+1):
    c[i] = a[i] + b[i]
```

6: Defining CPU Functions

CPU functions are defined with the keyword `def`. The return type is specified at the beginning of the declaration, followed by `def`, followed by the identifier and a list of parameters enclosed within parenthesis. Parameters are declared the same way that variable declarations are. CPU functions can be called from any other CPU function with the identifier and the list input parameters enclosed by parentheses.

```
int[5] def sequential_add(int[5] a, int[5] b):
    for (int i = 0, i < 5, i=i+1):
        c[i] = a[i] + b[i]
```

7: Defining GPU Functions

GPU functions are defined with the keyword `defg`. The return type is specified at the beginning of the declaration, followed by `defg`, followed by the identifier and

a list of parameters enclosed within parentheses. Parameters of the function are declared the same way that they are in CPU functions. A key difference between CPU and GPU functions is that GPU functions cannot be directly called from any CPU functions, they must be called from within the previously discussed higher order functions of map and reduce. The inputs to these higher order functions are a defg, followed by a list of constants from the current scope that will be used in the defg, followed by the input arrays. All inputs to map must be arrays. map operates by performing the defg on every single element of the input arrays and putting the output in the corresponding indices of the output array. Thus, the type of the input for a defg must be a single element from the input arrays in map.

```
int defg vector_add(int a, int b):
    return s * (a + b)

int def main():
    int[5] a = {1, 2, 3, 4, 5}
    int[5] b = {1, 2, 3, 4, 5}
    int scale = 5

    int[5] d = ~map(vector_add, consts(s=scale), a, b)
```

8: Printing Results

Printing results is as easy as calling the print () function. Print takes in a primitive datatype as an argument.

```
print("hello")
```

9: Putting it all together

```
int defg vector_add(int a, int b):
    return s * (a + b)

int[5] def sequential_add(int[5] a, int[5] b, int s):
    c[5]
    for (int i = 0, i < 5, i=i+1):
        c[i] = s * (a[i] + b[i])

int def main():
    string hello = "Hello World!"
    int[5] a = {1, 2, 3, 4, 5}
    int[5] b = {1, 2, 3, 4, 5}
    int scale = 5

    int[5] c = sequential_add(a, b, scale)
    int[5] d = ~map(vector_add, consts(s=scale), a, b)

    print(hello)
```

```
print("Sequential add: ")
for(int i=0, i<5, i=i+1):
    print(c)

print("Vector add: ")
for(int i=0, i<5, i=i+1):
    print(d)
```

3. LANGUAGE REFERENCE MANUAL

Types

The VLC language has two data types: primitives and non-primitives.

Primitive Types and Values

A primitive type is defined by the conventions listed below and is named by its reserve keyword.

```
bool  
int  
float  
void  
string
```

bool

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

int

An *int* is a 32-bit signed two's-complement integer. An *int* literal can be declared as a sequence of numeric characters ranging from -2,147,483,648 to 2,147,483,647, inclusive.

float

A *float* is a single precision 32-bit IEEE 754 floating point number ranging from 1.4e-45 to 3.4028235e38

void

A *void* datatype can only be used as a return type for functions with no return values.

string

A string is a sequence of alphanumeric characters.

Non-Primitive Types

Arrays

An array holds a fixed number of primitives contiguously in memory. All elements in an array must be of a single type. They are declared by first specifying the type of elements in the array, the size of the array and then the identifier.

Lexical Conventions

Whitespace

Whitespace refers to the space, horizontal tab, form feed and new line characters. White space is used to separate tokens as well as to determine scope. Other than in these uses, it is ignored.

Comments

VLC comments follow the standard comment conventions of C, C++ and Java.

```
COMMENT = /* * + [^*] * + */ | /* */ [^\\n] *
```

Identifiers

An identifier is a case-sensitive sequence of characters consisting of letters, numbers, or underscore, and the first character in an identifier cannot be a number. Identifiers may not take the form of reserved keywords.

```
ID = [a-zA-Z_][a-zA-Z_]*
```

Keywords

Keywords are identifiers that are reserved for use within the programming language. They cannot be re-assigned in a program.

```
int float char bool string if else for while  
continue break return map name def defg consts  
and or not xor true false
```

Integer Literals

An integer constant is an optionally signed sequence of digits.

```
INT = [+ -]?[0-9]+
```

Float Literals

A floating point constant is denoted by an optionally signed integer, a decimal point, a fraction part, an "e" or "E" and an optionally signed exponent. A floating point constant can take the form float. A `float` primitive's absolute value ranges from approximately 1.4E-45 to 3.4E38.

In the declaration of a float, either the fraction part or the integer part must be present, and either the decimal point or the "e" and signed exponent must be present.

```
FLOAT= ['+' '-' ]? ['0'-'9']+ '.' ['0'-'9']*(['e' 'E'] ['+' '-' ]? ['0'-'9']+)?  
| ['+' '-' ]? ['0'-'9']* '.' ['0'-'9']+([ 'e' 'E'] ['+' '-' ]? ['0'-'9']+)?  
| ['+' '-' ]? ['0'-'9'] ['e' 'E'] ['+' '-' ]? ['0'-'9']+?
```

Boolean Literals

A boolean has two possible values, true or false. These are denoted by the identifiers "true" and "false".

```
BOOL = 'true' | 'false'
```

String Literals

A string constant is denoted by enclosing double quotes "", and can be constructed from alphanumeric characters, traditional punctuation characters, and the specified valid escape characters.

- '\"' - single quote
- '\"'' - double quote
- '\\' - backslash
- '\\n' - newline
- '\\r' - carriage return
- '\\t' - tab

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

Separators

A separator is a character that separates tokens. White space is also used as a separator, unless it is defining scope.

' ({ LPAREN }
')	{ RPAREN }
' :'	{ COLON }
' [{ LBRACKET }
']'	{ RBRACKET }
' .'	{ DOT }
' ,'	{ COMMA }

Operators, Precedence and Associativity

Operators are reserved characters that are applied to one or two primitives in the language. Details about operator precedence and uses are defined in the following section.

+	-	*	/	%
>>	<<	++	--	&
	xor	and	or	not
==	!=	>	<	>=
<=	=			

The following sections will describe all operators, with each subsequent section explaining a set of operators with lower precedence than the previous.

Arithmetic Operators

There are nine basic arithmetic operators in VLC: *addition*, *subtraction*, *multiplication*, *division*, *modulo*, *bitshift right*, *bitshift left*, *increment-by-one*, *decrement-by-one*, *bitwise-and*, *bitwise-or* and *xor*. All arithmetic operators are left associative, with multiplication and division having higher operator precedence than addition and subtraction, and addition and subtraction having higher operator precedence than bitwise operators.

Logic Operators

Logic operators operate on expressions which evaluate to boolean values. The following three logical operators are used in VLC: *and*, *or* and *not*. The *and* operator is a binary operator which returns *true* if both of its operands evaluate to *true*, otherwise it returns *false*. The *or* operator is a binary operator which evaluates to *true* if either of its operators are *true*, otherwise it returns *false*. The *not* operator is a unary operator which returns *true* if the operand evaluates to *false* and *false* if the operand evaluates to *true*.

Relational Operators

Relational operators compare the values of two expressions. VLC has the following six relational operators: *equivalence*, *non-equivalence*, *greater-than*, *less-than*, *greater-than-or-equal-to*, and *less-than-or-equal-to*. These operators return a boolean value which can be used within conditional statements to control execution of code.

Array Access Operator

The double brackets [] are used to denote a right-associative access of the array label immediately to the left, where the expression within the brackets has to be of type in. Array access operators return the *i*th element in the array, where *i* is the integer that the expression within the bracket evaluates to.

Assignment operator

The *assignment* operator is a right-associative binary operator which evaluates the value of the right hand side of the operator and stores it in the left-hand side. Only identifiers, variable declarations and array expressions will be accepted on the left-hand side of an assignment operator.

Functions

There are two kinds of functions in VLC, CPU functions and GPU functions.

CPU Functions

CPU functions are declared using the *def* keyword, and must specify their arguments and argument types, return type, and end with a colon. Functions cannot be declared within other functions.

The scope of a function is defined by whitespace - that is, all statements that are part of the function cannot be aligned with the function declaration, but must be "indented", or prefaced by at least one whitespace character to be defined within the function scope.

All function arguments that are primitive types are passed by value, meaning all arguments are copied to the function. This means that changes to the argument within the function will not change the argument's value outside of the function.

All function arguments that are non-primitive types are passed by reference, meaning changes to the argument will change the argument's value outside of the function.

Declarations

```
<return type> def <function name>(<type1> arg1, <type2>
↪ arg2...):
```

Calls

```
<function name>(<type1> arg1, <type2> arg2...)
```

GPU Functions

The GPU function *defg* creates a user-defined function that is meant to be run on the GPU kernel. *defg* functions must specify arguments and argument types, return type, and end in a colon. A *defg* function cannot be declared within other functions and may not call other functions. These *defg* functions will be called by the higher-order function *map*.

There are N array-dependent arguments that must be declared in *defg* that will be called by *map*, taking N arrays as input. Each array-dependent argument is an identifier for a single element in the array(s) that are being handled by *map* and *reduce*.

Besides the identifiers of its arguments, *defg* function body may also reference const arguments that are passed to the higher order map function that takes the *defg* function as an input. (See the example for *map* to understand how to reference const arguments in *defg*)

Declaration

```
<return type> defg <function name> (<type1> arg_1,  
→ <type2> arg_2...):
```

Map

VLC contains the built-in higher order function *map* which takes a *defg*, constants and arrays as arguments. These built-in higher order functions provide needed abstraction for users who do not wish to be bogged by the specifics of GPU computing but still want to take advantage of GPU parallelism.

The first parameter in a *map* function must be a *defg*. An optional parameter to *map* is a list of declared constant arguments defined by

```
const=[<type1> const_arg1=value1, <type2>  
→ const_arg2=value2... ] }
```

These *const* arguments define variables that can be referenced by the *defg* input function; subsequently, *defg* can reference constant arguments by calling them by the same name declared in the *const* list. Also note that constant arguments must not only be declared, but also assigned a value. In the GPU, these constant arguments

are copied from host to the global memory in the GPU kernel, allowing all threads in the kernel to access these variables. For the remaining parameters, map may take a variable number of arrays so long as they all have the same dimensions. Further, if the input arrays are multi-dimensional, each dimension must have fixed-length rows.

The output of map is an N-dimensional array of the same size as all the input arrays to map, where *defg* has been applied to the element in the corresponding index as the output.

Usage

map is a reserved keyword and may not be used by the user to define any other variable, constant, or function. Further, the *defg* functions passed to map and reduce:

1. Must have the corresponding number of arguments specified by map.
2. Must have arguments that are the same type as the the array(s) passed into map and reduce. In the case of map, the order of the argument types to *defg* should match the order of the arrays inputted into map.
3. May reference by name any const arguments that are passed to map or reduce by using their identifier. For example, in the example below, the function <function name> can reference the constant *arg1* by simply calling *arg1*. If a user defines a variable in *defg* with the same name as a constant argument to map, the defined variable will override the reference to the constant argument.

Function Call

```
~map(<function name>, const=[<type1> arg_1, ...]  
→ <input array> ...>
```

Program Structure

Any statements at the beginning of the program outside of function definitions are in the global scope of all CPU functions. A program consists of zero or more variable declarations followed by one or more function declarations. The starting execution point for a VLC program is the required main() function.

Control Flow

If Else Statements

VLC uses standard *if* and *else* control statements. These control statements take a boolean expression as input, and execute branching according to the value of

the boolean expression. An *if* may be followed by optional *else* statement, and *if* need not be concluded with an *else*. Furthermore, every *if* and *else* block defines a new scope, which is determined by white space characters. *if* and *else* and *else* can also be nested in other *if* and *else* statements.

The below example demonstrates proper use of *if* *else* loops.

```
int a = 5
if (1 == 1):
    a = 1
else
    a = 2
```

While Loops

VLC supports traditional *while* loops that take a boolean expression condition as an input. The substatements within the scope of a while loop are repeated so long as the condition evaluates to true. Scope within a while loop is defined by white space characters. See White Space section for further clarification. Users can break out of a *while* loop using the *break* keyword, or skip to the next iteration of a *while* loop using the *continue* keyword.

A while loop in VLC has the following syntax:

```
int a = 0
while ( a < 20 ):
    if ( a % 2 == 0 ):
        a = a + 3
    continue
    a = a + 1
```

For Loops

For loops in VLC take as input an iterator assignment, a boolean expression condition, and an iterating statement each separated by a comma. Scope within a for loop is defined by preceding white space characters. See White Space section for further clarification. The substatements within the for loop will execute if condition is true, with the next iteration of the loop increasing the iterator defined in the iterator assignment by the iterating statement.

Users can break out of for loop iteration using the *break* keyword, or skip to the next iteration of a for loop using the *continue* keyword. In essence, VLC supports traditional for loops that follow the below structure:

```
int a = 0
for ( int i = 0, i < 10, i++ ):
    a = a + i
```

Scope

Scoping in VLC is static, and follows the conventions of block-level scoping. Variables defined at the top level of a program are available in the global scope of the program.

Grammar

```
let letter = ['a'-'z' 'A'-'Z']
let digit = ['0'-'9']
let whitespace = [' ' '\t']
let sign = ['+' '-']
let exp = ['e' 'E']
let newline = '\n' | "\r\n"

rule token = parse
  | whitespace* "://" { single_line_comment }
  ↵ lexbuf }
  | whitespace* "/*" { multi_line_comment }
  ↵ lexbuf }
  | newline { indent lexbuf }
  | whitespace { token lexbuf }

(* Punctuation *)
| '(' { LPAREN }
| ')' { RPAREN }
| ';' { COLON }
| '=' { ASSIGNMENT }
| '[' { LBRACKET }
| ']' { RBRACKET }
| '{' { LCURLY }
| '}' { RCURLY }
| ',' { COMMA }

(* Arithmetic Operators *)
| '+' { ADD }
| '-' { SUBTRACT }
| '*' { MULTIPLY }
| '/' { DIVIDE }
| '%' { MODULO }
| ">>" { BITSHIFT_RIGHT }
| "<<" { BITSHIFT_LEFT }
| "++" { PLUS_PLUS }
| "--" { MINUS_MINUS }
| "&" { BITWISE_AND }
| "|" { BITWISE_OR }

(* Logic Operators *)
| "and" { AND }
| "or" { OR }
```

```

| "not"      { NOT }
| "xor"      { XOR }

(* Comparison Operators *)
| "=="       { EQUAL }
| "!="       { NOT_EQUAL }
| ">"        { GREATER_THAN }
| ">="       { GREATER_THAN_EQUAL }
| "<"        { LESS_THAN }
| "<="       { LESS_THAN_EQUAL }

(* Datatypes *)
| ("string"
| "bool"     | "void"
| "ubyte"    | "byte"
| "uint"     | "int"
| "ulong"    | "long"
| "float"    | "double") as input { DATATYPE(input) }

(* Conditionals and Loops *)
(* / "elif"      { ELSEIF } *)
| "if"        { IF }
| "else"      { ELSE }
| "for"       { FOR }
| "while"     { WHILE }
| "break"     { BREAK }
| "continue"  { CONTINUE }

(* Function Declarations and Attributes *)
| "~"         { TILDA }
| "return"    { RETURN }
| "def"        { DEF }
| "defg"       { DEFG }
| "consts"    { CONSTS }

| ("true" | "false") as booleanlit
→                                {
→      BOOLEAN_LITERAL(bool_of_string booleanlit)
| "''" ((['-' '! '#' '-' & '(' '-' '[' ']'-'^']) | '\\' [ '\\''
→      "'n' 'r' 't' '''])* as stringlit) "'"
→          { STRING_LITERAL(stringlit) }
| digit+ as intlit           { INTEGER_LITERAL(int_of_string
→      intlit) }
| (digit+ '.' digit* | '.' digit+ | digit+ ('.' digit*)?
→      'e' '-'? digit+ | '.' digit+ 'e' '-'? digit+) as fsplit
→          { FLOATING_POINT_LITERAL(float_of_string fsplit) }
| (letter | '_')(letter | digit | '_')* as id {
→      IDENTIFIER(id) }
| eof                      { get_eof() }

(* Blocks for comments *)
and single_line_comment = parse

```

```

| newline                  { indent lexbuf }
| eof                     { get_eof() }
| _                      { single_line_comment lexbuf }

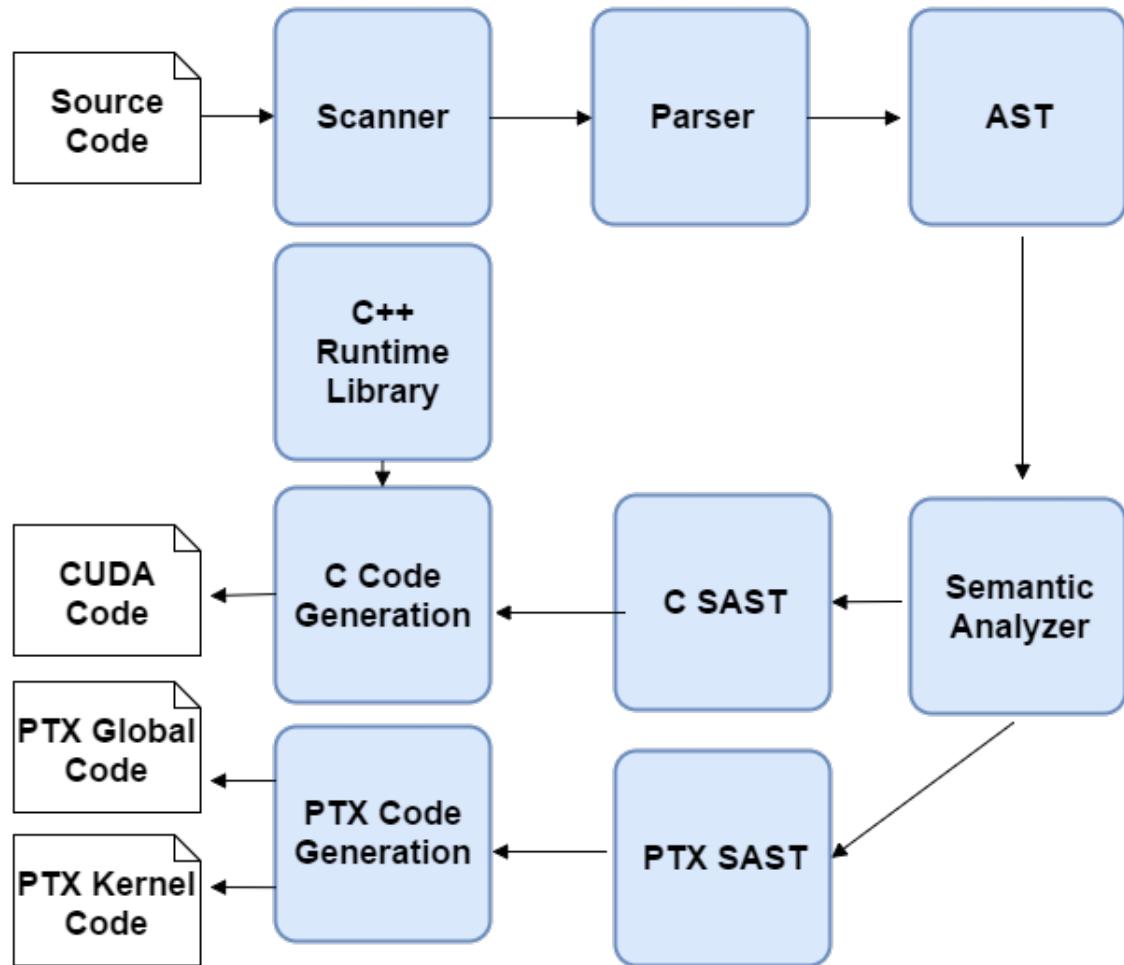
and multi_line_comment = parse
| newline                 { multi_line_comment lexbuf }
| /* */
| _                      { multi_line_comment lexbuf }

(* Block for handling white space delimiting *)
and indent = parse
| whitespace* newline     { indent lexbuf }
| whitespace* eof         { get_eof() }
| whitespace* as indentation
{
    let indent_length = (String.length indentation)
→ in
    let stacktop_length = (Stack.top indent_stack) in
    if indent_length > stacktop_length then
        begin
            Stack.push indent_length indent_stack;
            INDENT
        end
    else if indent_length = stacktop_length then
        TERMINATOR
    else
        let count =
            (* Function that pops indent lengths from the
→ stack until we reach the appropriate indent length *)
            let rec popped_from_stack counter =
                if (Stack.top indent_stack) >
→ indent_length then
                begin
                    ignore(Stack.pop indent_stack);
                    popped_from_stack (counter + 1)
                end
            else if (Stack.top indent_stack) <
→ indent_length then -1
            else counter
                in popped_from_stack 0
            in
            if count = - 1 then raise
→ (Exceptions.Bad_dedent)
            else DEIDENT_COUNT(count)
        }
    {
        Stack.push 0 indent_stack
    }
}

```

4. ARCHITECTURE

Block Diagram



Compiler files

- **codegen_c.ml:** This module converts a semantically checked AST into CUDA C code. This file is responsible for generating all c functions and memory transfers to and from the GPU as well as reading and instantiating the generated PTX modules.
- **codegen_ptx.ml:** This module a semantically checked AST into PTX instruc-

tions. This file is responsible for generating all *defg* kernels as well as generating the global kernel from which to call them.

- **parser.ml:** The parser in tokens from the scanner to produce an AST.
- **processor.ml:** This is a helper file for the parser. It reads in tokens from the scanner and helps parse white space.
- **scanner.ml:** The scanner reads the input file and produces tokens representing the language.
- **semant.ml:** This file is responsible for all of the semantic analysis in the language, verifying the validity of the AST by scope and type checking. This file is responsible for separating CPU and GPU code and generating the GPU-specific symbols such as register declarations, load and store instructions. This file also converts AST types to the appropriate variable types of their respective language. If the program passes through the semantic checker, it produces two SASTs, one for CUDA and one for PTX which are then generated by their respective generators.
- **utils.ml:** Contains general helper functions that are used throughout the compiler. It converts all intermediate representations of the program to strings, which is used to debug things in the parser and AST and SAST.
- **vlc.ml:** This module calls all the other modules.

Interfaces

- **ast.ml:** Takes in a sequence of tokens and generates an *Abstract Syntax Tree* from the grammar declared in the parser and Ast.
- **exceptions.ml:** All of the exceptions that can be raised in our compiler can be found in this file.
- **sast.ml:** Contains SAST type definitions for conversions during semantic analysis.

Library files

- **vlc.hpp:** This the run-time library used to create arrays within CUDA. The compiler requires array support to ensure that all array declarations are generated on the heap, so that memory transfers between the GPU and CPU can be easily done. The array library also flatten multi-dimensional arrays, which allows for more flexibility in the syntax of our language.

5. PROJECT PLAN

Our original Project Manager, Chance, left the class near the beginning of the semester. As Professor Edwards told us in a subsequent e-mail, this meant we no longer had a chance on this project. So, with neither a chance nor dedicated project manager on our side, this ended up being our project plan:

Planning Process

Our team met once or twice a week to discuss the project and collaborate in-person. Group collaboration consisted of discussions, pair-coding, as well as group work on a single compiler version and programming environment through an online collaborative programming editor called Madeye. This process was crucial to the development of our compiler because only one of our team members had a NVIDIA GPU. For individual work on the project we used Git as a distributed version control system, allowing all members of the group to create their own branches of the project and work independently.

Test Plan

The majority of the testing throughout the projects was done by making sure that each compiler version could compile and run the files in the sample-generated-files folder - there are even specific commands in our Makefile to print out the tokens, Ast, Sast and to Compile each of these files. These files are representative of the essential features in our language. Later on, we discovered that these tests were not enough, so a complete test suite was created to test specific features of the language and specific exceptions we would expect from the compiler. The test suite for VLC consists of simple language feature and semantic-checking exception tests as well as a few longer programs which tie everything together.

Style Guide

We frequently worked on a single collaborative environment, so a style guide never needed to be formally set. However, over time we developed a consistent approach to coding style as follows:

1. Maximum length of a single line must not exceed 80 characters.
2. Each code block following a Let.. in statement must be indented.

-
3. Underscore casing for all variable and function names.
 4. Fully written names for variable and function names except in cases the full names would be too long. This was for clarity and readability of the code.
 5. Capital letters for AST and SAST types, lower case letters for all other names.

Our git log is displayed below:

```
commit fead0ba89b2af305c1e4e085f7e0b8167d15dbed
Merge: 594a616 66c3ab6
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Fri May 13 08:02:47 2016 -0400

Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie

commit 594a6169146d958cba534f44eca3beb39e49b38a
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Fri May 13 08:02:33 2016 -0400

final

commit 4c38175a8ce46539d6065308348d4daeb7d5d5c4
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Fri May 13 07:06:22 2016 -0400

fixed datatype

commit 66c3ab680d1a657f6e0e6fc0d20444205c23fd38
Merge: bb24a21 53ad183
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Fri May 13 07:03:25 2016 -0400

Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie

commit 53ad183eab2eaf99717ca28062bc5e5a4e5c9ae0
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Fri May 13 07:03:11 2016 -0400

fixed last part

commit bb24a214bd688b52dd4496debc6cdcd319fd21f9
Merge: f9b084c 1cb377b
```

Author: Wumpkins <dhc2129@columbia.edu>
Date: Fri May 13 06:55:37 2016 -0400

Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie
commit 1cb377b86716ea01bacd41e9835942d520f5c5b4
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Fri May 13 06:55:29 2016 -0400

final?

commit f9b084c0b57ead7cfef08852eaca68ae2ac5468c
Merge: 42083bf 1d2ee99
Author: Wumpkins <dhc2129@columbia.edu>
Date: Fri May 13 05:22:41 2016 -0400

Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie
commit 1d2ee99702d90d89fe235a7f4e31de6166bb8860
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Fri May 13 05:22:08 2016 -0400

changed va args

commit 42083bf14729b79d49765bb046a776eb6c68ea10
Merge: 1ef8126 66e081e
Author: Wumpkins <dhc2129@columbia.edu>
Date: Fri May 13 03:11:28 2016 -0400

Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie
commit 66e081e0d1a41715900ca79b72718d724004bada
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Fri May 13 03:00:04 2016 -0400

fixed faulty quote

commit 1ef8126933258caa3a076a2ea23354285157de7f
Merge: 5844449 864559b
Author: Wumpkins <dhc2129@columbia.edu>
Date: Fri May 13 02:52:28 2016 -0400

```
Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie
commit 864559b1f978922b8061669bfcb1c9c8fd1cfdf
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Fri May 13 02:52:02 2016 -0400
```

added check errors

```
commit 5844449a5d10d4a181c81e5a02ae7fea27620887
Merge: a4e8913 c332eb8
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Fri May 13 02:47:57 2016 -0400
```

```
Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie
commit c332eb85e09a503a563dbad1d5e9f9d5d0ca0207
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Fri May 13 02:08:39 2016 -0400
```

fixed else

```
commit a4e89135ce301eb643820501962b8c7dc989c16
Merge: ded3bf2 1228b6f
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Thu May 12 23:06:08 2016 -0400
```

```
Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie
commit 1228b6fb1addbe57e85bad82ca9b244289496f78
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Thu May 12 23:05:44 2016 -0400
```

working expressions and assignments

```
commit ded3bf287e4ffe783a72f2108260835de3e9eb9b
Merge: 6455913 4ff0c9e
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Thu May 12 18:27:22 2016 -0400
```

```
Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie
```

```
commit 4ff0c9ee0ad4294f8f0c6cdbfd04486dccc0dec  
Author: Kellie Ren Lu <krl2130@columbia.edu>  
Date: Thu May 12 14:23:28 2016 -0400
```

assignment working for literals, but not tet for arrays

```
commit ce261eba2ac86583dbd8a6a900a409005d296671  
Author: Kellie Ren Lu <krl2130@columbia.edu>  
Date: Thu May 12 14:10:00 2016 -0400
```

declarations and initializatiobs generate

```
commit 64559136e29ace0139f7afe325010537f6486482  
Merge: 29c075f da254ee  
Author: Wumpkins <dhc2129@columbia.edu>  
Date: Thu May 12 13:57:35 2016 -0400
```

Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie

```
commit da254ee753db4567185fb79b9f07fd50f3a5964f  
Author: Kellie Ren Lu <krl2130@columbia.edu>  
Date: Thu May 12 13:48:44 2016 -0400
```

fixed cpp memory issue

```
commit 29c075fb60dcb1d0e0d9a7f78d513b68bb9bfd3b  
Merge: eb0dea3 e19a99e  
Author: Wumpkins <dhc2129@columbia.edu>  
Date: Thu May 12 12:34:09 2016 -0400
```

Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie

```
commit e19a99e2518501e6e2ec65baa0e61dacd1828eef  
Author: Kellie Ren Lu <krl2130@columbia.edu>  
Date: Thu May 12 12:31:00 2016 -0400
```

added load statements for params

```
commit eb0dea36a587c52ba063d7d332c89d2b476ed60a  
Author: Wumpkins <dhc2129@columbia.edu>
```

Date: Thu May 12 12:20:10 2016 -0400

null

commit 8a1fe5415fac41325f88307347fad879cf55cd11

Merge: 6e5c7f9 55242b5

Author: Wumpkins <dhc2129@columbia.edu>

Date: Thu May 12 12:19:46 2016 -0400

Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie

commit 55242b5d5917345e3a78cf766f0edd11f006bled

Author: Kellie Ren Lu <krl2130@columbia.edu>

Date: Thu May 12 12:10:38 2016 -0400

working binop - sort of - need to make sure that literals become assignments

commit 99d5dd9f5b83f50d7771f4228d13b68b906879ab

Author: Kellie Ren Lu <krl2130@columbia.edu>

Date: Thu May 12 11:53:50 2016 -0400

added expressions and compiles, haven't tested yet

commit 919fc8ea51cbccab037597bc08a93a35eb58273

Author: Kellie Ren Lu <krl2130@columbia.edu>

Date: Thu May 12 00:22:36 2016 -0400

before major changes

commit 6e5c7f9c299c0380805ae5a30ff56de3d3a56ddb

Merge: 3a3f53e 9c79ccb

Author: Wumpkins <dhc2129@columbia.edu>

Date: Wed May 11 22:07:34 2016 -0400

Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie

commit 9c79ccb3dbe642a8bc8c7d6a3e554eda04d7b6ab

Author: Kellie Ren Lu <krl2130@columbia.edu>

Date: Wed May 11 21:14:33 2016 -0400

added diana's test suite

```
commit 569d9e6a7844a6657d2c243e9e6b8fffcf140a91
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Wed May 11 20:13:59 2016 -0400
```

working map

```
commit 29033493164ba334590041d3a5ff3bea5d7193e9
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Wed May 11 18:24:17 2016 -0400
```

final version of global map

```
commit 4898b835c99d6a52aafa218fa167857ad6f6bd8e
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Wed May 11 17:06:38 2016 -0400
```

commiting to fix syntax error2

```
commit 67f0f5479252502d6826ac45bfaaffaec0bee55b
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Wed May 11 17:06:29 2016 -0400
```

commiting to fix syntax error

```
commit d0244af21c1a1b403c86346423cbe87975ffb2f8
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Wed May 11 15:53:35 2016 -0400
```

map halfway working

```
commit d17afa086226da414951384b237ca40005cd103d
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Wed May 11 12:00:36 2016 -0400
```

c part working

```
commit 3a3f53e101b086aa3730a558f4799c0244ca4053
Merge: 075b279 1991747
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Tue May 10 20:21:37 2016 -0400
```

```
Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie
commit 075b279e3201462ba76b9e19e02f3d12b618e6fe
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Tue May 10 20:21:34 2016 -0400
```

defg test

```
commit 19917479d1199a140ff866164f38e05448d1c5e5
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Tue May 10 20:21:26 2016 -0400
```

last push

```
commit 467e91cf23a6a3edf8b7881e6498c127edff08db
Merge: 2044f51 58e45ed
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Tue May 10 19:38:56 2016 -0400
```

before presentation

```
commit 2044f51d502919e49d6f626821672c8b2c424b51
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Tue May 10 19:11:19 2016 -0400
```

more progress

```
commit 58e45ed9cd0c84fab3142e7504ca9fce7ac60675
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Tue May 10 18:57:08 2016 -0400
```

binary op initialization working

```
commit 14e4fc5c376024467bda9bd85d4ed6a2c1e11c66
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Tue May 10 16:17:32 2016 -0400
```

before group work

```
commit 6163c074f6b5702b7978543f7d9b5e4b9ab9970d
```

Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Tue May 10 13:00:35 2016 -0400

c arrays and cuda working

commit e9a42c930451653063c6f4f46f5c9d655b87ce95
Merge: 1313f6f 9edc435
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Tue May 10 03:50:13 2016 -0400

Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie

commit 1313f6f9e662c3170572a6f62fd07e3a5af501ed
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Tue May 10 03:50:08 2016 -0400

vlc arrays and cuda generation working

commit 9edc43597969b76c0c3f8955da37439cb53e3ff0
Author: Wumpkins <dhc2129@columbia.edu>
Date: Tue May 10 03:45:15 2016 -0400

added initialization for several cases

commit 74625551be8395d0d1bf71cf794ae3b30ff043ba
Author: Wumpkins <dhc2129@columbia.edu>
Date: Tue May 10 01:50:22 2016 -0400

ptx code generation formatting

commit 78a86dc29c4e1a70f7e4da423115f1aa7b972731
Author: Wumpkins <dhc2129@columbia.edu>
Date: Tue May 10 01:26:14 2016 -0400

added register decls at beginning of file

commit 8f5ee0aab0bf4dc284bac534aaafdaedf381212e
Merge: 1bec61c b77a983
Author: Wumpkins <dhc2129@columbia.edu>
Date: Tue May 10 01:11:39 2016 -0400

```
gerge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie
```

```
commit 1bec61ce4e9a286d695f69cb837f31030b25c89e
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Tue May 10 01:11:26 2016 -0400
```

```
semantic done (sort of) with compiling ptx
```

```
commit b77a983068c5a35dfc0cac2f844e54b94b0e5e44
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Mon May  9 23:00:26 2016 -0400
```

```
editing vlc array
```

```
commit ee7151df05fdea022cde640f9bc567abb5905189
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Mon May  9 18:43:46 2016 -0400
```

```
same as before
```

```
commit e061cf915b32d92c10958209a0d186df8c9ab0c5
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Mon May  9 18:43:21 2016 -0400
```

```
renamed test files working on test script
```

```
commit 036befd0bf1e751be64e80d2aa05113f65c5808a
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Mon May  9 17:03:36 2016 -0400
```

```
vlc_array finished
```

```
commit 443fb4c1d4a129043acdfb9ff34b003c5572b563
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Mon May  9 15:47:10 2016 -0400
```

```
changed vlc array
```

```
commit 4e89fb40123f8ec86f438761c78068e044bc65d1
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Mon May  9 14:33:10 2016 -0400
```

updated map/reduce generation

```
commit 9e23bb0084fc000e95ab586b2c0f9e782c6e7ec9
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Mon May  9 12:42:16 2016 -0400
```

added more checks in semant, still have more to go

```
commit 2a55d1969191c6669e8ff197e2be93886fb7f434
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Mon May  9 02:14:36 2016 -0400
```

compiling works

```
commit 6daee80303ef593e701bfe654966586511d090de
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Mon May  9 01:58:25 2016 -0400
```

hello world reads successfully

```
commit e35a96adb220f8425ad4210c30fce4275190a5d3
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Mon May  9 01:39:24 2016 -0400
```

fixed more semant errors

```
commit ac521dd1b727055055871cf5f220520cc56dc141
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Mon May  9 01:34:46 2016 -0400
```

fixed a semant logic error- still debugging

```
commit c20d7070689f4b16dc2826e71644d11a4970a564
Merge: 0b911e5 08725d6
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Mon May  9 01:12:52 2016 -0400
```

Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie

```
commit 0b911e5ef87a07676121ebfccd129fb39ca21411
```

Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Mon May 9 01:12:38 2016 -0400

fixed parse error

commit 08725d6d80d664de842a423d390cb44a057df531
Merge: 7f9782f df4f9cc
Author: Wumpkins <dhc2129@columbia.edu>
Date: Mon May 9 00:58:53 2016 -0400

merge conflict

commit 7f9782fb0583b06981fd2b75c6068b3ede2cc83a
Author: Wumpkins <dhc2129@columbia.edu>
Date: Mon May 9 00:56:30 2016 -0400

some more work on ptx conversion

commit df4f9cc0207485bbc40e75e2a3c1a7dc4f431558
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Mon May 9 00:38:06 2016 -0400

added exception

commit e949d0a5cc0ee748698708650c115e739e4c29d0
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Mon May 9 00:23:03 2016 -0400

compiles but doesn't work

commit 4d724b80ab5d53c3da9546837cf25f64a2bf3580
Author: Wumpkins <dhc2129@columbia.edu>
Date: Mon May 9 00:08:16 2016 -0400

fixed some bugs

commit 543008d0b6e5d1fbf808b02175635be037452f18
Merge: 9d7f5d7 4a1cfae
Author: Wumpkins <dhc2129@columbia.edu>
Date: Sun May 8 23:54:25 2016 -0400

```
gerge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie
```

```
commit 4a1cfaed85bf0d77541bca3274da467682f25f76
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Sun May  8 23:32:38 2016 -0400
```

```
semant compiles, codegen still doesn't
```

```
commit e772a3a1ee6079f911fcd6073f046048afd39510
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Sun May  8 20:03:19 2016 -0400
```

```
debugging c semant, c code generation
```

```
commit 6e1c1d3dd57abb700c1d7d9aba82fc5157b1cf0c
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Sun May  8 17:13:29 2016 -0400
```

```
added hof generation - but codegen for it still incomplete
```

```
commit 9d7f5d7dd264c88be783739490eaf79134f9f9ed
Merge: 2501b37 5e780f1
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Sun May  8 07:01:55 2016 -0400
```

```
mergg branch 'kellie' of https://github.com/Wumpkins/vlc into kellie
```

```
commit 2501b3706d5f614c0bdb7e564adcfb8fd19c416a
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Sun May  8 07:01:53 2016 -0400
```

```
some changes for semant
```

```
commit 5e780f1823a55524644ce37577851fcf945a2b33
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Sun May  8 07:00:10 2016 -0400
```

```
group work with semant and ptx sast
```

```
commit 7e43920a729e6f9e1721e58cd459b1ba4e7479b8
Author: Kellie Ren Lu <krl2130@columbia.edu>
```

Date: Sun May 8 06:45:11 2016 -0400

added more semant.ml, more specific ptx sast

commit bd563a21b7d340749dedf3af94d661cc3c89d785

Merge: d26b36c e972b25

Author: Kellie Ren Lu <krl2130@columbia.edu>

Date: Sun May 8 03:42:48 2016 -0400

fixed merge before group work on semant

commit d26b36c0bd436c4953ddb41848f865e089523d84

Author: Kellie Ren Lu <krl2130@columbia.edu>

Date: Sun May 8 03:40:12 2016 -0400

updated semantic analyzer for c

commit e972b25a2bfeaca152ce86ede8b95a2e2a980064

Author: Wumpkins <dhc2129@columbia.edu>

Date: Sun May 8 03:39:45 2016 -0400

register counts

commit fa31b786ecb50bb22e5dec0930ec5be9dcc5c070

Author: Wumpkins <dhc2129@columbia.edu>

Date: Sun May 8 03:26:15 2016 -0400

merge

commit 8b9e951429bcd71c4f49e70e7a8a3eb203df6ff5

Merge: b041393 4df19b3

Author: Wumpkins <dhc2129@columbia.edu>

Date: Sun May 8 03:23:18 2016 -0400

update gitignore

commit b041393cef9ca819f5e073ddad6b080a5ff8b44

Author: Wumpkins <dhc2129@columbia.edu>

Date: Sun May 8 03:20:49 2016 -0400

finished ptx sast... for now

```
commit 4df19b3f7fd45a34f011bfa1893eb8fbe007d3d8
Author: dianarvp <dianarvp@gmail.com>
Date:   Sun May  8 03:17:58 2016 -0400
```

Template for final report

```
commit d7481e19e70c16fc0593c907cde764aefef729946
Merge: 674be03 e31c5d4
Author: dianarvp <dianarvp@gmail.com>
Date:   Sat May  7 23:58:41 2016 -0400
```

Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie

```
commit 674be03ffa8fb74ce711982e4d049eaacb3f24fe
Author: dianarvp <dianarvp@gmail.com>
Date:   Sat May  7 23:57:03 2016 -0400
```

Added tests

```
commit 123d8be82d0a850865e15e8a619c944f8f787709
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Sat May  7 23:55:33 2016 -0400
```

mid edit - fixing VLC Array and C map generation

```
commit e31c5d4f5fc782c365b213c653802b8e91524618
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Sat May  7 23:25:48 2016 -0400
```

refined sast and codegen for existing material as well as finishing some

```
commit a9099849ce5460f3f6c683500f24316d5e6a9fd0
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Sat May  7 13:03:31 2016 -0400
```

everything so far

```
commit ee04d8badd3ce79d94a2b4fa9a262081a783c16b
Merge: cf800b4 d7054e8
Author: Kellie Ren Lu <krl2130@columbia.edu>
```

Date: Mon May 2 02:24:42 2016 -0400

new merge

commit cf800b4c15215bbef091ce119d4d9ef15c15291

Merge: ec08c8d 3f67bc4

Author: Kellie Ren Lu <krl2130@columbia.edu>

Date: Sun May 1 20:02:38 2016 -0400

merged

commit d7054e857c9096c7e7bdef258436b5132c63f611

Author: Wumpkins <dhc2129@columbia.edu>

Date: Sun May 1 19:59:37 2016 -0400

added data movement to sast

commit ec08c8d2341b380442b360655514f3b8dd4a6dd8

Author: Kellie Ren Lu <krl2130@columbia.edu>

Date: Sun May 1 19:58:01 2016 -0400

before pull

commit 3f67bc40387bbcb1c06034fb8c2bab2a0c009606

Author: Wumpkins <dhc2129@columbia.edu>

Date: Sun May 1 19:39:36 2016 -0400

added register decl

commit af6468db4eb6195d406222b7001512c3f465ae84

Author: Wumpkins <dhc2129@columbia.edu>

Date: Sun May 1 19:17:19 2016 -0400

added some comments regarding codegen

commit a2fcfb3102d350986e5bc51799db56ac90f60319c

Author: Wumpkins <dhc2129@columbia.edu>

Date: Sun May 1 19:12:34 2016 -0400

shaping up basic ptx binop

```
commit 1af486df0c750e2b5cec761b546418388c2fc583
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Sun May 1 19:05:21 2016 -0400
```

more ptx

```
commit 97b6530c0e24cb3ec6fe95fe09259e058faa8299
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Sun May 1 18:40:47 2016 -0400
```

started implementing some basic stuff for sast

```
commit 19f2ea6f34df3fca3e33de5d513be3f7ad9d8856
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Sun May 1 18:20:45 2016 -0400
```

nvm it should be data type

```
commit 5af985d7ff63022f09171d7f8a340af1354ba1dd
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Sun May 1 18:19:30 2016 -0400
```

replaced data type with binary type

```
commit 3f8c49b49e25f9cd1b30af4dd03d45ebea665c6d
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Sun May 1 18:17:47 2016 -0400
```

more small changes

```
commit e12878234744ede7a3141ecc145bc094540fe78c
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Sun May 1 18:15:03 2016 -0400
```

working on codegen and sast for ptx

```
commit c36cd27699907ed1fd6628b32f8d5bd42733d5e2
Merge: aebb35a a1fe5c9
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Sun May 1 17:51:06 2016 -0400
```

```
Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie
```

```
commit a1fe5c94451414b28f1b912cbab840accded5211
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Sun May 1 00:32:52 2016 -0400
```

README update

```
commit 88f35472d54b4bc79dea7bb860e7911477d0eaf7
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Sun May 1 00:22:52 2016 -0400
```

c code generation complete

```
commit aebb35a0759a9fa3b57ba71ca2433982e7032c92
Merge: b2df06d 8cf6a1a
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Fri Apr 29 14:01:16 2016 -0400
```

```
Merge branch 'kellie' of https://github.com/Wumpkins/vlc into kellie
```

```
commit b2df06df884eac2f23d3d7c5de50a59ef3160b94
Merge: af11c09 955c649
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Fri Apr 29 14:01:12 2016 -0400
```

nothing

```
commit 8cf6a1a53abc8013e78d94b7e025342d38ad101c
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Fri Apr 29 14:00:41 2016 -0400
```

added new datatypes

```
commit b4de04cfb2355275af64514d47e0cef284948992
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Fri Apr 29 13:11:17 2016 -0400
```

made bitshift binop instead of unop

```
commit af11c097a9d418c7eddc4acafaef2bff91b7339d
```

Author: Wumpkins <dhc2129@columbia.edu>

Date: Fri Apr 29 12:39:30 2016 -0400

nothing

commit 955c6490db401f842f7f05498fd13cb831c7692

Merge: 036a0e2 5a03087

Author: Kellie Ren Lu <krl12130@columbia.edu>

Date: Thu Apr 28 17:46:42 2016 -0400

merge with master

commit 036a0e25855cdcd6918aad62fca6c9a58d176145

Author: Kellie Ren Lu <krl12130@columbia.edu>

Date: Thu Apr 28 15:58:16 2016 -0400

added all conditionals and blocks to ast, parser, utils, and scanner

commit 26da8e008da81dd604b2ca1abe37031e3add65f2

Author: Kellie Ren Lu <krl12130@columbia.edu>

Date: Thu Apr 28 09:59:41 2016 -0400

fixed shift reduce conflict in parser

commit 5f1d744c18f7a74ad104335f7ed88b90e15e5bda

Author: Kellie Ren Lu <krl12130@columbia.edu>

Date: Thu Apr 28 09:45:42 2016 -0400

added other parts of scanner, parser but need to resolve one shift redu

commit 789206dd394debfc2a9aeea69102f862ebdb7994

Author: Kellie Ren Lu <krl12130@columbia.edu>

Date: Tue Apr 26 01:44:00 2016 -0400

added basic checking functions in semant.ml, cleaned up some compiling v

commit 4e65179d2cde7da4c2ebd2bd99e6cff8ea732e43

Author: Kellie Ren Lu <krl12130@columbia.edu>

Date: Mon Apr 25 12:53:16 2016 -0400

make works, need to add more to semant.ml and alter ptx sast

```
commit 703a3bb797f51a1559f4d28191570184e3cdcb60
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Sat Apr 23 18:10:42 2016 -0400
```

faulty workflow

```
commit 5a030874928cca5674c932a6d52d57adced73ca0
Merge: 643f188 6afcfa1
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Sat Apr 23 03:22:44 2016 -0400
```

fixed test case to work

```
commit 6afcfa10c00d28214320aa57b2e736ca70c2526d
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Sat Apr 23 03:11:18 2016 -0400
```

test script for cuda

```
commit a201454f5fe25ea90a224c7207bb5e03039053f0
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Thu Apr 21 15:01:32 2016 -0400
```

before revision

```
commit 032139732c415b3d54eff7d80b3ec970c6850d2e
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Wed Apr 20 21:50:24 2016 -0400
```

nothing

```
commit 67b90b0ca47f477ad4cbeac83c493cad780761ec
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Tue Apr 12 14:23:32 2016 -0400
```

added cuda file with ptx that we want to generate

```
commit 4423bd79b49916df30b28b7b2b6795509a7d53b0
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Mon Apr 11 15:04:22 2016 -0400
```

made higher order function more concise and portable

commit c113d2b3e7b7ef8d277d6bea7b2ac3466afee98f

Author: Kellie Ren Lu <krl12130@columbia.edu>

Date: Sun Apr 10 21:28:38 2016 -0400

readded ptx that was accidentally deleted

commit 345932517f38f9c29c0500c11b80a6c1eb2fe660

Author: Kellie Ren Lu <krl12130@columbia.edu>

Date: Sun Apr 10 19:54:57 2016 -0400

compiler recognizes map and reduce but doesn't yet codegen ptx correctly

commit 3903f60107bc48206d36c9c22e7dd027ead2c07e

Merge: 643f188 5e5abda

Author: Kellie Ren Lu <krl12130@columbia.edu>

Date: Sat Apr 9 15:01:19 2016 -0400

merge with diana

commit 643f1885f42180fab3fb4cf00d07b79b57c04fb

Author: Kellie Ren Lu <krl12130@columbia.edu>

Date: Sat Apr 9 14:55:08 2016 -0400

fixed scanner white space problem by fixing processor.ml

commit 5e5abdaa72837af0e270c7f4a082f0f20b99c22d

Author: dianarvp <dianarvp@gmail.com>

Date: Sat Apr 9 00:08:18 2016 -0400

Added PTX skeleton code

commit 3a479eefd05bca847e9fa590631ac5a456383d0e

Author: Wumpkins <dhc2129@columbia.edu>

Date: Fri Apr 8 21:07:22 2016 -0400

testing

commit 310b7956f53133f1ab8fad27f7e42c24d7442c8d

```
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Fri Apr  8 21:03:29 2016 -0400
```

forgot some arrows

```
commit b13c985e43e25b3cf6falee9efcf8533120a86d0
Merge: 62578a5 71765fa
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Fri Apr  8 20:59:32 2016 -0400
```

merge with kellie

```
commit 62578a5f14bfa71bfb7bcd6e4894ee43b29f93c8
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Fri Apr  8 20:43:07 2016 -0400
```

fixed whitespace parsing for empty lines

```
commit bd0eba2425706d3deaecbaae1e231b2527db4e8d
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Fri Apr  8 20:24:54 2016 -0400
```

added function declaration test

```
commit dbbdab59c42150575967dd7fc15f4e986784151d
Merge: 0879e6d 71765fa
Author: dianarvp <dianarvp@gmail.com>
Date:   Fri Apr  8 20:20:32 2016 -0400
```

Merge branch 'kellie' of https://github.com/Wumpkins/vlc into diana

```
commit c06715b35d71dc6ccce5c228be38434715977a46
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Fri Apr  8 20:18:04 2016 -0400
```

added arithmetic test case

```
commit 0879e6dcf8f0da3b19c7262b478cf0d7ec738c51
Author: dianarvp <dianarvp@gmail.com>
Date:   Fri Apr  8 20:15:34 2016 -0400
```

Preparing for merge

```
commit 71765fae45a2f11bf5d49968879fb33d4cc1c54b
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Fri Apr  8 19:31:35 2016 -0400
```

added defg in ast, parser, codegen but defg doesn't yet codegen ptx

```
commit 8c91a2a68184dbe3f9963a40068b5c5d9229c717
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Thu Apr  7 14:19:17 2016 -0400
```

added multidimensional arrays

```
commit 2af0b2a9365799b30b35333e46f6bb97197ee3aa
Merge: 9b09150 472cbe9
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Thu Apr  7 04:11:51 2016 -0400
```

Merge branch 'kellie' of ssh://github.com/Wumpkins/vlc into kellie

```
commit 9b09150dc9f983e50a23a2d70a8bd305326020d1
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Thu Apr  7 04:03:24 2016 -0400
```

added one dimensional array, generates correct code

```
commit 77b39025ffe4cf5de9952cd37e75750a2d2e927d
Merge: 336da45 472cbe9
Author: dianarvp <dianarvp@gmail.com>
Date:   Thu Apr  7 04:07:09 2016 -0400
```

Merge branch 'kellie' of https://github.com/Wumpkins/vlc into diana

```
commit 336da4554611cf6dc2cd419dd61c07313ab18d2f
Author: dianarvp <dianarvp@gmail.com>
Date:   Thu Apr  7 04:06:51 2016 -0400
```

.

```
commit ea6e890d93a591efe0195d0de138562a12dacbe6
```

```
Merge: 932aa49 139b9cd
Author: dianarvp <dianarvp@gmail.com>
Date: Thu Apr 7 04:05:23 2016 -0400
```

```
    Merge branch 'kellie' of https://github.com/Wumpkins/vlc into diana

commit 472cbe9116f524cefadc5fe319ca65c1b7dd3d5c
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Thu Apr 7 04:03:24 2016 -0400
```

```
    added {} in hard coded ptx
```

```
commit 932aa49f01dbb0a71da73ed8d190018720e32bf4
Author: dianarvp <dianarvp@gmail.com>
Date: Thu Apr 7 04:02:35 2016 -0400
```

```
Whoops
```

```
commit 139b9cdf01bf2a8058d63b565c4b134592ed9ddf
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Thu Apr 7 03:51:05 2016 -0400
```

```
    array now generates correctly
```

```
commit fd2ed93d9cf9c3eedad41c0445f2fe196efc6498e
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Thu Apr 7 03:12:33 2016 -0400
```

```
before alterig array, currently generates code but generated code for a
```

```
commit fbeeff56638f8132dee80a8841d5844b9624021d
Merge: 88a5b99 27c33ed
Author: dianarvp <dianarvp@gmail.com>
Date: Thu Apr 7 02:54:09 2016 -0400
```

```
    Merge branch 'kellie' of https://github.com/Wumpkins/vlc into diana

commit 88a5b9973b7bc72713395627b9d58e7f207112bc
Author: dianarvp <dianarvp@gmail.com>
Date: Thu Apr 7 02:49:21 2016 -0400
```

Test added

```
commit 27c33edfcdb2dc17671d08a4ccf0e97e7e495f02
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Thu Apr  7 02:48:13 2016 -0400
```

added array and compiles

```
commit f64645bb32f18cd2186949b9b06ad5550a670895
Author: dianarvp <dianarvp@gmail.com>
Date:   Thu Apr  7 02:47:47 2016 -0400
```

Array types implemented

```
commit 4e8c93ab61cb4c5c51eaa5dd262566886756a2b6
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Thu Apr  7 02:26:00 2016 -0400
```

added binary operators

```
commit 4c8c865392d64494f7acf86a278ef7f23123dbd2
Merge: 35c6db2 662595c
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Thu Apr  7 00:34:09 2016 -0400
```

Merge branch 'master' of ssh://github.com/Wumpkins/vlc into kellie

```
commit a069975615894fcd4f880a2f6d7a4b1f8feb0a00
Merge: 71971f8 662595c
Author: dianarvp <dianarvp@gmail.com>
Date:   Thu Apr  7 00:33:10 2016 -0400
```

Merge branch 'master' of https://github.com/Wumpkins/vlc into diana

```
commit 662595c5b65959e1ea9fec3a48193fa52dbf6ccf
Author: Wumpkins <dhc2129@columbia.edu>
Date:   Tue Apr  5 07:02:07 2016 -0400
```

basic testing, copied from micro c

```
commit 8960ed497332add8686bde0190ccc2d7639fce20
```

Author: Wumpkins <dhc2129@columbia.edu>
Date: Tue Apr 5 03:18:30 2016 -0400

output

commit c9926c6eb5cb331a742d3aeadb7ebf69b5d7bb06
Author: Wumpkins <dhc2129@columbia.edu>
Date: Tue Apr 5 03:17:05 2016 -0400

more clean up

commit 71971f831719bdd5a70a993faef458e2586bc92
Merge: 0998225 35c6db2
Author: dianarvp <dianarvp@gmail.com>
Date: Tue Apr 5 03:16:32 2016 -0400

Merge branch 'kellie' of https://github.com/Wumpkins/vlc into diana

commit 7c99ada3820d07b92c5ba5e033da3e7be5478653
Author: Wumpkins <dhc2129@columbia.edu>
Date: Tue Apr 5 03:15:46 2016 -0400

git ignore and file cleanup

commit 0998225af9ba4bc909df0a209d3853dd0d53bf77
Author: dianarvp <dianarvp@gmail.com>
Date: Tue Apr 5 03:14:32 2016 -0400

Stuff

commit 35c6db205224cb9dcf5c66b58e3093c88edeb319
Merge: 0f36e0c 21d85ff
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Tue Apr 5 03:13:23 2016 -0400

fixed merges

commit 21d85ff917ccecd63520f816fac1009ebf71fd98
Merge: 24cb104 d58606a
Author: dianarvp <dianarvp@gmail.com>
Date: Tue Apr 5 02:55:05 2016 -0400

Merged with Kellie

```
commit 24cb104f580c2c61925d2f2ff748fde30ccbb29f
Author: dianarvp <dianarvp@gmail.com>
Date:   Tue Apr  5 02:48:17 2016 -0400
```

Added scope stack and codegen type inference /incomplete

```
commit 0f36e0ceeda78412eb81129255cedb8250bea7d1
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Tue Apr  5 02:38:56 2016 -0400
```

added array and identifier in ast.ml

```
commit cd69b811676f904ba8e7bc0ef16b32785c423ab4
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Tue Apr  5 01:54:30 2016 -0400
```

added new identifier

```
commit 064cd6549c63b8ac6a1b6fcraf43d36e45e218aa3
Merge: 3db77e7 47ebdf0
Author: dianarvp <dianarvp@gmail.com>
Date:   Tue Apr  5 00:15:02 2016 -0400
```

Working helloworld

```
commit dd013dfdd813776118a9fa482246b4ed393a5e4d
Merge: d77fc63 d58606a
Author: Kellie Lu <krl2130@columbia.edu>
Date:   Tue Apr  5 00:13:46 2016 -0400
```

Merge pull request #2 from Wumpkins/diana

working hello world to cuda c

```
commit d58606aeeffe9dea595be512895bd3dce314ab52a
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Mon Apr  4 23:52:54 2016 -0400
```

```
hello world compiles! with diana
```

```
commit 035e3b7934d0c4908cad0a50ab211c672748c8a2
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Mon Apr  4 22:42:20 2016 -0400
```

```
compiles
```

```
commit 47ebdf0c55850fb952d62829174fb4bf990e1860
Merge: 346a82d 360fa9e
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Mon Apr  4 12:18:29 2016 -0400
```

```
Merge branch 'kellie' of ssh://github.com/Wumpkins/vlc into kellie
```

```
commit 346a82d501a419ff5986d018c87626014b0d7702
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Mon Apr  4 12:15:19 2016 -0400
```

```
small syntax fixes
```

```
commit 360fa9e2acf7ea50165ecb0c4009f98cbd490494
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Mon Apr  4 12:15:19 2016 -0400
```

```
small syntax fixes
```

```
commit 3db77e755c1430894c5130bed80241c7a24804bd
Author: dianarvp <dianarvp@gmail.com>
Date:   Mon Apr  4 05:23:52 2016 -0400
```

```
Codegen and environment now compile
```

```
commit a6211f99053ae983b02a3116449e7963aab5a7d
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date:   Sun Apr  3 08:42:16 2016 -0400
```

```
working parser and scanner
```

```
commit 9b064df0596a2371dbc5f9d39b84c3f284472d6f
Author: Kellie Ren Lu <krl12130@columbia.edu>
```

Date: Sun Apr 3 06:01:24 2016 -0400

diana

commit 54a73568db4b81c96e80778da1cbb0cf43eea3e9

Author: Kellie Ren Lu <krl12130@columbia.edu>

Date: Sun Apr 3 03:43:40 2016 -0400

before altering new parser

commit b8d0b5562a88a9ac9a44f37f2883ef7d4633555d

Author: Kellie Ren Lu <krl12130@columbia.edu>

Date: Sat Apr 2 19:17:55 2016 -0400

new parser implementation

commit d77fc630cd0127425410e939b6da8a4c7d035611

Author: Kellie Ren Lu <krl12130@columbia.edu>

Date: Wed Mar 30 09:35:53 2016 -0400

updated scanner and parser files after group work over weekend

commit ad414eaf844fcf1bcac0f32519cfecbc41c41e35

Author: Kellie Ren Lu <krl12130@columbia.edu>

Date: Sun Mar 27 14:15:00 2016 -0400

updated sample programs

commit 6c3cab724a23cd0c22607a98beb75b5f1c2c9456

Author: Kellie Ren Lu <krl12130@columbia.edu>

Date: Sun Mar 27 14:06:18 2016 -0400

ast.ml scanner.mll and parser.mly rough drafts for hello world

commit d6b64e5afee195434e2acfafad9eab89d1fea979

Author: Kellie Ren Lu <krl12130@columbia.edu>

Date: Wed Mar 23 18:34:09 2016 -0400

skeleton for compiler

commit 50cf98f618e977bb9f566ad055246bd0619955d8

Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Tue Mar 8 17:27:50 2016 -0500

changed map

commit 54649c661142c4f95ed740bf5ed35cfef0ee3018
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Mon Mar 7 20:15:45 2016 -0500

added paren

commit ce1c0ff77f2a543d7fcba283a3df1ba8c959f4ba
Merge: ff775f5 b835158
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Mon Mar 7 18:40:17 2016 -0500

fixed conflict

commit ff775f52111abeb1e376b633d2fa61b39d636c4fd
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Mon Mar 7 18:36:19 2016 -0500

revisions

commit b8351582e834b15fa8a76ab8415c24a60dddf3f1
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Mon Mar 7 18:36:19 2016 -0500

revisions

commit 5de939d8c9c8c4d7bfb1edaa851b606d538afd
Merge: b5bff23 9170d73
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Mon Mar 7 18:23:49 2016 -0500

Merge branch 'master' of ssh://github.com/Wumpkins/vlc into kellie

commit 9170d73815310cce667b0e8d3eb28c0b5c7bb2a0
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date: Mon Mar 7 18:08:04 2016 -0500

added page breaks for printing

```
commit 56782951f90374d13d7957c111d6c61eff97bd45
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Mon Mar  7 17:09:11 2016 -0500
```

final table fix

```
commit ceb5f251c83f96b940c955532249003df4102ff5
Merge: 5092b9e c1741f4
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Mon Mar  7 17:07:53 2016 -0500
```

Merge branch 'master' of ssh://github.com/Wumpkins/vlc

```
commit 5092b9e9e3cac06fd4c4776ff70e7263e57f3e11
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Mon Mar  7 17:05:52 2016 -0500
```

fixed table rendering

```
commit c1741f48a95ac1d3daa670c617deb3dad6916ae8
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Mon Mar  7 17:05:52 2016 -0500
```

fixed table rendering

```
commit ec54b2380b3f30ee3788097b163283491ed54806
Merge: e1e1123 159edde
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Sun Mar  6 21:47:07 2016 -0500
```

fixed merge conflicts

```
commit e1e1123741a8f864d3ef30d381a90ba407574112
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Sun Mar  6 21:44:42 2016 -0500
```

fixed table of contents

```
commit 159edde29ff2f7f03cea54525c6a55b8b88752d8
```

Author: David <dhc2129@columbia.edu>
Date: Sun Mar 6 21:43:23 2016 -0500

test br

commit 390cc09998222cc6530a9a264845f49212e43f65
Merge: 1b22e9f f3a57c8
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date: Sun Mar 6 21:40:30 2016 -0500

solved merge conflicts

commit 1b22e9f997589381534e8d6553a240ac4bf8b875
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date: Sun Mar 6 21:36:34 2016 -0500

lrm after review

commit f3a57c86e6721a927f5ef3281308e900452b8431
Merge: 81940b2 13f725c
Author: David <dhc2129@columbia.edu>
Date: Sun Mar 6 19:47:06 2016 -0500

Merge branch 'kellie' of https://github.com/Wumpkins/vlc

commit b5bff236f3f9a0e05c5bb0070bd8290bb250f7ff
Merge: 13f725c 81940b2
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date: Sun Mar 6 19:44:14 2016 -0500

Merge branch 'master' of ssh://github.com/Wumpkins/vlc into kellie

commit 13f725c31c7cd57c0efafe2ec60daa604c658616
Author: Kellie Ren Lu <krl12130@columbia.edu>
Date: Sun Mar 6 19:43:22 2016 -0500

finished lrm

commit 81940b2d557d9477806b5deb29dca311802fe39c
Merge: da86a4f b51cfca
Author: Kellie Lu <krl12130@columbia.edu>

Date: Thu Mar 3 20:54:16 2016 -0500

Merge pull request #1 from Wumpkins/kellie

lrm

commit b51cfcad255a787bc95c261f8e1f0028e2076ef0

Author: Kellie Ren Lu <krl2130@columbia.edu>

Date: Thu Mar 3 19:35:03 2016 -0500

lrm

commit da86a4fbccbf54fb3263dfbc29d19250fd1ff88

Author: David Chen <dhc2129@columbia.edu>

Date: Sat Feb 20 19:19:29 2016 -0500

Initial commit

6. LESSONS LEARNED

C'est la vie.

Just kidding.

Kellie

I learned that things that seem simple in concept take four times as long to implement.

David

I've learned a lot about the compiler process through this language, especially because of the two step sast that we implemented for vlc. I've also gone from practically no experience with GPU coding to being able to understand and semantically analyze PTX assembly, which is nice.

Diana

The group roles that Professor Edwards specifies at the beginning of the project are not arbitrary. As hard as it is to believe, there is a genuine need for someone whose primary or full responsibility it is to manage the team and keep highly organized documentation. There is also a genuine need for someone whose sole responsibility is to write tests. This is not to say that everyone should just ignore the others' parts, but rather to reiterate that the best compilers come from one mind, and that it's simply not reasonable to expect that you and the three random people you meet in class will become a hivemind for the rest of the semester. Of course, some groups have become successful through doing just that; for everyone else, go the true and tried method.

7. CODE LISTING

Compiler

scanner.mll

```
1 214 lines (175 sloc) 7.61 KB
2 open Ast
3 (* Contains sast type definitions for conversions during
   ↳ semantic analysis *)
4 (* -----PTX types
   ↳ -----*)
5 type ptx_data_movement =
6   | Ptx_Move | Ptx_Load | Ptx_Store
7
8 type ptx_binary_operator =
9   | Ptx_Add | Ptx_Subtract | Ptx_Multiply | Ptx_Divide |
10  ↳ Ptx_Modulo
11
12 type ptx_data_type =
13   | U16 | U32 | U64 | S16 | S32 | S64
14
15 (* should use this as our information about global/param
   ↳ etc.*)
16 type ptx_variable_type =
17   | Ptx_Primitive of ptx_data_type
18   | Ptx_Array of ptx_variable_type * int          (* 'int'
   ↳ refers to the length of the array *)
19   | Ptx_Pointer of ptx_variable_type * int         (* 'int'
   ↳ refers to size of memory pointed by the pointer *)
20
21 type ptx_register_decl =
22   | Register_Declaration of ptx_data_type * string * int
23   ↳ (* type, name, number of registers *)
24
25 type ptx_register =
26   | Register of string * int                      (* register
   ↳ name, register number *)
27   (* Not sure what this is | Typed_Register of ptx_data_type
   ↳ * string * int      (* type, register name, register
   ↳ number *) *)
28   (* Implement later | Special_Register of string      (*
   ↳ register name *) *)
```

```

28 type ptx_parameter =
29   | Parameter_register of ptx_register
30   | Parameter_constant of int
31   | Parameter_variable of Ast.identifier
32
33
34 type ptx_expression =
35   | Ptx_reg_declaration of ptx_register_decl
36   | Ptx_movement of ptx_data_movement * ptx_data_type *
37     ↳ ptx_variable_type * ptx_parameter * ptx_parameter
38   | Ptx_Binop of ptx_binary_operator * ptx_data_type *
39     ↳ ptx_parameter * ptx_parameter * ptx_parameter
40   | Ptx_Return
41   (*      | Ptx_Array_Literal of ptx_expression list
42   | Ptx_Function_Call of Ast.identifier * ptx_expression
43     ↳ list
44   | Ptx_Identifier_Expression of Ast.identifier
45   *)
46
47
48 type ptx_subroutine = {
49   routine_name : Ast.identifier;
50   routine_expressions : ptx_expression list;
51 }
52
53
54 type ptx_statement =
55   (*      | Ptx_Initialization of ptx_vdecl * ptx_expression
56     ↳ *)
57   (*      | Ptx_Assignment of Ast.identifier * ptx_expression
58     ↳ *)
59     | Ptx_expression of ptx_expression
60     | Ptx_subroutine of ptx_subroutine
61
62
63 type ptx_function_type =
64   | Global
65   | Device
66
67
68 type ptx_constant =
69   {
70     ptx_constant_name : Ast.identifier;
71     ptx_constant_variable_type : ptx_variable_type;
72   }
73
74
75 type ptx_variable_space =
76   | Global
77   | Local
78   | Shared
79
80 type ptx_vdecl =
81   | Ptx_Vdecl of ptx_data_type * ptx_variable_space (*
82     ↳ need something about global/ptrs here*)
83     ↳ ptx_variable_type * Ast.identifier
84
85
86
87
88
89
90
91
92
93

```

```

74 (* ptx_fdecl is the entire file
75   it seems it really only needs to be composed of a few
76   → parts - a name, a variable declaration list
77   and a statement list
78   register_decl list should go inside body generated from
79   → semantic analyzer
80 *)
81 type ptx_fdecl = {
82   (* Global or Device *)
83   ptx_fdecl_type : ptx_function_type; (*
84   → probably not needed *)
85
86   (* Name of the function *)
87   ptx_fdecl_name : Ast.identifier;
88
89   (* Expected parameters of the function *)
90   ptx_fdecl_params : ptx_vdecl list;
91
92   (* List of constants that function needs to know - aka
93   → variables that aren't in scope of function when it goes
94   → through semantic analyzer
95   If this constant list doesn't match the constant list
96   → of the higher order function, throw error in semant.ml
97   → *)
98   ptx_consts : ptx_constant list;
99
100  (* Declares the virtual registers that are needed for the
101   → function *)
102  register_decls : ptx_register_decl list;
103  (* Statements within the function *)
104  ptx_fdecl_body : ptx_statement list;
105 }
106
107
108
109
110
111
112
113 (* -----C types
114   → -----*)
115
116 (*-----*
117   → Unnecessary?????????-----*
118 *)
119 type c_binary_operator =
120   | Add | Subtract | Multiply | Divide | Modulo
121   (*   | Plus_Equal | Subtract_Equal | Multiply_Equal |
122   → Divide_Equal *)
123   (*   | Exp | Dot | Matrix_Multiplication *)
124   | And | Or | Xor
125   | Equal | Not_Equal | Greater_Than | Less_Than |
126   → Greater_Than_Equal | Less_Than_Equal
127   | Bitshift_Right | Bitshift_Left
128
129 type c_unary_operator =

```

```

114     | Not | Negate
115     | Plus_Plus | Minus_Minus
116
117 type c_data_type =
118     | String
119     | Byte
120     | Unsigned_Byte
121     | Integer
122     | Unsigned_Integer
123     | Long
124     | Unsigned_Long
125     | Float
126     | Double
127     | Boolean
128     | Void
129
130 type c_variable_type =
131     | Primitive of c_data_type
132     | Array of c_variable_type * int
133 (*   / Struct of variable_type list * expression list * int
134   ↵   *)
135
136 type c_vdecl =
137     Variable_Declaration of c_variable_type *
138     ↵ Ast.identifier
139
138 (*
139   ↵ -----Necessary-----
140   ↵ *)
141
140 type c_kernel_variable_info = {
141     variable_type      : c_variable_type;
142     host_name         : Ast.identifier;
143     kernel_name       : Ast.identifier;
144 }
145
146 type c_higher_order_function_call = {
147     (* Map or reduce *)
148     higher_order_function_type      : Ast.identifier;
149     (* Name of kernel function that is called from host
150      (would be kernel function corresponding to map/reduce)
151      *)
152     applied_kernel_function        : Ast.identifier;
153     (* List of constants passed into map and reduce *)
154     constants                     : c_kernel_variable_info list;
155     (* Size of input and return arrays *)
156     array_length                  : int;
157     (* Input array information
158      --If an array has no name (just simply passed in as
159      something like {1,2,3}) then it is given a temporary
160      generated name *)
161     input_arrays_info             : c_kernel_variable_info
162     list; (* type, host name, kernel name *)

```

```

158     (* Return array information *)
159     return_array_info           :
160     ↳ c_kernel_variable_info; (* type, host name, kernel
161     ↳ name*)
160 }
161
162 (* Type for calling defg functions directly from host *)
163 type c_kernel_function_call = {
164     (* Name of the function that is called from the host *)
165     kernel_function           : Ast.identifier;
166     (* Input array information
167     --If an array has no name (just simply passed in as
168     ↳ something like {1,2,3}) then it is given a temporary
168     ↳ generated name *)
168     input_args_info           : c_kernel_variable_info
169     ↳ list; (* type, host name, kernel name *)
169     (* Return array information *)
170     return_arg_info           :
170     ↳ c_kernel_variable_info; (* type, host name, kernel
170     ↳ name*)
171 }
172
173 type c_expression =
174 | Function_Call of Ast.identifier * c_expression list
175 | Higher_Order_Function_Call of
176     ↳ c_higher_order_function_call
177 | Kernel_Function_Call of c_kernel_function_call
178 | String_Literal of string
179 | Integer_Literal of int
180 | Boolean_Literal of bool
181 | Floating_Point_Literal of float
182 | Array_Literal of c_expression list
183 | Identifier_Literal of Ast.identifier
184 | Cast of c_variable_type * c_expression
184 | Binop of c_expression * c_binary_operator *
185     ↳ C_expression
186 | Unop of c_expression * c_unary_operator
186 | Array_Accessor of c_expression * c_expression list (*
186     ↳ Array, indexes *)
187 | Ternary of c_expression * c_expression * c_expression
187     ↳ (* expression if true, condition, expression if false
187     ↳ *)
188
189 type c_variable_statement =
190 | Declaration of c_vdecl
191 | Initialization of c_vdecl * c_expression
192 | Assignment of Ast.identifier * c_expression
193
194 type c_statement =
195 | Variable_Statement of c_variable_statement
196 | Expression of c_expression
197 | Block of c_statement list (* Used for if, else, for,
197     ↳ while blocks *)

```

```
198      | If of c_expression * c_statement * c_statement (*
→   expression-condition, statement-if block,
→   statement-optional else block *)
199      | While of c_expression * c_statement
200      | For of c_statement * c_expression * c_statement *
→   c_statement
201      | Return of c_expression
202      | Return_Void
203      | Continue
204      | Break
205
206 type c_fdecl = {
207     c_fdecl_return_type      : c_variable_type;
208     c_fdecl_name            : Ast.identifier;
209     c_fdecl_params          : c_vdecl list;
210     c_fdecl_body            : c_statement list;
211 }
212
213 (* Overall Program *)
214 type program = c_variable_statement list * ptx_fdecl list *
→   c_fdecl list
```

parser.mly

```
1  %{ open Ast;; (*open Exceptions;;*)  
2  
3  
4      (* Converts keywords to appropriate datatype *)  
5  let string_to_data_type = function  
6    | "string" -> String  
7    | "bool" -> Boolean  
8    | "void" -> Void  
9    | "ubyte" -> Unsigned_Byte  
10   | "byte" -> Byte  
11   | "uint" -> Unsigned_Integer  
12   | "int" -> Integer  
13   | "ulong" -> Unsigned_Long  
14   | "long" -> Long  
15   | "float" -> Float  
16   | "double" -> Double  
17   | dtype -> raise (Exceptions.Invalid_data_type dtype)  
18  
19  %}  
20  
21  %token LPAREN RPAREN LBRACKET RBRACKET LCURLY RCURLY INDENT  
22      ↪ DEIDENT COLON TERMINATOR EOF COMMA  
22  %token DEF DEFG RETURN CONSTS TILDA  
23  %token <int> DEIDENT_EOF, DEIDENT_COUNT  
24  
25  %token ADD SUBTRACT MULTIPLY DIVIDE MODULO  
26  %token PLUS_PLUS_MINUS_MINUS  
27  %token BITSHIFT_RIGHT BITSHIFT_LEFT  
28  %token AND OR NOT XOR  
29  %token EQUAL NOT_EQUAL GREATER_THAN GREATER_THAN_EQUAL  
29      ↪ LESS_THAN LESS_THAN_EQUAL  
30  %token IF ELSE WHILE FOR  
31  %token CONTINUE BREAK  
32  
33  %token ASSIGNMENT  
34  
35  %token <int> INTEGER_LITERAL  
36  %token <string> STRING_LITERAL  
37  %token <float> FLOATING_POINT_LITERAL  
38  %token <bool> BOOLEAN_LITERAL  
39  
40  %token <string> IDENTIFIER  
41  %token <string> DATATYPE  
42  
43  %nonassoc ELSE NOELSE  
44  %right ASSIGNMENT  
45  %left IF  
46  %left LBRACKET RBRACKET  
47  %left EQUAL NOT_EQUAL GREATER_THAN GREATER_THAN_EQUAL  
47      ↪ LESS_THAN LESS_THAN_EQUAL
```

```

48 %left AND NOT OR XOR
49 %left BITSHIFT_RIGHT BITSHIFT_LEFT
50 %left ADD SUBTRACT PLUS_PLUS MINUS_MINUS
51 %left MULTIPLY DIVIDE MODULO
52 %right NEGATE
53
54 %start program
55 %type <Ast.program> program
56
57 %%%
58
59 program:
60     | /* nothing */
61     | [ ] /* variable statements, function declarations */
62     | program variable_statement TERMINATOR
63     | { List.rev ($2 :: List.rev (fst $1)), snd $1 }
64     | program fdecl
65     | { fst $1, List.rev($2 :: List.rev(snd $1)) }
66
67 identifier:
68     | IDENTIFIER
69     | { Identifier($1) }
70
71 /* Kernel and host function declarations */
72 fdecl:
73     | variable_type DEF identifier LPAREN parameter_list
74     | RPAREN COLON indent_block
75
76     | {
77         | is_kernel_function = false;
78         | return_type = $1;
79         | name = $3;
80         | params = $5;
81         | body = $8;
82     }
83     | {
84         | variable_type DEFG identifier LPAREN parameter_list
85         | RPAREN COLON indent_block
86
87         | {
88             | is_kernel_function = true;
89             | return_type = $1;
90             | name = $3;
91             | params = $5;

```

```

83     ↵   body = $8;
84
85     ↵   }
86
87 /* Constant parameters for higher order function calls */
88 constant:
89     | identifier ASSIGNMENT expression
90     ↵   {Constant($1,$3)}
91
92 constant_list:
93     | /* nothing */
94     ↵   { [] }
95     | nonempty_constant_list
96     ↵   { $1 }
97
98 nonempty_constant_list:
99     | constant COMMA nonempty_constant_list
100    ↵   {$1 :: $3}
101    | constant
102    ↵   { [$1] }

103 /* Higher order function calls */
104 higher_order_function_call:
105     | TILDA identifier LPAREN identifier COMMA CONSTS
106     ↵   LPAREN constant_list RPAREN COMMA
107     ↵   nonempty_array_expression_list RPAREN
108
109     ↵   {
110
111     ↵   higher_order_function_type = $2;
112
113     ↵   kernel_function_name = $4;
114
115     ↵   constants = $8;
116
117     ↵   input_arrays = $11;
118
119     ↵   }
120
121     | TILDA identifier LPAREN identifier COMMA
122     ↵   nonempty_array_expression_list RPAREN
123
124     ↵   {
125
126     ↵   higher_order_function_type = $2;
127
128     ↵   kernel_function_name = $4;
129
130     ↵   constants = [];
131
132     ↵   input_arrays = $6;
133
134     ↵   }

```

```

115
116
117
118 /* Parameters for normal host functions and kernel
   ↳ functions */
119 vdecl:
   | variable_type identifier
   ↳ { Variable_Declaration($1,$2) }
120
121 nonempty_parameter_list:
   | vdecl
   ↳ { [$1] }
122   | nonempty_parameter_list COMMA vdecl
   ↳ {$3 :: $1}
123
124 parameter_list:
   | /* nothing */
   ↳ { [] }
125   | nonempty_parameter_list
   ↳ { $1 }
126
127
128
129
130
131
132 /* Statements */
133 variable_statement:
134   | vdecl TERMINATOR
   ↳ { Declaration($1) }
135   | assignment_expression ASSIGNMENT expression
   ↳ TERMINATOR           { Assignment( $1, $3 ) }
136   | vdecl ASSIGNMENT expression TERMINATOR
   ↳ { Initialization ($1, $3) }
137
138 for_statement:
139   | assignment_expression ASSIGNMENT expression
   ↳ { Variable_Statement(Assignment($1,$3)) }
140   | vdecl ASSIGNMENT expression
   ↳ { Variable_Statement(Initialization($1,$3)) }
141
142 statement:
143   | expression TERMINATOR
   ↳ { Expression($1) }
144   | RETURN expression TERMINATOR
   ↳ { Return($2) }
145   | RETURN TERMINATOR
   ↳ { Return_Void }
146   | CONTINUE TERMINATOR
   ↳ { Continue }
147   | BREAK TERMINATOR
   ↳ { Break }
148   | IF LPAREN expression RPAREN COLON indent_block %prec
   ↳ { If($3,
NOELSE
   ↳ { Block($6), Block([]) ) }
```

```

149      | IF LPAREN expression RPAREN COLON indent_block ELSE
150      | COLON indent_block                                { If($3,
151      | Block($6), Block($9)) }
150      | FOR LPAREN for_statement COMMA expression COMMA
151      | for_statement RPAREN COLON indent_block    {
151      | For($3,$5,$7,Block($10)) }
151      | WHILE LPAREN expression RPAREN COLON indent_block
152      | { While($3, Block($6)) }
152      | variable_statement
152      | { Variable_Statement($1) }

153
154 nonempty_statement_list:
155     | statement
155     | { [$1] }
156     | nonempty_statement_list statement
156     | { List.rev($2 :: List.rev($1)) }

157
158 /* Group of statements */
159 indent_block:
160     | /* nothing */
160     | { [] }
161     | INDENT nonempty_statement_list DEDENT
161     | { $2 }

162
163
164
165 /* Expressions */
166 expression:
167     | identifier LPAREN expression_list RPAREN
167     | { Function_Call($1,$3) }
168     | higher_order_function_call
168     | { Higher_Order_Function_Call($1) }

169
170     | LPAREN expression RPAREN
170     | { $2 }

171
172     | STRING_LITERAL
172     | { String_Literal($1) }
173     | INTEGER_LITERAL
173     | { Integer_Literal($1) }
174     | BOOLEAN_LITERAL
174     | { Boolean_Literal($1) }
175     | FLOATING_POINT_LITERAL
175     | { Floating_Point_Literal($1) }
176     | array_literal
176     | { $1 }
177     | identifier
177     | { Identifier_Literal($1) }

178
179     | expression AND expression
179     | { Binop($1, And, $3) }
180     | expression OR expression
180     | { Binop($1, Or, $3) }

```

```

181      | expression XOR expression
182      ↳ { Binop($1, Xor, $3) }
183      | NOT expression
184      ↳ { Unop($2, Not) }

185      | expression EQUAL expression
186      ↳ { Binop($1, Equal, $3) }
187      | expression NOT_EQUAL expression
188      ↳ { Binop($1, Not_Equal, $3) }
189      | expression GREATER_THAN expression
190      ↳ { Binop($1, Greater_Than, $3) }
191      | expression GREATER_THAN_EQUAL expression
192      ↳ { Binop($1, Greater_Than_Equal, $3) }
193      | expression LESS_THAN expression
194      ↳ { Binop($1, Less_Than, $3) }
195      | expression LESS_THAN_EQUAL expression
196      ↳ { Binop($1, Less_Than_Equal, $3) }

197      | SUBTRACT expression
198      ↳ { Unop($2, Negate) }
199      | expression ADD expression
200      ↳ { Binop($1, Add, $3) }
201      | expression PLUS_PLUS
202      ↳ { Unop($1, Plus_Plus) }
203      | expression MINUS_MINUS
204      ↳ { Unop($1, Minus_Minus) }
205      | expression SUBTRACT expression
206      ↳ { Binop($1, Subtract, $3) }
207      | expression MULTIPLY expression
208      ↳ { Binop($1, Multiply, $3) }
209      | expression DIVIDE expression
210      ↳ { Binop($1, Divide, $3) }
211      | expression MODULO expression
212      ↳ { Binop($1, Modulo, $3) }
213      | expression BITSHIFT_RIGHT expression
214      ↳ { Binop($1, Bitshift_Right, $3) }
215      | expression BITSHIFT_LEFT expression
216      ↳ { Binop($1, Bitshift_Left, $3) }
217      | variable_type LPAREN expression Rparen
218      ↳ { Cast($1, $3) }

219      | expression IF LPAREN expression Rparen ELSE
220      ↳ { expression { Ternary($1, $4, $7) } }
221      | array_expression nonempty_array_accessor_list
222      ↳ { Array_Accessor($1, $2) }

223      nonempty_expression_list:
224          | expression COMMA nonempty_expression_list
225          ↳ { $1 :: $3 }
226          | expression
227          ↳ { [$1] }

```

```

211 expression_list:
212     | /* nothing */
213     | { [] }
214     | nonempty_expression_list
215     | { $1 }
216
217 array_accessor:
218     | LBRACKET expression RBRACKET
219     | { $2 }
220
221 nonempty_array_accessor_list:
222     | nonempty_array_accessor_list array_accessor
223     | { $2 :: $1 }
224     | array_accessor
225     | { [$1] }
226
227 array_literal:
228     | LCURLY nonempty_expression_list RCURLY
229     | { Array_Literal($2) }
230
231 array_expression:
232     | identifier
233     | { Identifier_Literal($1) }
234     | array_literal
235     | { $1 }
236
237 nonempty_array_expression_list:
238     | array_expression
239     | { [$1] }
240     | nonempty_array_expression_list COMMA array_expression
241     | { $3 :: $1 }
242
243 /* Expressions that can be assigned on the right side of
244    the assignment statement */
245 assignment_expression:
246     | identifier
247     | { Identifier_Literal($1) }
248     | array_expression nonempty_array_accessor_list
249     | { Array_Accessor($1,$2) }
250
251 /* Variable types and Data types */
252 data_type:
253     | DATATYPE
254     | { string_to_data_type $1 }
255
256 variable_type:
257     | data_type
258     | { Primitive($1) }
259     | data_type array_dimension_list
260     |
261         {
262             let rec create_array vtype dim_list=

```

```
249             match dim_list with
250               | [] -> raise
251               | (Exceptions.Array_parsing_error)
252               | head::[] ->
253                 Array(Primitive(vtype),head)
254                 | head::tail -> Array((create_array
255                   vtype tail),head)
256                   in create_array $1 $2
257
258
259   array_dimension:
260     | LBRACKET INTEGER_LITERAL RBRACKET
261     | { $2 }
262
263   array_dimension_list:
264     | array_dimension
265     | { [$1] }
266     | array_dimension array_dimension_list
267     | { $1 :: $2 }
```

semant.ml

```
1  open Ast
2  open Sast
3  (* open Utils *)
4  open Exceptions
5
6  (* Maps variable name to variable type and value *)
7  module Variable_Map = Map.Make(String);;
8  (* Maps function name to return type *)
9  module Function_Map = Map.Make(String);;
10
11 (* For generating names for the device pointers *)
12 let dev_name_counter = ref 0
13 (* For generating names for the host pointers *)
14 let host_name_counter = ref 0
15 (* For generating names for each ptx map function *)
16 let map_ptx_name_counter = ref 0
17 (* For generating names for each c map function *)
18 let map_c_name_counter = ref 0
19 (* For generating names for each reduce function *)
20 let reduce_c_name_counter = ref 0
21 (* For generating names for ptx reduce function *)
22 let reduce_ptx_name_counter = ref 0
23 (* For generating arg names*)
24 let arg_counter = ref 0
25 (* Generates names for ptx return values *)
26 let ptx_return_counter = ref 0
27 (* Generates names for subroutines*)
28 let subroutine_counter = ref 0
29
30 (* For generating register counters for datatypes *)
31 let signed_int_counter = ref 1
32 let signed_float_counter = ref 1
33 let predicate_counter = ref 1
34 let block_counter = ref 1
35 let pointer_counter = ref 1
36 (*-----Generates Symbols
   -----Based on Counters-----*)
37 let generate_device_pointer_name () =
38   let name = "dev_ptr" ^ (string_of_int
39   ↳ !dev_name_counter) in
40   incr dev_name_counter;
41   name
42
43 let generate_host_pointer_name () =
44   let name = "host_ptr" ^ (string_of_int
45   ↳ !host_name_counter) in
46   incr host_name_counter;
47   name
48
49 let generate_map_c_function_name () =
```

```

48   let name = "map_c" ^ (string_of_int
49   ↵ !map_c_name_counter) in
50   incr map_c_name_counter;
51   name
52
53 let generate_map_ptx_function_name () =
54   let name = "map_ptx" ^ (string_of_int
55   ↵ !map_ptx_name_counter) in
56   incr map_ptx_name_counter;
57   name
58
59 let generate_reduce_c_function_name () =
60   let name = "red_c" ^ (string_of_int
61   ↵ !reduce_c_name_counter) in
62   incr reduce_c_name_counter;
63   name
64
65 let generate_reduce_ptx_function_name () =
66   let name = "red_ptx" ^ (string_of_int
67   ↵ !reduce_ptx_name_counter) in
68   incr reduce_ptx_name_counter;
69   name
70
71 let generate_arg_name () =
72   let name = "arg" ^ (string_of_int !arg_counter) in
73   incr arg_counter;
74   name
75
76 let generate_ptx_return_name () =
77   let name = "func_ret" ^ (string_of_int
78   ↵ !ptx_return_counter) in
79   incr arg_counter;
80   name
81
82 let generate_subroutine_name () =
83   let name = "SUB_ROUT" ^ (string_of_int
84   ↵ !subroutine_counter) in
85   incr subroutine_counter;
86   name
87
88 let get_signed_int_counter () =
89   let orig = !signed_int_counter in
90   incr(signed_int_counter);
91   orig
92
93 let get_signed_float_counter () =
94   let orig = !signed_float_counter in
95   incr(signed_float_counter);
96   orig
97
98 let get_predicate_counter () =
99   let orig = !predicate_counter in
100  incr(predicate_counter);
101  orig
102
103 let get_pointer_counter () =
104   let orig = !pointer_counter in

```

```

96     incr(pointer_counter);
97     orig
98     (*-----Types for Semantic
99        Analysis-----*)
100    (* Three types of functions *)
101    type cuda_function_type =
102      | Kernel_Global
103      | Kernel_Device
104      | Host
105
106    (* Stores information about a function *)
107    type function_info = {
108      (* Host, kernel_device, kernel_global *)
109      function_type : cuda_function_type;
110      function_name : Ast.identifier;
111      (* Function return type and arguments *)
112      function_return_type : Ast.variable_type;
113      function_args : (Ast.variable_type)
114      → list;
115      (* Functions that are called within this function - needs
116      → to be specifically noted for gpu and ptx functions *)
117      dependent_functions : Ast.identifier list;
118      (* Unknown ,possibly constant variables -> for
119      → kernel_device and kernel_global *)
120      unknown_variables : Ast.identifier list;
121    }
122
123    type variable_info = {
124      vtype : Ast.variable_type;
125      register_number : int;
126    }
127
128    (* Stores information about the environment *)
129    type environment = {
130      (* Variables that have been declared in the environment -
131      → stores variable name, variable type *)
132      variable_scope_stack : variable_info Variable_Map.t list;
133      (* List of kernel functions that have been declared in
134      → the environment - info from function_info record *)
135      kernel_function_map : function_info Function_Map.t;
136      (* List of host functions that have been declared in the
137      → environment - info from function_info record *)
138      host_function_map : function_info Function_Map.t;
139      (* Bool specifying whether environment is being evaluated
140      → on the gpu *)
141      is_gpu_env : bool;
142      (*List of functions for higher order functions *)
143      hof_c_function_list : Sast.c_higher_order_fdecl list;
144    }

```

```

137  hof_ptx_function_list :  

138    ↳ Sast.ptx_higher_order_fdecl list;  

139    (* Contains list of ptx_identifiers *)  

140    expression_stack :  

141      ↳ Sast.ptx_literal list list;  

142      return_lit :  

143        ↳ Sast.ptx_literal;  

144    }  

145  

146  (*-----Helper functions to  

147   check variables and functions in the environment  

148   -----*)  

149  

150  let builtin_functions = ["print"];  

151  

152  (* Checks if function is a builtin function *)  

153  (* Used to check function declarations to make sure they  

154   aren't declaring anything with the same name *)  

155  let is_builtin_function id =  

156    List.exists (fun function_name -> function_name = id)  

157    ↳ builtin_functions  

158  

159  (* Creates a function_info record with information *)  

160  let create_function_info ftype rtype args df uv name = {  

161    function_type = ftype;  

162    function_name = Identifier(name);  

163    function_return_type = rtype;  

164    function_args = args;  

165    dependent_functions = df;  

166    unknown_variables = uv;  

167  }  

168  

169  (* Function for adding initializing host function map, adds  

170   builtin functions to host function map*)  

171  let init_host_function_map =  

172    let fmap = Function_Map.empty in  

173    let rec add_functions fmap function_list =  

174      match function_list with  

175        | [] -> fmap  

176        | f_info::tl -> add_functions (Function_Map.add  

177          ↳ (Utils.idtos(f_info.function_name)) f_info fmap) tl  

178      in  

179      let print_function = {  

180        function_type = Host;  

181        function_name = Ast.Identifier("print");  

182        function_return_type = Ast.Primitive(Ast.Void);  

183        function_args = [Ast.Primitive(Ast.String)];  

184        dependent_functions = [];  

185        unknown_variables = [];  

186      }  

187      in

```

```

182 let random_function = {
183     function_type = Host;
184     function_name = Ast.Identifier("random");
185     function_return_type = Ast.Primitive(Ast.Integer);
186     function_args = [];
187     dependent_functions = [];
188     unknown_variables = [];
189 }
190 in
191 (* let create_builtin_function = (create_function_info
192    → Host (Ast.Primitive(Ast.Void)) [] [] []) in
193    let builtin_function_info_structs = List.map
194    → create_builtin_function builtin_functions in *)
195 add_functions fmap [print_function; random_function]
196
197 let make_ptx_id name reg num write_reg is_ptr =
198 {
199     var_name      = name; (* Name as passed as a param or
200    → declared *)
201     reg_name      = reg; (* Register name it is stored in *)
202     reg_num       = num;
203     write_reg     = write_reg;
204     is_ptr        = is_ptr;
205 }
206
207 (* Creates a new environment *)
208 let init_env = {
209     variable_scope_stack      = Variable_Map.empty :: [];
210     kernel_function_map       = Function_Map.empty;
211     host_function_map         = init_host_function_map;
212     is_gpu_env                = false;
213     (* Two lists that stores the new higher order functions
214    → we need to add*)
215     hof_c_function_list      = [];
216     hof_ptx_function_list     = [];
217     expression_stack          = [[]];
218     return_lit                =
219     → Sast.Ptx_Identifier_Literal(make_ptx_id (Ast.Identifier
220     → "") "") 0 true true )
221 }
222
223 (* Updates the environment *)
224 let update_env vscope_stack kfmap hfmap is_gpu hof_c_list
225    → hof_ptx_list ptx_e_stack rlit= {
226     variable_scope_stack      = vscope_stack;
227     kernel_function_map       = kfmap;
228     host_function_map         = hfmap;
229     is_gpu_env                = is_gpu;
230     hof_c_function_list      = hof_c_list;
231     hof_ptx_function_list     = hof_ptx_list;
232     expression_stack          = ptx_e_stack;
233     return_lit                = rlit;
234 }

```

```

229
230
231 (* Pushes a new scope on top of the variable_scope_stack
232   *)
233 let push_scope env =
234   update_env (Variable_Map.empty :::
235     ← env.variable_scope_stack) env.kernel_function_map
236   ← env.host_function_map env.is_gpu_env
237   ← env.hof_c_function_list env.hof_ptx_function_list
238   ← env.expression_stack env.return_lit
239
240 (* Pops a scope from the top of the variable_scope_stack *)
241 let pop_scope env =
242   match env.variable_scope_stack with
243     | [] -> raise
244       Exceptions.Cannot_pop_empty_variable_scope_stack
245     | local_scope :: tail ->
246       update_env tail env.kernel_function_map
247       ← env.host_function_map env.is_gpu_env
248       ← env.hof_c_function_list env.hof_ptx_function_list
249       ← env.expression_stack env.return_lit
250
251 let update_scope updated_scope env =
252   let env = pop_scope env in
253     update_env (updated_scope :: env.variable_scope_stack)
254     ← env.kernel_function_map env.host_function_map
255     ← env.is_gpu_env env.hof_c_function_list
256     ← env.hof_ptx_function_list env.expression_stack
257     ← env.return_lit
258
259 let update_kernel_fmap f_info env =
260   let new_kfmap = Function_Map.add
261     (Utils.idtos(f_info.function_name)) f_info
262   ← env.kernel_function_map in
263   update_env env.variable_scope_stack new_kfmap
264   ← env.host_function_map env.is_gpu_env
265   ← env.hof_c_function_list env.hof_ptx_function_list
266   ← env.expression_stack env.return_lit
267
268 let update_host_fmap f_info env =
269   let new_hfmap = Function_Map.add
270     (Utils.idtos(f_info.function_name)) f_info
271   ← env.host_function_map in
272   update_env env.variable_scope_stack
273   ← env.kernel_function_map new_hfmap env.is_gpu_env
274   ← env.hof_c_function_list env.hof_ptx_function_list
275   ← env.expression_stack env.return_lit
276
277 let update_hof_lists hof_c_fdecl hof_ptx_fdecl env =

```

```

255     update_env env.variable_scope_stack
256     ← env.kernel_function_map env.host_function_map
257     ← env.is_gpu_env
258     ← (List.rev(hof_c_fdecl::List.rev(env.hof_c_function_list)))
259     ← (List.rev(hof_ptx_fdecl::List.rev(env.hof_ptx_function_list)))
260     ← env.expression_stack env.return_lit
261
262 let update_return_lit ptx_lit env =
263   update_env env.variable_scope_stack
264   ← env.kernel_function_map env.host_function_map
265   ← env.is_gpu_env env.hof_c_function_list
266   ← env.hof_ptx_function_list env.expression_stack ptx_lit
267
268 let pop_expression_stack env =
269   match env.expression_stack with
270   | [] -> env
271   | hd::tl -> update_env env.variable_scope_stack
272   ← env.kernel_function_map env.host_function_map
273   ← env.is_gpu_env env.hof_c_function_list
274   ← env.hof_ptx_function_list tl env.return_lit
275
276 let push_expression_stack env =
277   update_env env.variable_scope_stack
278   ← env.kernel_function_map env.host_function_map
279   ← env.is_gpu_env env.hof_c_function_list
280   ← env.hof_ptx_function_list ([]::env.expression_stack)
281   ← env.return_lit
282
283 let update_expression_stack lit env =
284   let update = lit::(List.hd env.expression_stack) in
285   let env = pop_expression_stack env in
286     update_env env.variable_scope_stack
287     ← env.kernel_function_map env.host_function_map
288     ← env.is_gpu_env env.hof_c_function_list
289     ← env.hof_ptx_function_list
290     ← (update::env.expression_stack) env.return_lit
291
292 (* Retrieves nth element from head list of expression stack
293    *)
294 let get_from_expression_stack nth env =
295   if nth > (List.length (List.hd env.expression_stack)) ||
296   nth < 0 then raise
297   ← Exceptions.Invalid_expression_stack_access
298   else match env.expression_stack with
299   | [] -> raise
300   ← Exceptions.Cannot_access_empty_expression_stack
301   | hd::tl -> List.nth (List.hd env.expression_stack)
302   ← nth
303
304 (* Checks if variable has been declared - is valid - in the
305    scope *)
306 let is_variable_in_scope id env =
307   let rec check_scopes scope_stack =

```

```

283     match scope_stack with
284     | [] -> false
285     | [scope] ->
286         if env.is_gpu_env then false
287         else (Variable_Map.mem id scope)
288     | scope :: larger_scopes ->
289         if (Variable_Map.mem id scope) then true
290         else check_scopes larger_scopes
291     in check_scopes env.variable_scope_stack
292
293
294 (* Searches variable in scope for CUDA C and returns its
295    ~ type *)
295 let get_variable_type id env =
296     let rec check_scopes scope_stack =
297         match scope_stack with
298         | [] -> raise (Exceptions.Variable_not_found_in_scope
299             ~ (id))
300         | scope::larger_scopes ->
301             if Variable_Map.mem id scope then
302                 (Variable_Map.find id scope).vtype
303             else
304                 check_scopes larger_scopes
305     in check_scopes env.variable_scope_stack
306
307 let get_variable_info id env =
308     let rec check_scopes scope_stack =
309         match scope_stack with
310         | [] -> raise (Exceptions.Variable_not_found_in_scope
311             ~ (id))
312         | scope::larger_scopes ->
313             if Variable_Map.mem id scope then
314                 Variable_Map.find id scope
315             else
316                 check_scopes larger_scopes
317     in check_scopes env.variable_scope_stack
318
319 let update_variable_register id reg_num env =
320     let old_info = get_variable_info id env in
321     let new_info = { vtype = old_info.vtype; register_number
322         = reg_num; } in
323     let new_vmap = Variable_Map.add id new_info (List.hd
324         env.variable_scope_stack) in
325     update_scope new_vmap env
326
327 (* Helper function that returns checks types are the same
328    ~ *)
329 let same_types t1 t2 = (t1 = t2)
330
331 (* Checks if function is valid in the environment *)
332 let is_function_in_scope id env =
333     if env.is_gpu_env = true then (Function_Map.mem id
334         env.kernel_function_map)

```

```

329 else (Function_Map.mem id env.host_function_map) ||
→ (Function_Map.mem id env.kernel_function_map)
330
331
332 (* Searches for function called in function call and
→ returns information about the function *)
333 let get_function_info id env =
334   if env.is_gpu_env = true then
335     (if (Function_Map.mem id env.kernel_function_map)
→ then
          (Function_Map.find id env.kernel_function_map)
337     else raise (Exceptions.Function_not_defined (id)))
338   else
339     (if (Function_Map.mem id env.host_function_map)
→ then
          (Function_Map.find id env.host_function_map)
341     else if (Function_Map.mem id
→ env.kernel_function_map) then
          (Function_Map.find id env.kernel_function_map)
343     else raise (Exceptions.Function_not_defined (id)))
344
345 (* ----- Functions for
→ Checking Ast -----*)
346 (* Checks a variable declaration and initialization to
→ ensure variable hasn't already been declared *)
347 let check_already_declared id env =
348   if ((is_variable_in_scope id env) = true) then true else
→ false
349
350 (* Helper function that performs type inference for
→ expressions *)
351 let rec infer_type expression env=
352   let f type1 type2 =
353     match type1 with
354       | Some(t) -> (if t = type2 then Some(t)
355                     else raise (Exceptions.Type_mismatch
→ "wrong types"))
356       | None -> Some(type2) in
357 let match_type expression_list =
358   let a = List.fold_left f None expression_list in
359   match a with
360     | Some(t) -> t
361     | None -> raise
→ Exceptions.Empty_array_expression_list in
362 match expression with
363   | Ast.String_Literal(_) -> Ast.Primitive(Ast.String)
364   | Ast.Integer_Literal(_) -> Ast.Primitive(Ast.Integer)
365   | Ast.Floating_Point_Literal(_) ->
→ Ast.Primitive(Ast.Float)
366   | Ast.Boolean_Literal(_) -> Ast.Primitive(Ast.Boolean)
367   | Ast.Array_Literal(expr_list) ->
      let f expression = infer_type expression env in

```

```

369      Ast.Array (match_type (List.map f
370    ← expr_list), (List.length expr_list))
371    | Ast.Identifier_Literal (id) ->
372      if (check_already_declared (Utils.idtos id) env) =
373        false then raise
374      (Exceptions.Variable_not_found_in_scope ((Utils.idtos
375        id)) )
376      else (get_variable_type (Utils.idtos id) env)
377    | Ast.Binop (e1, op, e2) ->
378      (match op with
379        | Ast.And | Ast.Or | Ast.Xor ->
380          Ast.Primitive (Ast.Boolean)
381        | _ -> if (same_types (infer_type e1 env)
382          (infer_type e2 env)) = true then infer_type e1 env
383          else (raise
384            (Exceptions.Type_mismatch ("Binop types don't match")))
385        )
386      | Ast.Cast (vtype, e) -> vtype
387      | Ast.Unop (e, unop) -> infer_type e env
388      | Ast.Array_Accessor (e1, e_list, is_lvalue) ->
389        (* Check e1 is an array *)
390        (match infer_type e1 env with
391          | Ast.Array (t, n) -> ()
392          | _ -> (raise
393            (Exceptions.Not_an_array_expression)))
394        );
395        (* Check valid access *)
396        let rec get_array_type arr dim_list =
397          match dim_list with
398            | [] -> raise Exceptions.Empty_array_access
399            | hd :: tl ->
400              (match arr with
401                | Ast.Array (t, n) -> t
402                | _ -> raise Invalid_array_expression
403              )
404            | hd :: tl ->
405              (match arr with
406                | Ast.Array (t, n) -> get_array_type t tl
407                | _ -> raise
408                  Exceptions.Invalid_array_expression
409              )
410            in get_array_type (infer_type e1 env) e_list
411        | Ast.Ternary (e1, e2, e3) ->
412          if (same_types (infer_type e1 env) (infer_type e2
413            env)) = true then infer_type e1 env else (raise
414            (Exceptions.Type_mismatch ("Ternary doesn't return same
415              type")))
416        | Ast.Higher_Order_Function_Call (hof) ->
417          let f_info = get_function_info (Utils.idtos
418            hof.kernel_function_name) env in
419          let vtype = infer_type (List.hd hof.input_arrays) env
420          in
421            let length = match vtype with

```

```

408         | Ast.Primitive(p) -> raise
409     ↳ Exceptions.Invalid_array_expression
410         | Ast.Array(t,n) -> n
411         in
412         Ast.Array(f_info.function_return_type,length)
413     | Ast.Function_Call(id,e_list) ->
414         let f_info = get_function_info (Utils.idtos id) env
415     ↳ in
416         f_info.function_return_type
417
418 (* Check that array has only one dimension - used for
419    ↳ certain operations *)
420 let is_one_layer_array expression env =
421     match expression with
422     | Ast.Array_Literal(e_list) as array_literal ->
423         let arr = infer_type array_literal env in
424         (match arr with
425         | Ast.Array(vtype,size) -> if size > 1 then false
426     ↳ else true
427         | _ -> raise Exceptions.Not_an_array_expression)
428     | _ -> raise Exceptions.Not_an_array_expression
429
430
431 (* Helper function that returns a list of dimensions for an
432    ↳ array variable type *)
433 let rec get_array_dimensions vtype dimensions =
434     match vtype with
435     | Ast.Array(t,n) ->
436         get_array_dimensions t
437         ↳ (List.rev(n::List.rev(dimensions)))
438     | Ast.Primitive(p) -> dimensions
439     (*    | _ -> raise Exceptions.Unknown_variable_type *)
440
441
442 (* ----- Functions for
443    ↳ converting ast to sast (Also performs advanced
444    ↳ checking) -----*)
445 (* Converts a list of something to another list *)
446 let rec convert_list func ast_list sast_list env =
447     match ast_list with
448     | [] -> sast_list,env
449     | hd::tl ->
450         let sast_type, env = func hd env in
451         convert_list func tl (List.rev
452         ↳ (sast_type::List.rev(sast_list))) env
453
454
455 (* Generates a register for every variable type, keeps a
456    ↳ counter for the types as well *)
457 let generate_reg vtype =

```

```

451     match vtype with
452     | Ast.Primitive(Ast.Integer) ->
453     ↳ "%si", (get_signed_int_counter())
454     | Ast.Primitive(Ast.Float) -> "%fl",
455     ↳ get_signed_float_counter()
456     | Ast.Primitive(Ast.Boolean) -> "%pr",
457     ↳ get_predicate_counter()
458     | Ast.Primitive(Ast.String) -> raise
459     ↳ Exceptions.NO_STRINGS_ALLOWED_IN_GDECL
460     | Ast.Primitive(Ast.Void) -> raise
461     ↳ Exceptions.Void_type_in_gdecl
462     | Ast.Array(vtype, size) -> "%ptr",
463     ↳ get_pointer_counter()

464 (*Gets the string of the register type*)
465 let get_reg_type vtype =
466   match vtype with
467   | Ast.Primitive(Ast.Integer) -> "%si"
468   | Ast.Primitive(Ast.Float) -> "%fl"
469   | Ast.Primitive(Ast.Boolean) -> "%pr"
470   | Ast.Primitive(Ast.String) -> raise
471   ↳ Exceptions.NO_STRINGS_ALLOWED_IN_GDECL
472   | Ast.Primitive(Ast.Void) -> raise
473   ↳ Exceptions.Void_type_in_gdecl
474   | Ast.Array(vtype, size) -> "%ptr"

475 (* Checks statement order - nothing follows a return ,
476    ↳ break, or continue in a block*)
477 let rec good_statement_order stmt_list =
478   match stmt_list with
479   | [] -> true
480   | hd ::[] -> true
481   | hd :: tl ->
482     match hd with
483     | Ast.Return_Void | Ast.Continue | Ast.Break ->
484     ↳ false
485     | Ast.Return(e) -> false
486     | _ -> good_statement_order tl

487 (* Binop *)
488 let convert_to_c_binop binop env =
489   match binop with
490   | Ast.Add -> Sast.Add,env
491   | Ast.Subtract -> Sast.Subtract,env
492   | Ast.Multiply -> Sast.Multiply,env
493   | Ast.Divide -> Sast.Divide,env
494   | Ast.Modulo -> Sast.Modulo,env
495   | Ast.And -> Sast.And,env
496   | Ast.Or -> Sast.Or,env
497   | Ast.Xor -> Sast.Xor,env
498   | Ast.Equal -> Sast.Equal,env
499   | Ast.Not_Equal -> Sast.Not_Equal,env

```

```

493   | Ast.Greater_Than -> Sast.Greater_Than,env
494   | Ast.Less_Than -> Sast.Less_Than,env
495   | Ast.Greater_Than_Equal -> Sast.Greater_Than_Equal,env
496   | Ast.Less_Than_Equal -> Sast.Less_Than_Equal,env
497   | Ast.Bitshift_Right -> Sast.Bitshift_Right,env
498   | Ast.Bitshift_Left -> Sast.Bitshift_Left,env
499   | Ast.Bitwise_Or -> Sast.Bitwise_Or,env
500   | Ast.Bitwise_And -> Sast.Bitwise_And,env
501
502 let convert_to_ptx_binop binop env =
503   match binop with
504   | Ast.Add -> Sast.Ptx_Add,env
505   | Ast.Subtract -> Sast.Ptx_Subtract,env
506   | Ast.Multiply -> Sast.Ptx_Multiply,env
507   | Ast.Divide -> Sast.Ptx_Divide,env
508   | Ast.Modulo -> Sast.Ptx_Modulo,env
509   | Ast.And -> Sast.Ptx_And,env
510   | Ast.Or -> Sast.Ptx_Or,env
511   | Ast.Xor -> Sast.Ptx_Xor,env
512   | Ast.Equal -> Sast.Ptx_Equal,env
513   | Ast.Not_Equal -> Sast.Ptx_Not_Equal,env
514   | Ast.Greater_Than -> Sast.Ptx_Greater_Than,env
515   | Ast.Less_Than -> Sast.Ptx_Less_Than,env
516   | Ast.Greater_Than_Equal ->
517     ← Sast.Ptx_Greater_Than_Equal,env
518   | Ast.Less_Than_Equal -> Sast.Ptx_Less_Than_Equal,env
519   | Ast.Bitshift_Right -> Sast.Ptx_Bitshift_Right,env
520   | Ast.Bitshift_Left -> Sast.Ptx_Bitshift_Left,env
521   | Ast.Bitwise_Or -> Sast.Ptx_Bitwise_Or,env
522   | Ast.Bitwise_And -> Sast.Ptx_Bitwise_And,env
523
524 (* Unop *)
525 let convert_to_c_unop unop env =
526   match unop with
527   | Ast.Not -> Sast.Not,env
528   | Ast.Negate -> Sast.Negate,env
529   | Ast.Plus_Plus -> Sast.Plus_Plus,env
530   | Ast.Minus_Minus -> Sast.Minus_Minus,env
531
532 let convert_to_ptx_unop unop env =
533   match unop with
534   | Ast.Not -> Sast.Ptx_Not,env
535   | Ast.Negate -> Sast.Ptx_Negate,env
536   | Ast.Plus_Plus -> Sast.Ptx_Plus_Plus,env
537   | Ast.Minus_Minus -> Sast.Ptx_Minus_Minus,env
538
539 (* Datatype *)
540 let convert_to_c_data_type dtype env =
541   match dtype with
542   | Ast.Integer -> Sast.Integer,env

```

```

544     | Ast.Float -> Sast.Float,env
545     | Ast.String -> Sast.String,env
546     | Ast.Boolean -> Sast.Boolean,env
547     | Ast.Void -> Sast.Void,env
548
549 let convert_to_ptx_data_type dtype env =
550   match dtype with
551     | Ast.Integer -> Sast.S32,env
552     | Ast.Float -> Sast.F32,env
553     | Ast.Boolean -> Sast.Pred,env
554     | Ast.String -> raise
555     → Exceptions.NO_STRINGS_ALLOWED_IN_GDECL
556     | Ast.Void -> Sast.Ptx_Void,env
557
558 (* Variable Type *)
559 let rec convert_to_c_variable_type vtype env =
560   match vtype with
561     | Ast.Primitive(p) ->
562       let c_p,env = convert_to_c_data_type p env in
563       Sast.Primitive(c_p),env
564     | Ast.Array(t,n) ->
565       let array_dims = get_array_dimensions vtype [] in
566       let inside,env = (match t with
567         | Ast.Array(t,n) ->
568           convert_to_c_variable_type t env
569         | Ast.Primitive(p) ->
570           let c_p,env= convert_to_c_data_type p env
571           in
572             Sast.Primitive(c_p),env
573         ) in
574         Sast.Array(inside, array_dims),env
575
576 (* TO IMPLEMENT *)
577 let rec convert_to_ptx_variable_type vtype env =
578   match vtype with
579     | Ast.Primitive(p) ->
580       let p2,env = convert_to_ptx_data_type p env in
581       Sast.Ptx_Primitive(p2),env
582     | Ast.Array(t,n) ->
583       let array_dims = get_array_dimensions vtype [] in
584       let c_t,env = convert_to_ptx_variable_type t env
585       in
586         Sast.Ptx_Array(c_t,array_dims),env
587
588 (* Variable Declarations *)
589 let convert_to_c_vdecl vdecl env =
590   match vdecl with
591     | Ast.Variable_Declaration(vtype,id) ->
592       if(check_already_declared (Utils.idtos(id)) env)
593       = true then (raise
594       → (Exceptions.Variable_already_declared
595       → (Utils.idtos(id)))) else

```

```

591      let v_info = { vtype = vtype; register_number =
592      ↵ 0; } in
593      let new_vmap = Variable_Map.add (Utils.idtos
594      ↵ id) v_info (List.hd env.variable_scope_stack) in
595      let env = update_scope new_vmap env in
596      let c_vtype, env = convert_to_c_variable_type
597      ↵ vtype env in
598      Sast.Variable_Declaration(c_vtype,id),env
599
600 (*Statements are found as parameters in the defg*)
601 let convert_to_ptx_param vdecl env =
602   match vdecl with
603     | Ast.Variable_Declaration(vtype,id) ->
604       if(check_already_declared (Utils.idtos id) env) =
605       true then raise (raise
606       (Exceptions.Variable_already_declared
607       (Utils.idtos(id)))))
608     else
609       (* Generate a register for each parameter - since
610       its parameters, arrays are pointers *)
611       let reg_name, reg_num = generate_reg vtype in let
612       v_info = { vtype = vtype; register_number = reg_num; }
613       in
614         (* Update the variable in our variable map*)
615         let new_vmap = Variable_Map.add (Utils.idtos id)
616         ↵ v_info (List.hd env.variable_scope_stack) in
617         let env = update_scope new_vmap env in
618           (* Generates a PTX identifier that we want to use
619           *)
620           let is_array = match vtype with Ast.Primitive(p)
621           ↵ -> false | Ast.Array(t,n) -> true in
622           let ptx_id = make_ptx_id id reg_name reg_num
623           ↵ false is_array in
624             (match vtype with
625               (* Convert these types to have state space
626               param*)
627               | Ast.Primitive(p) ->
628                 Sast.Ptx_Vdecl(Sast.Param,
629                 fst(convert_to_ptx_variable_type vtype env),ptx_id),env
630               | Ast.Array(t,n) ->
631                 Sast.Ptx_Vdecl(Sast.Param,
632                 fst(convert_to_ptx_variable_type vtype env),ptx_id),env
633               )
634
635
636
637 (*Statements are found in the body of the defg*)
638 let convert_to_ptx_vdecl vdecl env =
639   match vdecl with
640     | Ast.Variable_Declaration(vtype,id) ->
641       if(check_already_declared (Utils.idtos id) env) =
642       true then raise (raise
643       (Exceptions.Variable_already_declared
644       (Utils.idtos(id)))))


```

```

623     else
624         (* Generate a register name for the variable and add
625            it to our vmap*)
625         let reg_name, reg_num = generate_reg vtype in let
626         v_info = { vtype = vtype; register_number = reg_num; }
627         in
628         let new_vmap = Variable_Map.add (Utils.idtos id)
629         v_info (List.hd env.variable_scope_stack) in
630         let env = update_scope new_vmap env in
631         let is_array = match vtype with Ast.Primitive(p) ->
632         false | Ast.Array(t,n) -> true in
633         let ptx_id = make_ptx_id id reg_name reg_num true
634         is_array in
635             (match vtype with
636                 (* Predicates can only be declared in
637                    register space *)
638                 | Ast.Primitive(Ast.Boolean) ->
639                 Sast.Ptx_Vdecl(Sast.Register,
640                 fst(convert_to_ptx_variable_type vtype env),ptx_id),env
641                 | Ast.Primitive(p)
642                 ->Sast.Ptx_Vdecl(Sast.Local,fst(convert_to_ptx_variable_type
643                 vtype env),ptx_id),env
644                 | Ast.Array(t,n) ->
645                 Sast.Ptx_Vdecl(Sast.Local,fst(convert_to_ptx_variable_type
646                 vtype env),ptx_id),env
647             )
648
649 let same_types_list type_list =
650     let main_type = (List.hd type_list) in
651     let rec check_each_type main_type type_list =
652         (match type_list with
653             | [] -> true
654             | hd::tl ->
655                 if(same_types main_type hd) then (check_each_type
656                 main_type tl)
657                 else raise
658                 Exceptions.Array_elements_not_all_same_type
659             )
660     in check_each_type main_type type_list
661
662 (* Creates list of sast structs storing information about
663    constants for higher order function *)
664 let rec get_constants_info constant_list c_constant_list
665     env =
666     match constant_list with
667     | [] -> c_constant_list
668     | hd::tl ->
669         (match hd with
670             | Ast.Constant(id,e) ->
671                 let vtype = infer_type e env in
672                 (* Name of constant in defg gpu function*)
673                 let h_name =
674                 Ast.Identifier(generate_host_pointer_name ()) in

```

```

658             (* Name of constant when input as an argument
659             *)
660             let a_name = Ast.Identifier(generate_arg_name
661             () ) in
662             let v_type, env = convert_to_c_variable_type
663             vtype env in
664             (* Sast.type*)
665             let constant_info = {
666               variable_type = v_type;
667               host_name = h_name;
668               arg_name = a_name;
669               kernel_name = id;
670             } in get_constants_info tl
671             (List.rev(constant_info)::List.rev(c_constant_list)))
672             env
673           )
674
675 (* Creates list of sast structs storing information about
676   info arrays from higher order function *)
677 let rec get_input_arrays_info input_arrays var_info_list
678   env =
679   match input_arrays with
680   | [] -> var_info_list
681   | hd::tl ->
682     (
683       match infer_type hd env with
684       | Ast.Array(t, n) ->
685         let h_name =
686           Ast.Identifier(generate_host_pointer_name ()) in
687         let k_name =
688           Ast.Identifier(generate_device_pointer_name ()) in
689         let a_name = Ast.Identifier(generate_arg_name
690           () ) in
691         let vtype, env = convert_to_c_variable_type(
692           infer_type hd env) env in
693         let var_info = {
694           variable_type = vtype;
695           host_name = h_name;
696           kernel_name = k_name;
697           arg_name = a_name;
698         }
699         in get_input_arrays_info tl
700         (List.rev(var_info)::List.rev(var_info_list))) env
701         | _ -> raise
702           Exceptions.Nonarray_argument_passed_into_higher_order_function
703         )
704
705 (* Creates sast struct storing information about return
706   array from higher order function *)
707 let get_return_array_info kfunc_id length env =
708   let f_info = get_function_info kfunc_id env in
709   let return_vtype, env = convert_to_c_variable_type
710   (Ast.Array(f_info.function_return_type, length)) env in

```

```

696  let h_name           =
697    ↳ Ast.Identifier(generate_host_pointer_name ()) in
698  let k_name           =
699    ↳ Ast.Identifier(generate_device_pointer_name ()) in
700  let a_name           = Ast.Identifier(generate_arg_name
701    ↳ () ) in
702  let var_info          =
703    ↳ variable_type      = return_vtype;
704    ↳ host_name          = h_name;
705    ↳ kernel_name         = k_name;
706    ↳ arg_name           = a_name;
707  } in
708  var_info
709
710 (* Main function for creating the C map function (when we
711   ↳ see a map function call) *)
712 let make_hof_c_fdecl hof_call env =
713  let arr_length =
714    ↳ let arr = infer_type (List.hd
715      ↳ hof_call.input_arrays) env in
716      ↳ (match arr with
717        | Ast.Array(t,n) -> n
718        | _ -> raise
719        ↳ Exceptions.Not_an_array_expression)
720      in
721  match Utils.idtos(hof_call.hof_type) with
722    | "map" ->
723      let kfunc_name =
724        ↳ Ast.Identifier(generate_map_ptx_function_name ()) in
725        ↳ {
726          ↳ higher_order_function_type
727          ↳ = Ast.Identifier("map");
728          ↳ higher_order_function_name
729          ↳ = Ast.Identifier(generate_map_c_function_name ());
730          ↳ applied_kernel_function
731          ↳ = kfunc_name;
732          ↳ higher_order_function_constants
733          ↳ = get_constants_info hof_call.constants [] env;
734          ↳ array_length
735          ↳ = arr_length;
736          ↳ input_arrays_info
737          ↳ = get_input_arrays_info hof_call.input_arrays [] env;
738          ↳ return_array_info
739          ↳ = get_return_array_info
740          ↳ (Utils.idtos(hof_call.kernel_function_name)) arr_length
741          ↳ env;
742          ↳ called_functions
743          ↳ = [hof_call.kernel_function_name]
744          ↳ }
745        | "reduce" ->
746          let kfunc_name =
747            ↳ Ast.Identifier(generate_reduce_ptx_function_name ()) in
748            ↳ {

```

```

731             higher_order_function_type
732             ← = Ast.Identifier("reduce");
733             higher_order_function_name
734             ← = Ast.Identifier(generate_reduce_c_function_name ());
735             applied_kernel_function
736             ← = kfunc_name;
737             higher_order_function_constants
738             ← = get_constants_info hof_call.constants [] env;
739             array_length
740             ← = arr_length;
741             input_arrays_info
742             ← = get_input_arrays_info hof_call.input_arrays [] env;
743             return_array_info
744             ← = get_return_array_info
745             (Utils.idtos(hof_call.kernel_function_name)) arr_length
746             env;
747             called_functions
748             ← = [hof_call.kernel_function_name]
749             }
750             | _ -> raise
751             (Exceptions.Unknown_higher_order_function_call
752             (Utils.idtos hof_call.hof_type))
753
754 (* TO IMPLEMENT
755 Converts c_kernel_variable_info to ptx_kernel_variable_info
    *)
756 let convert_to_register_declaration dtype id num_reg =
757 {
758     reg_type      = dtype;
759     reg_id        = id;
760     num_registers = num_reg;
761 }
762
763 let hof_param_reg_counter = ref 0
764
765 let change_to_ptx_vdecl ckv_info =
766 let change_to_ptx_data_type sast_c_dtype =
767 match sast_c_dtype with
768 | Sast.Integer -> S32
769 | Sast.Float   -> F32
770 | Sast.Boolean -> Pred
771 | Sast.Void    -> Ptx_Void
772 | _ -> raise Exceptions.NO_STRINGS_ALLOWED_IN_GDECL
773 in
774 let rec get_vtype sast_c_vtype =
775     (match sast_c_vtype with
776     | Sast.Primitive(p) ->
777         Ptx_Primitive(change_to_ptx_data_type p)
778     | Sast.Array(t,n) -> Ptx_Array(get_vtype t, n)
779     )
780 in

```

```

768     incr
    ↳ hof_param_reg_counter; (Sast.Ptx_Vdecl (Sast.Global, (get_vtype
    ↳ ckv_info.variable_type), (make_ptx_id
    ↳ ckv_info.kernel_name "ptr" !hof_param_reg_counter false
    ↳ false)))
769 (* Creates a ptx_fdecl based on the hof_c_fdecl*)
770 let make_hof_ptx_fdecl hof_c_fdecl hof env=
771   let regs = [ convert_to_register_declaration (Sast.Pred)
    ↳ "pred" 2;
    ↳ convert_to_register_declaration (Sast.U64)
    ↳ "ptr" (2*((List.length hof.input_arrays)+2));
    ↳ convert_to_register_declaration (Sast.S32)
    ↳ "mytid" 2;
    ↳ convert_to_register_declaration (Sast.S32)
    ↳ "rtype" 2;
    ↳ convert_to_register_declaration (Sast.S32)
    ↳ "vlc" ((List.length hof.input_arrays)+1);
    ↳ convert_to_register_declaration (Sast.S32)
    ↳ "asize" 2;
772   ] in
773   {
774     ptx_higher_order_function_type =
775     ↳ hof_c_fdecl.higher_order_function_type;
776     ptx_higher_order_function_name =
777     ↳ hof_c_fdecl.applied_kernel_function;
778     ptx_applied_kernel_function =
779     ↳ hof.kernel_function_name;
780     ptx_higher_order_function_constants =
781     ↳ List.map change_to_ptx_vdecl
782     ↳ (hof_c_fdecl.higher_order_function_constants);
783     ptx_array_length =
784     ↳ hof_c_fdecl.array_length;
785     ptx_input_arrays_info =
786     ↳ List.map change_to_ptx_vdecl
787     ↳ (hof_c_fdecl.input_arrays_info);
788     ptx_return_array_info =
789     ↳ change_to_ptx_vdecl hof_c_fdecl.return_array_info;
790     ptx_called_functions =
791     ↳ [hof.kernel_function_name];
792     ptx_register_decls =
793     ↳ regs;
794   }
795
796   let rec get_types args types env =
797     match args with
798       | [] -> types
799       | hd::tl -> get_types tl (List.rev((infer_type hd
    ↳ env)::List.rev types)) env
800
801   let rec add_lists list_lists newlist=
802     match list_lists with
803       | [] -> newlist
804       | hd::tl -> add_lists tl (newlist @ hd)

```

```

800
801 let rec convert_to_c_expression e env =
802   let rec flatten_array e flattened_array env =
803     (match e with
804      | Ast.Array_Literal(e_list) ->
805        (match List.hd e_list with
806         | Ast.Array_Literal(e1_list) ->
807           let list_of_flattened_arrays = List.map
808             (fun x-> flatten_array x [] env) e_list in
809           add_lists list_of_flattened_arrays []
810         | _ -> flattened_array @ (List.map (fun x ->
811           fst(convert_to_c_expression x env)) e_list)
812             )
813       | _ -> raise Exceptions.Not_an_array_expression
814     )
815   in
816     match e with
817       | Ast.Function_Call(id,e_list) ->
818         (* Check that function exists in environment *)
819         if (is_function_in_scope (Utils.idtos id) env) =
820           false then (raise (Exceptions.Function_not_defined
821           (Utils.idtos id)));
822           (* Check that function arguments match that of
823           function declaration *)
824           let f_info = (get_function_info (Utils.idtos id)
825             env) in
826             let f_arg_types = f_info.function_args in
827               let check_args expected_arg_types f_args =
828                 List.map2 same_types expected_arg_types
829                 f_args in
830                   ignore(check_args f_arg_types (get_types e_list []
831                     env));
832                     (* Convert *)
833                     let c_e_list = List.map (fun x ->
834                       fst(convert_to_c_expression x env)) e_list
835                         in Sast.Function_Call(id,c_e_list),env
836                           | Ast.String_Literal(s) -> Sast.String_Literal(s),env
837                           | Ast.Integer_Literal(i) ->
838                             Sast.Integer_Literal(i),env
839                           | Ast.Boolean_Literal(b) ->
840                             Sast.Boolean_Literal(b),env
841                           | Ast.Floating_Point_Literal(f) ->
842                             Sast.Floating_Point_Literal(f),env
843                           | Ast.Array_Literal(e_list) ->
844                             (* Check all elements of the array are the same
845                               type *)
846                             let type_list = List.map (fun x -> infer_type x
847                               env) e_list in
848                               ignore(same_types_list type_list);
849                               (* Get array dimensions and pass to sast *)
850                               let arr = Ast.Array(infer_type (List.hd e_list)
851                                 env ,List.length e_list) in
852                                   let array_dim = get_array_dimensions arr [] in

```

```

838          (* Convert *)
839          let c_e_list = flatten_array e [] env in
840            Sast.Array_Literal(c_e_list, array_dim), env
841        | Ast.Identifier_Literal(id) ->
842          if (check_already_declared (Utils.idtos id) env) =
843            false then raise
844            (Exceptions.Variable_not_found_in_scope ( Utils.idtos
845              id))
846            else Sast.Identifier_Literal(id), env
847        | Ast.Cast(vtype, e) ->
848          let c_vtype, env = convert_to_c_variable_type
849            vtype env in
850            let c_e, env = convert_to_c_expression e env in
851              Sast.Cast(c_vtype, c_e), env
852        | Ast.Unop(e, op) ->
853          (match op with
854            | Ast.Not ->
855              if ((infer_type e env) =
856                Ast.Primitive(Ast.Boolean)) = false then raise
857                (Exceptions.Type_mismatch("Must use boolean expression
858                  with boolean unop"))
859              else
860                let c_e, env = convert_to_c_expression e env
861                in
862                  let c_op, env = convert_to_c_unop op env in
863                    Sast.Unop(c_e, c_op), env
864                  | _ ->
865                      if ((infer_type e env) = (Ast.Primitive
866                        (Ast.String))) then raise
867                        (Exceptions.Cannot_perform_operation_on_string
868                          (Utils.unary_operator_to_string op))
869                      else
870                        let c_e, env = convert_to_c_expression e env
871                        in
872                          let c_op, env = convert_to_c_unop op env in
873                            Sast.Unop(c_e, c_op), env
874                          )
875        | Ast.Ternary(e1, e2, e3) ->
876          (*Check e1 and e3 match*)
877          if (same_types (infer_type e1 env) (infer_type e3
878            env)) = false then raise
879            (Exceptions.Type_mismatch("Ternary expression don't
880              match"))
881          else
882            (*Check e2 is boolean*)
883            if (same_types (infer_type e2 env)
884              (Ast.Primitive(Ast.Boolean))) = false then (raise
885                (Exceptions.Conditional_must_be_a_boolean))
886                else
887                  let c_e1, env = convert_to_c_expression e1 env
888                  in
889                    let c_e2, env = convert_to_c_expression e2 env
890                    in

```

```

872           let c_e3,env = convert_to_c_expression e3 env
873           in
874             Sast.Ternary(c_e1,c_e2,c_e3),env
875           | Ast.Array_Accessor(e,e_list,is_lvalue) ->
876             (* Check e is an array *)
877             (match infer_type e env with
878               | Ast.Array(t,n) -> ()
879               | _ -> raise
880             ← Exceptions.Not_an_array_expression);
881             (* Check that e_list can access a*)
882             ignore(List.map (fun x -> same_types (infer_type
883               x env) (Ast.Primitive(Ast.Integer))) e_list);
884             (* Convert *)
885             let c_e,env = convert_to_c_expression e env in
886             let c_e_list = List.map (fun x ->
887               fst(convert_to_c_expression x env)) e_list in
888             let array_type = infer_type e env in
889             let array_dims = get_array_dimensions array_type
890             []
891             in
892             let array_access = ((List.length array_dims) >
893               (List.length e_list)) in
894
895             ← Sast.Array_Accessor(c_e,c_e_list,is_lvalue,array_access),env
896             | Ast.Binop(e1,op,e2) ->
897               (* Check that expressions match *)
898               if((same_types (infer_type e1 env) (infer_type e2
899                 env)) = false) then raise (Exceptions.Type_mismatch
900                 "Binop doesn't match")
901               else
902                 (match op with
903                   | Ast.And | Ast.Or | Ast.Xor ->
904                     (* Check that type is boolean if using
905                       boolean operator *)
906                     ignore(same_types (infer_type e1 env)
907                           (Ast.Primitive(Ast.Boolean)));
908                     let c_e1,env = convert_to_c_expression e1
909                     env in
910                     let c_op,env = convert_to_c_binop op env
911                     in
912                     let c_e2,env = convert_to_c_expression e2
913                     env in
914                     Sast.Binop(c_e1,c_op,c_e2),env
915                     | _ ->
916                       (* Check if type is string, array *)
917                       (match (infer_type e1 env) with
918                         | Ast.Primitive(t) -> if t = Ast.String
919                           then raise
920                             (Exceptions.Cannot_perform_operation_on_string
921                               (Utils.binary_operator_to_string op)) else ()
922                           | Ast.Array(t,n) -> raise
923                             (Exceptions.Cannot_perform_operation_on_array
924                               (Utils.binary_operator_to_string op))
925                           );

```

```

906           let c_e1, env = convert_to_c_expression e1
907           → env in
908           → in
909           → env in
910           →   Sast.Binop(c_e1, c_op, c_e2), env
911           )
912           | Ast.Higher_Order_Function_Call(hof) ->
913             (* Check that function exists in environment *)
914             if (is_function_in_scope
915               (Utils.idtos(hof.kernel_function_name)) env) = false
916             then raise (Exceptions.Function_not_defined
917               (Utils.idtos hof.kernel_function_name));
918               (* Check that arrays are valid arrays *)
919               (* let input_arrays = List.map (fun e -> infer_type
920                 e env) hof.input_arrays in
921                 let good_arrays = (List.iter same_types_list
922                   input_arrays) in *)
923                 (* Check that function arguments match that of
924                   function declaration *)
925                 let f_info = (get_function_info (Utils.idtos
926                   hof.kernel_function_name) env) in
927                   (* if f_info.function_type != Kernel_Device then
928                     raise
929                     (Exceptions.Higher_order_function_call_only_takes_defg_functions)
930                     else *)
931                     let expected_arg_types = f_info.function_args in
932                     let get_array_types arr =
933                       match arr with
934                         | Ast.Array(t, n) -> t
935                         | _ -> raise
936           → Exceptions.Invalid_input_argument_to_map
937             in
938             let f_arg_types = List.map get_array_types
939               (get_types hof.input_arrays [] env) in
940               let check_args expected_arg_types f_args =
941                 List.map2 same_types expected_arg_types f_args in
942                   ignore(same_types (List.length f_arg_types))
943                   (List.length expected_arg_types));
944                   ignore(check_args f_arg_types expected_arg_types);
945                   (*Check that constants match those unknown
946                     variables in the defg*)
947                     let retrieve_constant_name c =
948                       match c with
949                         | Ast.Constant(id, e) -> Utils.idtos(id)
950                         in
951                           let hof_call_constants_names = List.map
952                             (retrieve_constant_name) hof.constants in
953                             let hof_constants_names = List.map (fun x ->
954                               Utils.idtos(x)) f_info.unknown_variables in
955                               let rec check_constants hof_call_c hof_fdecl_c =
956                                 match hof_fdecl_c with

```

```

941             | [] -> true
942             | hd::tl -> if (List.exists (fun s -> s = hd)
943             ← hof_call_c) = false then raise
944             ← Exceptions.Constants_missing_in_defg
945             else check_constants hof_call_c tl
946             in
947             ignore(check_constants hof_call_constants_names
948             ← hof_constants_names);
949             (match Utils.idtos(hof.hof_type) with
950             | "map" ->
951               (*Add the c map function to the
952               environment*)
953               let hof_c_fdecl = make_hof_c_fdecl hof env
954               in
955               let hof_ptx_fdecl = make_hof_ptx_fdecl
956               hof_c_fdecl hof env in
957               let env = update_hof_lists hof_c_fdecl
958               hof_ptx_fdecl env in
959               (* Convert *)
960
961             ← Sast.Function_Call(hof_c_fdecl.higher_order_function_name, (List.map
962             (fun x -> fst(convert_to_c_expression x env)) (List.rev
963             hof.input_arrays))), env
964             (* / "reduce" ->
965               in Sast.FunctionCall(c_ma) *)
966             | _ -> raise
967             ← (Exceptions.Unknown_higher_order_function_call
968             (Utils.idtos(hof.hof_type))) )
969
970             (* Stack Algorithm *)
971             (*
972               PUSH EXPRESSION STACK
973               RECURSE
974               SAVE LAST PTX ID
975               POP EXPRESSION STACK
976               PUSH LAST SAVED ONTO HIGHER STACK
977             *)
978             (* FILL IN WITH SEMANTIC CHECKING !!!!!!! *)
979
980             let rec bool_sum bool_list sum =
981               match bool_list with
982               | [] -> sum
983               | hd::tl ->
984                 let num = (if hd = true then 1 else 0) in
985                 bool_sum (tl) (sum + num)
986
987             let rec is_constant expr =
988               match expr with
989               | Ast.Integer_Literal(i) -> true
990               | Ast.Floating_Point_Literal(f) -> true
991               | Ast.Boolean_Literal(f) -> true (*Maybe want to
992               change*)

```

```

980      | Ast.Array_Literal(e_list) -> ((bool_sum (List.map
981        is_constant e_list) 0) = (List.length e_list))
982      | _ -> false
983
984  let rec convert_to_ptx_expression e env =
985    match e with
986      | Ast.String_Literal(s) -> raise
987        Exceptions.NO_STRINGS_ALLOWED_IN_GDECL;
988      | Ast.Higher_Order_Function_Call(hof) -> raise
989        Exceptions.No_Hof_Allowed
990      | Ast.Integer_Literal(i) ->
991        let env = update_expression_stack
992        (Sast.Ptx_Signed_Integer(i)) env in
993          Sast.Ptx_Block([Ptx_Empty]), env
994      | Ast.Boolean_Literal(b) ->
995        let env = update_expression_stack
996        (Sast.Ptx_Predicate(if b = true then 1 else 0)) env in
997          Sast.Ptx_Block([Ptx_Empty]), env
998      | Ast.Floating_Point_Literal(f) ->
999        let env = update_expression_stack
1000       (Sast.Ptx_Signed_Float f) env in
1001         Sast.Ptx_Block([Ptx_Empty]), env
1002      | Ast.Identifier_Literal(i) ->
1003        if(check_already_declared (Utils.idtos i) env) =
1004        false then raise
1005        (Exceptions.Variable_not_found_in_scope ( Utils.idtos
1006          i) )
1007        else
1008          let v_info = get_variable_info (Utils.idtos i)
1009          env in
1010            let is_array = match v_info.vtype with |
1011              | Ast.Primitive(p) -> false | Ast.Array(t,n) -> true in
1012                let ptx_lit =
1013                  Sast.Ptx_Identifier_Literal(make_ptx_id i (get_reg_type
1014                    (v_info.vtype)) v_info.register_number true is_array)
1015                in
1016                  let env = update_expression_stack ptx_lit env in
1017                    Sast.Ptx_Block([Ptx_Empty]), env
1018                | Ast.Binop(e1,o,e2) ->
1019                  if((same_types (infer_type e1 env) (infer_type e2
1020                    env)) = false) then raise (Exceptions.Type_mismatch
1021                    "Binop doesn't match")
1022                  else
1023                    (match o with
1024                      | Ast.And | Ast.Or | Ast.Xor ->
1025                        (* Check that type is boolean if using
1026                          boolean operator *)
1027                          ignore(same_types (infer_type e1 env)
1028                            (Ast.Primitive(Ast.Boolean)));
1029                          | _ ->
1030                            (* Check if type is string, array *)
1031                            (match (infer_type e1 env) with

```

```

1014 | Ast.Primitive(t) -> if t = Ast.String
1015   then raise
1016   (Exceptions.Cannot_perform_operation_on_string
1017   (Utils.binary_operator_to_string o)) else ()
1018   | Ast.Array(t,n) -> raise
1019   (Exceptions.Cannot_perform_operation_on_array
1020   (Utils.binary_operator_to_string o))
1021   );
1022   );
1023 let vtype = infer_type e1 env in
1024   (* Push stack *)
1025   let env = push_expression_stack env in
1026   let ptx_e1, env = convert_to_ptx_expression e1 env
1027   in
1028   let ptx_e2, env = convert_to_ptx_expression e2 env
1029   in
1030   (* We now have two values on our current stack,
1031   resolve *)
1032   (* For binop, we need to generate a third
1033   register to store value of addition*)
1034   let reg_name, reg_num = generate_reg vtype in
1035   let ptx_lit =
1036   Sast.Ptx_Identifier_Literal(make_ptx_id
1037   (Ast.Identifier(""))) reg_name reg_num true false) in
1038   let ptx_binop, env = convert_to_ptx_binop o
1039   env in
1040   let ptx_vtype, env =
1041   convert_to_ptx_variable_type vtype env in
1042   let resolve =
1043   Sast.Ptx_Binop(ptx_binop, ptx_vtype, ptx_lit, get_from_expression_stack
1044   1 env, get_from_expression_stack 0 env) in
1045   (* Pop stack *)
1046   let env = pop_expression_stack env in
1047   (* Push the ptx_lit on current stack *)
1048   let env = update_expression_stack ptx_lit env
1049   in
1050
1051   let ptx_expr_block = [ptx_e1; ptx_e2; resolve] in
1052   Sast.Ptx_Block(ptx_expr_block), env
1053 | Ast.Unop(e,o) ->
1054   (match o with
1055   | Ast.Not ->
1056     if ((infer_type e env) =
1057     Ast.Primitive(Ast.Boolean)) = false then raise
1058     (Exceptions.Type_mismatch("Must use boolean expression
1059     with boolean unop"))
1060     | _ -> ());
1061     if ((infer_type e env) = (Ast.Primitive
1062     (Ast.String))) then raise
1063     (Exceptions.Cannot_perform_operation_on_string
1064     (Utils.unary_operator_to_string o))
1065     else
1066     let vtype = infer_type e env in
1067     (* Push stack *)

```

```

1046           let env = push_expression_stack env in
1047           let ptx_e,env = convert_to_ptx_expression e env
1048           in
1049             (* We now have a value on our current stack,
1050                resolve *)
1051               (* Unop requires a generated second register
1052                *)
1053               let reg_name,reg_num = generate_reg vtype in
1054               let ptx_lit =
1055                 Sast.Ptx_Identifier_Literal(make_ptx_id (Ast.Identifier
1056                   "") reg_name reg_num true false) in
1057               let ptx_unop,env = convert_to_ptx_unop o env
1058               in
1059                 let ptx_vtype,env =
1060                   convert_to_ptx_variable_type vtype env in
1061                   let resolve =
1062                     Sast.Ptx_Unop(ptx_unop,ptx_vtype,ptx_lit,get_from_expression_stack
1063                       0 env) in
1064                     (* Pop stack *)
1065                     let env = pop_expression_stack env in
1066                     (* Push the ptx_lit on current stack *)
1067                     let env = update_expression_stack ptx_lit env
1068                     in
1069                       let ptx_expr_block = [ptx_e;resolve] in
1070                         Sast.Ptx_Block(ptx_expr_block),env
1071                         | Ast.Array_Literal(e_list) ->
1072                           (* Check all elements of the array are the same
1073                             type *)
1074                           let type_list = List.map (fun x -> infer_type x
1075                             env) e_list in
1076                             ignore(same_types_list type_list);
1077                             (* Get array dimensions and pass to sast *)
1078                             let arr = Ast.Array(infer_type (List.hd e_list)
1079                               env ,List.length e_list) in
1080                               let array_dim = get_array_dimensions arr [] in
1081                                 (* Check that all the expressions are primitives
1082                                   because PTX doesn't allow expressions *)
1083                                 let valid_array = (bool_sum (List.map is_constant
1084                                   e_list) 0) = List.length(e_list) in
1085                                   if (valid_array = false) then raise
1086                                     Exceptions.Defg_arrays_must_be_defined_with_constants
1087                                     else
1088                                       (* Now we know that the array list is only full
1089                                         of array lits, basically just convert all of them*)
1090                                         (* Push on stack *)
1091                                         let env = push_expression_stack env in
1092                                           let lit_list, env = convert_list
1093                                             convert_to_ptx_expression e_list [] env in
1094                                               (* For an array literal, we will push the
1095                                                 entire thing onto the stack *)
1096                                                 let rec get_elements stack alist = match
1097                                                   stack with [] -> alist| hd::tl -> get_elements tl
1098                                                     (hd::alist) in

```

```

1078      let array_lit = Sast.Ptx_Array_Literal
1079      ((get_elements(List.hd env.expression_stack) [])) in
1080      let env = update_expression_stack array_lit env
1081      in
1082      Sast.Ptx_Block([Ptx_Empty]), env
1083      | Ast.Function_Call(id, e_list) ->
1084          if (is_function_in_scope (Utils.idtos id) env) =
1085          false then (raise (Exceptions.Function_not_defined
1086          (Utils.idtos id)));
1087          (* Check that function arguments match that of
1088          function declaration *)
1089          let f_info = (get_function_info (Utils.idtos id)
1090          env) in
1091          let f_arg_types = f_info.function_args in
1092          let check_args expected_arg_types f_args =
1093              List.map2 same_types expected_arg_types f_args in
1094              ignore(check_args f_arg_types (get_types
1095              e_list [] env));
1096          let rtype = f_info.function_return_type in
1097          (* Push stack *)
1098          let env = push_expression_stack env in
1099          let lit_list, env = convert_list
1100          convert_to_ptx_expression e_list [] env in
1101          (* For a function call, need to define a return
1102          register *)
1103          let reg_name, reg_num = generate_reg rtype in
1104          let ptx_lit =
1105          Sast.Ptx_Identifier_Literal(make_ptx_id (Ast.Identifier
1106          "") reg_name reg_num true false) in
1107          let rec get_elements stack alist = match stack
1108          with [] -> alist | hd::tl -> get_elements tl (hd::alist)
1109          in
1110          let expr = match rtype with
1111              | Ast.Primitive(Ast.Void) ->
1112                  Sast.Ptx_Empty_Call(id, (get_elements(List.hd
1113                  env.expression_stack) []))
1114                  | _ -> Sast.Ptx_Call(ptx_lit, id,
1115                  (get_elements(List.hd env.expression_stack) []))
1116                  in
1117                  let env = pop_expression_stack env in
1118                  let env = update_expression_stack ptx_lit env in
1119                  expr, env
1120                  (* Pop stack *)
1121                  (* For function call, we resolve the expressions
1122                  and then *)
1123                  | Ast.Cast(vtype, e) -> raise
1124                      Exceptions.Casting_not_allowed_in_defg
1125                  | Ast.Array_Accessor(e, e_list, b) -> raise
1126                      Exceptions.C'est_La_Vie
1127                  | Ast.Ternary(e1, e2, e3) -> raise
1128                      Exceptions.C'est_La_Vie

```

```

1108      (* if(same_types (infer_type e1 env) (infer_type e3
1109      ← env)) = false then raise
1110      ← (Exceptions.Type_mismatch("Ternary expression don't
1111      ← match")))
1112      else
1113          Check e2 is boolean
1114          if(same_types (infer_type e2 env)
1115          ← (Ast.Primitive(Ast.Boolean))) = false then (raise
1116          ← (Exceptions.Conditional_must_be_a_boolean))
1117          else *)
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
```

```

(* if(same_types (infer_type e1 env) (infer_type e3
env)) = false then raise
(Exceptions.Type_mismatch("Ternary expression don't
match"))

else
    Check e2 is boolean
    if(same_types (infer_type e2 env)
(Ast.Primitive(Ast.Boolean))) = false then (raise
(Exceptions.Conditional_must_be_a_boolean))
    else *)

let rec get_array_el_type arr num_dim =
  match num_dim with
  | 1 ->
    (match arr with
    | Ast.Array(t,n) -> t
    | _ -> raise Exceptions.Not_an_array_expression
    )
  | _ ->
    if num_dim <= 0 then raise
    Exceptions.Invalid_accessor_value
    else
      (match arr with
      | Ast.Array(t,n) -> get_array_el_type t
      | (num_dim-1)
        | _ -> raise Exceptions.Not_an_array_expression
      )

let convert_to_c_variable_statement vstmt env =
  match vstmt with
  | Ast.Declaration(vdecl) -> (* Check that it isn't
already declared in convert_to_c_vdecl *)
    let c_vdecl, new_env = convert_to_c_vdecl vdecl
    env in
      Sast.Declaration(c_vdecl), new_env
  | Ast.Initialization(vdecl,e) ->
    (*Check same types*)
    let vtype = match vdecl with
    | Ast.Variable_Declaration(v,id) -> v
    in
      ignore(same_types (vtype) (infer_type e env));
      (* Convert - note vdecl also checks if
declared *)
    let c_vdecl, env = convert_to_c_vdecl vdecl env
    in
      let c_e, env = convert_to_c_expression e env in
      Sast.Initialization(c_vdecl,c_e),env
  | Ast.Assignment(e1,e2) ->
    (* Check that identifiers are declared *)
    match e1 with
```

```

1150           | Ast.Identifier_Literal(id) ->
1151               if (check_already_declared
1152                   (Utils.idtos(id)) env) = false then raise
1153                   (Exceptions.Name_not_found (Utils.idtos id)))
1154               else
1155                   (* Check same types*)
1156                   ignore(same_types (get_variable_type
1157                       (Utils.idtos id) env) (infer_type e2 env));
1158                   (*Convert*)
1159                   let c_e1, env = convert_to_c_expression
1160                   e1 env in
1161                   let c_e2, env = convert_to_c_expression
1162                   e2 env in
1163                   Sast.Assignment(c_e1,c_e2),env
1164           | Ast.Array_Accessor(e,e_list,is_lvalue)->
1165               (match e with
1166                   | Ast.Identifier_Literal(id) ->
1167                       if (check_already_declared
1168                           (Utils.idtos id) env) = false then raise
1169                           (Exceptions.Name_not_found (Utils.idtos id)))
1170                       else
1171                           (* Check same types*)
1172                           let arr = get_variable_type
1173                           (Utils.idtos id) env in ignore(same_types
1174                               (get_array_el_type arr (List.length e_list))
1175                               (infer_type e2 env));
1176                               (*Convert*)
1177                               let c_e1, env =
1178                                   convert_to_c_expression e1 env in
1179                               let c_e2, env =
1180                                   convert_to_c_expression e2 env in
1181                               Sast.Assignment(c_e1,c_e2),env
1182                               | _ -> (raise
1183                                   (Exceptions.Cannot_assign_expression)
1184                                   )
1185                               | _ -> raise
1186                               (Exceptions.Cannot_assign_expression)
1187
1188           (* TO IMPLEMENT *)
1189           (*Stack algorithm for conversion:
1190              Push stack
1191              Recurse on expression, Every expression will push a new
1192              stack, and when resolved will pop its stack
1193              Save last ptx_id we obtain from resolving -> this will be
1194              different for different expressions -> this is done in
1195              expressions
1196              Pop stack
1197
1198              Push last ptx_id onto new stack
1199              Return Sast.datatype, new env
1200          *)
1201
1202           let get_vdecl_parts vdecl =

```

```

1186     (match vdecl with
1187     | Ast.Variable_Declaration(t,i) -> i,t)
1188
1189 let convert_to_ptx_variable_statement vstmt env =
1190   match vstmt with
1191   | Ast.Declaration(vdecl) ->
1192     let ptx_vdecl,env = convert_to_ptx_vdecl vdecl
1193     env in
1194   Sast.Ptx_Variable_Declaration(ptx_vdecl),env
1195   | Ast.Initialization(vdecl, e) ->
1196     let ptx_vdecl,env = convert_to_ptx_vdecl vdecl
1197     env in
1198     (* Push scope for expression stack *)
1199     let env = push_expression_stack env in
1200     let vdecl_expr =
1201       Sast.Ptx_Variable_Declaration(ptx_vdecl) in
1202       (* Must save ptx value for vdecl on the stack
1203       *)
1204       let id,vtype = get_vdecl_parts vdecl in
1205       let v_info = get_variable_info
1206       (Utils.idtos id) env in
1207       let ptx_lit =
1208       Sast.Ptx_Identifier_Literal(make_ptx_id id
1209       (get_reg_type v_info.vtype) v_info.register_number true
1210       false) in
1211       let env = update_expression_stack
1212       ptx_lit env in
1213       let ptx_e,env = convert_to_ptx_expression e
1214       env in
1215       (* convert_to_ptx_expression has saved a
1216       value in the stack. Let us fetch it and resolve*)
1217       let resolve =
1218       Sast.Ptx_Move(fst(convert_to_ptx_variable_type vtype
1219       env),get_from_expression_stack 1 env,
1220       get_from_expression_stack 0 env) in
1221       (* Pop the stack *)
1222       let env = pop_expression_stack env in
1223       let expr_block =
1224       [vdecl_expr;ptx_e;resolve] in
1225       Sast.Ptx_Block(expr_block),env
1226   | Ast.Assignment(e1, e2) ->
1227     match e1 with
1228     | Ast.Identifier_Literal(id) ->
1229       (* Ast checking...*)
1230       if (check_already_declared
1231       (Utils.idtos(id)) env) = false then raise
1232       (Exceptions.Name_not_found (Utils.idtos id))
1233       else
1234         ignore(same_types (get_variable_type
1235       (Utils.idtos id) env) (infer_type e2 env));
1236         let env = push_expression_stack env in
1237         (* Must save ptx value for vdecl on the stack
1238         *)

```

```

1220           let v_info = get_variable_info
1221   ↳ (Utils.idtos id) env in
1222     (* Update vmap *)
1223     let ptx_id = make_ptx_id id
1224     (get_reg_type v_info.vtype) v_info.register_number true
1225   ↳ false in
1226     let env = update_variable_register
1227   ↳ (Utils.idtos id) ptx_id.reg_num env in
1228     let ptx_lit =
1229       Sast.Ptx_Identifier_Literal(ptx_id) in
1230     let env = update_expression_stack
1231     ptx_lit env in
1232     let ptx_e,env = convert_to_ptx_expression e2
1233   ↳ env in
1234     (* convert_to_ptx_expression has saved a
1235      value in the stack. Let us fetch it and resolve*)
1236     let resolve =
1237       Sast.Ptx_Move(fst(convert_to_ptx_variable_type
1238       v_info.vtype env),get_from_expression_stack 1 env,
1239       get_from_expression_stack 0 env) in
1240     (* Pop the stack *)
1241     let env = pop_expression_stack env in
1242     let expr_block = [ptx_e;resolve] in
1243       Sast.Ptx_Block(expr_block),env
1244     | Ast.Array_Accessor(e,e_list,is_lvalue)->
1245       raise Exceptions.C'est_La_Vie
1246         (* match e with
1247           | Ast.Identifier_Literal(id) ->
1248             if (check_already_declared
1249               (Utils.idtos id) env )= false then raise
1250               (Exceptions.Name_not_found (Utils.idtos id))
1251             else
1252               (* Check same types*)
1253               let arr = get_variable_type
1254                 (Utils.idtos id) env in ignore(same_types
1255                 (get_array_el_type arr (List.length e_list)))
1256                 (infer_type e2 env));
1257
1258               (* This case is weird because
1259               we know e is an identifier literal, and that it is an
1260               array, so we can gets its information to make a ptx_id
1261               from get_variable_info *)
1262               (* Don't need to push pop -
1263               special case for assign *)
1264               (* We get the variable
1265               information for the array *)
1266               let v_info = get_variable_info
1267               id env in
1268             (* NEED TO RESOLVE*)      let arr_ptx_id = make_ptx_id id
1269             (get_reg_type v_info.vtype) v_info.register_number true
1270             in

```

```

1248                                We need to create a load
1249
1250                                let reg_name, reg_num =
1251                                generate_reg vtype in let new_v_info = { vtype =
1252                                v_info.vtype; register_number = reg_num;} in
1253                                let new_vmap = Variable_Map.add
1254                                (Utils.idtos id) new_v_info (List.hd
1255                                env.variable_scope_stack) in
1256                                let env = update_scope new_vmap
1257                                env in
1258                                let ptx_e = make_ptx_id id
1259                                reg_name reg_num true in
1260                                (*Push expression stack*)
1261                                let env = push_expression_stack
1262                                env in
1263                                let e1_stmt_block,env =
1264                                convert_to_c_expression e1 env in
1265                                let e2_stmt_block,env =
1266                                convert_to_c_expression e2 env in
1267                                Sast.Load(Sast.Global,c_e1,c_e2),env
1268                                (*Pop expression stack*)
1269                                let env = pop_expression_stack
1270                                env in
1271                                Sast.Block(),env
1272                                | _ -> (raise
1273                                Exceptions.Cannot_assign_expression)
1274                                | _ -> raise
1275                                Exceptions.Cannot_assign_expression
1276
1277
1278
1279                                (* Converts global vstmt list into c vstmt list *)
1280                                let rec convert_to_c_variable_statement_list vstmt_list
1281                                c_vstmt_list env =
1282                                match vstmt_list with
1283                                | [] -> (c_vstmt_list,env)
1284                                | hd::tl ->
1285                                let c_vstmt, env =
1286                                convert_to_c_variable_statement hd env in
1287                                convert_to_c_variable_statement_list tl
1288                                (List.rev(c_vstmt)::List.rev(c_vstmt_list))) env
1289
1290
1291
1292                                let rec convert_to_c_statement stmt env =
1293                                match stmt with
1294                                | Ast.Variable_Statement(vstmt) ->
1295                                let c_vstmt,env = convert_to_c_variable_statement
1296                                vstmt env in
1297                                Sast.Variable_Statement(c_vstmt),env
1298                                | Ast.Expression(e) ->
1299                                let c_e,env = convert_to_c_expression e env in

```

```

1283     Sast.Expression(c_e), env
1284     | Ast.If(e,stmt1,stmt2) ->
1285         (* Check that e is a boolean expression *)
1286         ignore(same_types (infer_type e env)
1287         ← (Ast.Primitive(Ast.Boolean)));
1288         (* Convert *)
1289         let c_e, env = convert_to_c_expression e env in
1290         let c_stmt1,env = convert_to_c_statement stmt1 env
1291         ← in
1292         let c_stmt2,env = convert_to_c_statement stmt2 env
1293         ← in
1294         Sast.If(c_e,c_stmt1,c_stmt2), env
1295     | Ast.While(e,stmt) ->
1296         ignore(same_types (infer_type e env)
1297         ← (Ast.Primitive(Ast.Boolean)));
1298         (* Check that e is a boolean expression *)
1299         let c_e, env = convert_to_c_expression e env in
1300         let c_stmt,env = convert_to_c_statement stmt env in
1301         Sast.While(c_e,c_stmt), env
1302     | Ast.For(stmt1,e,stmt2,stmt3) ->
1303         (* Check that stmt1 is an initialization expression
1304         ← *)
1305         (match stmt1 with
1306             | Ast.Variable_Statement(vstmt) ->
1307                 (match vstmt with
1308                     | Ast.Assignment(e1,e2) -> ()
1309                     | Ast.Initialization(vdecl,e) -> ()
1310                     | _ -> raise
1311             ← Exceptions.Invalid_statement_in_for)
1312             | _ -> raise
1313             ← Exceptions.Invalid_statement_in_for);

1314             (* Convert *)
1315             let env = push_scope env in
1316             let c_stmt1, env = convert_to_c_statement stmt1
1317             ← env in
1318             (* Check that e is a boolean expression *)
1319             ignore(same_types (infer_type e env)
1320             ← (Ast.Primitive(Ast.Boolean)));
1321             let c_e, env = convert_to_c_expression e
1322             ← env in
1323             let c_stmt2, env = convert_to_c_statement stmt2
1324             ← env in
1325             let c_stmt3, env = convert_to_c_statement stmt3
1326             ← env in
1327             let env = pop_scope env in
1328             Sast.For(c_stmt1,c_e,c_stmt2,c_stmt3), env
1329     | Ast.Return(e) ->
1330         let c_e, env = convert_to_c_expression e env in
1331         Sast.Return(c_e), env
1332     | Ast.Return_Void -> Sast.Return_Void, env
1333     | Ast.Continue -> Sast.Continue, env
1334     | Ast.Break -> Sast.Break, env

```

```

1324     | Ast.Block(stmt_list) ->
1325         (* Check that nothing follows a return , break, or
1326        continue in a block *)
1327         if (good_statement_order stmt_list) = false then
1328             raise
1329             Exceptions.Have_statements_after_return_break_continue
1330             else
1331                 (* Convert *)
1332                 let c_stmt_list,env = convert_list
1333                 convert_to_c_statement stmt_list [] env in
1334                 Sast.Block(c_stmt_list),env
1335
1336 let rec convert_to_ptx_statement stmt env =
1337     match stmt with
1338     | Ast.Variable_Statement(v) ->
1339         convert_to_ptx_variable_statement v env
1340     | Ast.Expression(e) -> convert_to_ptx_expression e env
1341     | Ast.Return_Void -> Sast.Ptx_Return_Void, env
1342     | Ast.Return(e) ->
1343         let vtype = infer_type e env in
1344         let env = push_expression_stack env in
1345         let ptx_e, env = convert_to_ptx_expression e env
1346         in
1347             let rlit = env.return_lit in
1348             let expr =
1349                 Sast.Ptx_Store(Sast.Global,fst(convert_to_ptx_variable_type
1350 vtype env),rlit,get_from_expression_stack 0 env) in
1351                 let expr_block = [expr;Sast.Ptx_Return_Void] in
1352                 let env = pop_expression_stack env in
1353                 Sast.Ptx_Block(expr_block),env
1354
1355     | Ast.Block(stmt_list) ->
1356         let expr_block, env = convert_list
1357         convert_to_ptx_statement stmt_list [] env in
1358             Sast.Ptx_Block(expr_block),env
1359     | Ast.If(e, s1, s2) -> raise Exceptions.C'est_La_Vie
1360     (*
1361         let env = push_expression_stack env in
1362         let vtype = infer_type e env in
1363         let ptx_e,env = convert_to_ptx_expression e
1364         env in
1365             (* Create a literal referencing the
1366            predicate *)
1367             let ptx_lit =
1368                 Sast.Ptx_Identifier_Literal(make_ptx_id (get_reg_type
1369 vtype) (get_predicate_counter ()) true false) in
1370                 let env = update_expression_stack ptx_lit env
1371                 in
1372                     let bool_expr = Sast.Block([ptx_e]) in
1373                         let env = pop_expression_stack env in
1374                             let branch =
1375                                 Sast.Branch(get_from_expression_stack 0
1376 env,generate_subroutine_name()) in
1377                                 let

```

```

1361           Sast.Ptx_Block(expr_block), env *)
1362   | Ast.While(e, s) -> raise Exceptions.C'est_La_Vie
1363   | Ast.For(s1, e, s2, s3) -> raise
1364   ↵ Exceptions.C'est_La_Vie
1365   | Ast.Continue -> raise Exceptions.C'est_La_Vie
1366   | Ast.Break -> raise Exceptions.C'est_La_Vie
1367
1368
1369 let convert_to_c_param vdecl env =
1370   match vdecl with
1371     | Ast.Variable_Declaration(vtype, id) ->
1372       if (check_already_declared (Utils.idtos id) env) =
1373         true then raise (raise
1374         (Exceptions.Variable_already_declared
1375          (Utils.idtos (id)) ))
1376       else
1377         let v_info = {
1378           vtype = vtype;
1379           register_number = 0;
1380         }
1381         in
1382         let updated_scope = Variable_Map.add
1383           (Utils.idtos id) v_info (List.hd
1384             env.variable_scope_stack) in
1385           let env = update_scope updated_scope env in
1386             let c_vtype, env = convert_to_c_variable_type
1387               vtype env in
1388               Sast.Variable_Declaration(c_vtype, id), env
1389
1390
1391 let rec check_rtype rtype body env =
1392   match body with
1393     | [] -> ()
1394     | hd::tl->
1395       match hd with
1396         | Ast.Return_Void -> if (rtype != Ast.Primitive(Ast.Void)) then raise
1397           Exceptions.Missing_return_type
1398           else check_rtype rtype tl env
1399         | Ast.Return(e) ->
1400           if (same_types (infer_type e
1401             env) rtype) = false then raise
1402             Exceptions.Return_type_doesn't_match
1403             else check_rtype rtype tl env
1404           | _ -> check_rtype rtype tl env
1405
1406 (* Converts from fdecl to c_fdecl *)
1407 let convert_to_c_fdecl fdecl env =
1408   if (is_function_in_scope (Utils.idtos fdecl.name) env)
1409     = true then (raise
1410      Exceptions.Function_already_declared)
1411    else
1412      let vdecl_to_param vdecl =

```

```

1401     match vdecl with
1402         | Ast.Variable_Declaration(vtype, id) -> vtype
1403     in
1404         (* Add to function map *)
1405         let host_func_info = {
1406             function_type = Host;
1407             function_name = fdecl.name;
1408             function_return_type = fdecl.return_type;
1409             function_args = List.map vdecl_to_param
1410             ← fdecl.params;
1411             dependent_functions = [];
1412             unknown_variables = [];
1413         }
1414         in
1415         let env = update_host_fmap host_func_info env in
1416             (* Push new scope for function *)
1417
1418         let env = push_scope env in
1419             (* Do conversion while passing environment *)
1420             let return_type, env = convert_to_c_variable_type
1421             ← fdecl.return_type env in
1422             let params, env = convert_list
1423             ← convert_to_c_param (List.rev fdecl.params) []
1424             ← env in
1425             let body, env = convert_list
1426             ← convert_to_c_statement fdecl.body [] env in
1427             let c_fdecl = {
1428                 c_fdecl_return_type = return_type;
1429                 c_fdecl_name = fdecl.name;
1430                 c_fdecl_params = params;
1431                 c_fdecl_body = body;
1432             }
1433             in
1434             check_rtype fdecl.return_type fdecl.body env;
1435             (* Pop the variable scope for the function *)
1436             let env = pop_scope env in
1437             c_fdecl, env)
1438
1439
1440 let convert_rtype_to_ptx_vdecl rtype env =
1441     let rname = Ast.Identifier( generate_ptx_return_name () )
1442     in
1443     let reg_name, reg_num = generate_reg rtype in
1444     let is_array = match rtype with | Ast.Primitive(p) ->
1445         false | Ast.Array(t, n) -> true in
1446     let ptx_id = make_ptx_id rname reg_name reg_num false
1447     ← false in
1448     let env = update_return_lit
1449     ← (Sast.Ptx_Identifier_Literal(ptx_id)) env in
1450     (Sast.Ptx_Vdecl(Sast.Param, fst
1451     ← (convert_to_ptx_variable_type rtype env), ptx_id)), env
1452
1453

```

```

1444
1445 let convert_to_ptx_fdecl fdecl env =
1446   if (is_function_in_scope (Utils.idtos fdecl.name) env)
1447   ↵ = true then (raise
1448   ↵ Exceptions.Function_already_declared)
1449 else
1450   let vdecl_to_param vdecl =
1451     match vdecl with
1452       | Ast.Variable_Declaration(vtype, id) -> vtype
1453     in
1454       (* Add to function map*)
1455     let kernel_func_info = {
1456       function_type = Kernel_Device;
1457       function_name = fdecl.name;
1458       function_return_type = fdecl.return_type;
1459       function_args = List.map vdecl_to_param
1460     fdecl.params;
1461     dependent_functions = [];
1462     unknown_variables = [];
1463   }
1464   in
1465   let env = update_kernel_fmap kernel_func_info env in
1466     (* Push new scope for function *)
1467   let env = push_scope env in
1468     (* Convert sections of the function *)
1469   let return_type, env      =
1470     convert_to_ptx_variable_type fdecl.return_type env in
1471   let params, env          = convert_list
1472   convert_to_ptx_param (List.rev fdecl.params) [] env in
1473   let output, env          = convert_rtype_to_ptx_vdecl fdecl.return_type env in
1474   let body,    env          = convert_list
1475   convert_to_ptx_statement fdecl.body  [] env in
1476   let registers, env       =
1477     [ convert_to_register_declaration (S32) "si"
1478     !signed_int_counter;
1479     convert_to_register_declaration (F32) "fl"
1480     !signed_float_counter;
1481     convert_to_register_declaration (Pred) "pr"
1482     !predicate_counter;
1483   ], env in
1484   check_rtype fdecl.return_type fdecl.body env;
1485   (* Create function item *)
1486   let ptx_fdecl = {
1487     ptx_fdecl_type = Sast.Device_Function;
1488     ptx_fdecl_name = fdecl.name;
1489     ptx_fdecl_input_params = params;
1490     ptx_fdecl_return_param = output;
1491     register_decls = registers;
1492     ptx_fdecl_body = body;
1493   }
1494   in
1495   (* Pop the variable scope for the function *)
1496   let env = pop_scope env in

```

```

1488     ptx_fdecl, env)
1489
1490 (* Converts a list of function declarations to ptx and c
1491    functions *)
1491 let rec convert_fdecl_list fdecl_list ptx_fdecl_list
1492   ~> c_fdecl_list env =
1492   match fdecl_list with
1493   | [] -> (ptx_fdecl_list, c_fdecl_list, env)
1494   | hd::tl ->
1495     ( match hd.is_kernel_function with
1496       | false ->
1497         let c_fdecl, env = convert_to_c_fdecl hd env
1498         ~> in
1498         convert_fdecl_list tl ptx_fdecl_list
1499         ~> (List.rev(c_fdecl)::List.rev(c_fdecl_list))) env
1499       | true ->
1500         let ptx_fdecl, env = convert_to_ptx_fdecl hd
1501         ~> env in
1501         convert_fdecl_list tl
1502         ~> (List.rev(ptx_fdecl)::List.rev(ptx_fdecl_list)))
1502         c_fdecl_list env
1502       )
1503
1504 (* Main function for converting ast to sast *)
1505 let convert ast env =
1506   let vstmt_list,env = convert_list
1506   ~> convert_to_c_variable_statement (fst(ast)) [] env in
1507   let ptx_fdecl_list,c_fdecl_list, env = convert_fdecl_list (snd(ast)) [] [] env in
1508   let sast =
1508   ~> (vstmt_list,ptx_fdecl_list,(env.hof_ptx_function_list),(env.hof_c_function_list))
1509   ~> in
1509   sast
1510
1511 (* Main function for Sast *)
1512 let analyze ast =
1513   let env = init_env in
1514   let sast = convert ast env in
1515   sast

```

codegen_c.ml

```
1  open Sast
2  (* open Exceptions *)
3  (* open Codegen_ptx *)
4
5  (* For sprintf *)
6  open Printf
7
8  (*-----Generating
   Functions-----*)
9
10 (* Calls generate_func for every element of the list and
    concatenates results with specified concat symbol
   Used if you need to generate a list of something - e.x.
   list of statements, list of params *)
11
12 let generate_list generate_func concat mylist =
13   let list_string = String.concat concat (List.map
14     generate_func mylist) in
15   sprintf "%s" list_string
16
17 (* Generate operators *)
18 let generate_binary_operator operator =
19   let op = match operator with
20     | Add -> "+"
21     | Subtract -> "-"
22     | Multiply -> "*"
23     | Divide -> "/"
24     | Modulo -> "%"
25     | And -> "&&"
26     | Or -> "||"
27     | Xor -> "^"
28     | Equal -> "=="
29     | Not_Equal -> "!="
30     | Greater_Than -> ">"
31     | Less_Than -> "<"
32     | Greater_Than_Equal -> ">="
33     | Less_Than_Equal -> "<="
34     | Bitshift_Right -> ">>"
35     | Bitshift_Left -> "<<"
36   in
37   sprintf "%s" op
38
39 let generate_unary_operator operator =
40   let op = match operator with
41     | Not -> "!"
42     | Negate -> "-"
43     | Plus_Plus -> "++"
44     | Minus_Minus -> "--"
45   in sprintf "%s" op
```

```

46 (* Generate data type*)
47 let generate_data_type dtype =
48   let data_type = match dtype with
49     | String -> "char *"
50     | Unsigned_Byte -> "unsigned char"
51     | Byte -> "signed char"
52     | Unsigned_Integer -> "unsigned int"
53     | Integer -> "int"
54     | Unsigned_Long -> "unsigned long"
55     | Long -> "long"
56     | Float -> "float"
57     | Double -> "double"
58     | Boolean -> "bool"
59     | Void -> "void"
60   in sprintf "%s" data_type
61
62 (* Generate variable type *)
63 let rec generate_variable_type variable_type =
64   let vtype = match variable_type with
65     | Primitive(p) -> generate_data_type p
66     | Array(t,n) ->
67       (match t with
68         | Array(t1,n1) -> generate_variable_type t1
69         | Primitive(p) -> generate_data_type p)
70   in sprintf "%s" vtype
71
72 (* Generate id *)
73 let generate_id id =
74   let id_string = Utils.idtos(id) in sprintf "%s" id_string
75 (* match id_string with
76   | "print" -> sprintf "printf"
77   | _ as identifier -> sprintf identifier *)
78
79 (* Generates CUDA device pointer *)
80 let generate_device_ptr ptr_name =
81   sprintf "CUdeviceptr %^ ptr_name ^ ";"
```

```

91     String.concat "\n" (List.map2
92     generate_mem_alloc_statement_host_to_device
93     fcalls.input_arrays_info (create_list [] (List.length
94     fcalls.input_arrays_info) fcalls.array_length)) in
95     sprintf "%s" mem_alloc_string
96
97 (* Generates CUDA copying from host to device*)
98 let generate_mem_cpy_statement_host_to_device arr_info
99     arr_length =
100    let mem_cpy_string =
101        "checkCudaErrors(cuMemcpyHtoD(^
102        Utils.idtos(arr_info.kernel_name) ^", " ^
103        Utils.idtos(arr_info.host_name) ^ ", sizeof(^
104        (generate_variable_type arr_info.variable_type) ^ ") *"
105        ^ string_of_int arr_length ^ "));\n" in
106    sprintf "%s" mem_cpy_string
107
108 let generate_mem_cpy_host_to_device fcalls =
109     let rec create_list mylist length element = if length >
110         0 then create_list (element::mylist) (length-1) element
111     else mylist in
112     let mem_cpy_string = String.concat "\n" (List.map2
113     generate_mem_cpy_statement_host_to_device
114     fcalls.input_arrays_info (create_list [] (List.length
115     fcalls.input_arrays_info) fcalls.array_length)) in
116     sprintf "%s" mem_cpy_string
117
118 (* Generates CUDA statement for kernel params *)
119 let generate_kernel_params arr_info =
120     let rec get_kernel_names a_info_list name_list =
121         match a_info_list with
122         | [] -> name_list
123         | hd::tl -> get_kernel_names tl
124         (hd.kernel_name)::name_list)
125     in
126     let kernel_names = (get_kernel_names arr_info []) in
127     let kernel_param_string = generate_list generate_id ", "
128     &" kernel_names in
129     sprintf "void *KernelParams[] = { &" ^
130     kernel_param_string ^ "};"
131
132 (* Generate CUDA memory cleanup *)
133 let generate_mem_cleanup arr_info =
134     sprintf "checkCudaErrors(cuMemFree(^
135     Utils.idtos(arr_info.kernel_name) ^ "));"
136
137 (* Generates variable declaration statements *)
138 let generate_vdecl d =
139     match d with
140     | Variable_Declaration(vtype, id) ->
141         match vtype with
142         | Array(t, n) -> sprintf "vlcarray fill"
143             (* Fill in with VLC_Array*)

```

```

127          (* let array_dimensions= (get_array_dimensions
128    → t [n]) in
129      Environment.combine [
130          Generator(generate_variable_type t);
131          Verbatim(" ");
132          Generator(generate_id d.name);
133          (* Get the array dimensions *)
134          Verbatim("[");
135          Verbatim(String.concat "][]" (List.map
136            string_of_int array_dimensions));
137          Verbatim("]")
138          ]
139          | Primitive(p) ->
140              let param_string = (generate_data_type p) ^ " "
141              ^ (generate_id id) in
142                  sprintf "%s" param_string
143          (*           / _ -> raise Exceptions.Unknown_variable_type
144          / _ -> raise Exceptions.Unknown_type_of_vdecl *)
145
146 let generate_param d =
147   match d with
148     | Variable_Declaration(vtype,id) ->
149       match vtype with
150         | Array(t,n) -> sprintf "vlcarray fill"
151             (* Fill in with VLC_Array*)
152             (* let array_dimensions= (get_array_dimensions
153               → t [n]) in
154               Environment.combine [
155                   Generator(generate_variable_type t);
156                   Verbatim(" ");
157                   Generator(generate_id d.name);
158                   (* Get the array dimensions *)
159                   Verbatim("[");
160                   Verbatim(String.concat "][]" (List.map
161                     string_of_int array_dimensions));
162                     Verbatim("]")
163                     ]
164                     | Primitive(p) ->
165                         let param_string = (generate_data_type p) ^ " "
166                         ^ (generate_id id) in
167                             sprintf "%s" param_string
168                         (*           / _ -> raise Exceptions.Unknown_variable_type
169                         / _ -> raise Exceptions.Unknown_type_of_param *)
170
171 (* Generate expressions - including higher order function
172   → calls - and constants *)
173 let rec generate_expression expression =
174   let expr = match expression with
175     | Function_Call(id, expr_list) ->
176         (generate_id id) ^ "(" ^ generate_list
177         → generate_expression "," expr_list ^ ")"
178     | Higher_Order_Function_Call(fcall) ->
179         generate_higher_order_function_call fcall

```

```

171      | Kernel_Function_Call(kfcall) ->
172      generate_kernel_function_call kfcall
173      | String_Literal(s) ->
174          "\"^ s ^ \""
175      | Integer_Literal(i) ->
176          string_of_int i
177      | Boolean_Literal(b) ->
178          string_of_bool b
179      | Floating_Point_Literal(f) ->
180          string_of_float f
181      | Array_Literal(s) ->
182          "vlcarray fill"
183          (* Fill in with VLC_Array*)
184          (* sprintf "{" ^ (generate_expression_list s) ^ "}" *)
185          *)
186      | Identifier_Literal(id) ->
187          (generate_id id)
188      | Cast(vtype,e) ->
189          "(" ^ (generate_variable_type vtype) ^ ")" ^ "
190          (generate_expression e)
191      | Binop(e1, o, e2) ->
192          (generate_expression e1) ^ " " ^ "
193          (generate_binary_operator o) ^ " " ^
194          (generate_expression e2)
195      | Unop(e,o) ->
196          (match o with
197              | Not | Negate -> (generate_unary_operator o) ^
198              (generate_expression e)
199              | Plus_Plus | Minus_Minus -> (generate_expression
200                  e) ^ (generate_unary_operator o))
201      | Array_Accessor(e,e_list) -> (generate_expression e) ^
202          "[" ^ (generate_list generate_expression "] [" e_list) ^
203          "]"
204      | Ternary(e1,e2,e3) -> "(" ^ (generate_expression e2) ^
205          ") ? " ^ (generate_expression e1) ^ ":" ^
206          (generate_expression e3)
207      in sprintf "%s" expr
208      (* Generates CUDA statements that copy constants from host
209         to gpu *)
210      and generate_constant_on_gpu const =
211          let mem_alloc_constant_string = match const.variable_type
212          with
213              | Primitive(vtype) ->
214                  generate_device_ptr
215                  (Utils.idtos(const.kernel_name)) ^
216                      generate_mem_alloc_statement_host_to_device const
217                  1 ^
218                      generate_mem_cpy_statement_host_to_device const 1
219              | Array(vtype,length) ->
220                  "vlcarray fill"
221                  (* / _ -> raise Exceptions.Unknown_variable_type *)
222                  in
223                      sprintf "%s" mem_alloc_constant_string

```

```

209 and generate_kernel_function_call kfcall = sprintf "hi" (*
210   → Why do we need semicolon?????*)
211   (* Fill in with VLC_Array *)
212   (* Generates statements for higher order map or reduce
213     → calls *)
214 and generate_higher_order_function_call fcall =
215   let higher_order_function_call_string =
216     match Utils.idtos(fcall.higher_order_function_type)
217   → with
218     | "map" ->
219       (* Fill in with VLC_Array *)
220       "{0};\n" ^
221       (* Initializes CUDA driver and loads needed function *)
222       "checkCudaErrors(cuCtxCreate(&context, 0,
223         device));\n" ^
224       "std::ifstream t(\"" ^ Utils.idtos
225       fcall.applied_kernel_function ^ ".ptx\");\n" ^
226       "if (!t.is_open()) {\n" ^
227         " std::cerr << \"\" ^ Utils.idtos
228       fcall.applied_kernel_function ^ ".ptx not found\n\";\n"
229       ^
230         "return 1;\n" ^
231       "}\n" ^
232       "std::string " ^ Utils.idtos
233       fcall.applied_kernel_function ^ "_str" ^
234       "((std::istreambuf_iterator<char>(t)),
235       std::istreambuf_iterator<char>());\n" ^
236       "checkCudaErrors(cuModuleLoadDataEx(&cudaModule, "
237       (Utils.idtos fcall.applied_kernel_function) ^ "_str" ^
238       ", 0, 0, 0));\n" ^
239       "checkCudaErrors(cuModuleGetFunction(&function,
240         cudaModule, \"\" ^ (Utils.idtos
241         fcall.applied_kernel_function) ^ "_str" ^ "\"\"));
242       (* Copies over constants *)
243       generate_list generate_constant_on_gpu "\n"
244       fcall.constants ^ "\n" ^
245       (* Allocates GPU pointers for input and result array *)
246       let rec get_kernel_names a_info_list name_list =
247         match a_info_list with
248           | [] -> name_list
249           | hd::tl -> get_kernel_names tl
250         → (Utils.idtos(hd.kernel_name)::name_list)
251         in
252         let kernel_names = (get_kernel_names
253           fcall.input_arrays_info []) in
254           generate_list generate_device_ptr "\n" kernel_names ^
255           "\n" ^
256             generate_device_ptr
257             (Utils.idtos((fcall.return_array_info).kernel_name)) ^
258             "\n" ^
259             (* Allocations memory and copies input arrays over to
260               GPU memory *)
261             generate_mem_alloc_host_to_device fcall ^ "\n" ^
262             generate_mem_cpy_host_to_device fcall ^

```

```

242
243     (* Sets Kernel params and other information needed to
244     ← call cuLaunchKernel *)
245     generate_kernel_params fcall.input_arrays_info ^ "\n"
246     ^
247     "unsigned int blockSizeX = 16;\n" ^
248     "unsigned int blockSizeY = 1;\n" ^
249     "unsigned int blockSizeZ = 1;\n" ^
250     "unsigned int gridSizeX = 1;\n" ^
251     "unsigned int gridSizeY = 1;\n" ^
252     "unsigned int gridSizeZ = 1;\n" ^
253     (* Launches kernel *)
254     "checkCudaErrors(cuLaunchKernel(function, gridSizeX,
255     ← gridSizeY, gridSizeZ, blockSizeX, blockSizeY,
256     ← blockSizeZ, 0, NULL, KernelParams, NULL));\n" ^
257     (* Copies result array back to host *)
258     "checkCudaErrors(cuMemcpyDtoH(c, "
259     ← Utils.idtos((fcall.return_array_info).host_name) ^ ", "
260     ← sizeof(" ^ generate_variable_type
261     ← ((fcall.return_array_info).variable_type) ^ ") * "
262     ← string_of_int fcall.array_length ^ ));\n" ^
263     (* Cleanup *)
264     generate_list generate_mem_cleanup "\n"
265     ← fcall.input_arrays_info ^ "\n" ^
266     generate_mem_cleanup fcall.return_array_info ^ "\n" ^
267     generate_list generate_mem_cleanup "\n" fcall.constants
268     ^ "\n" ^
269     "checkCudaErrors(cuModuleUnload(cudaModule));\n" ^
270     "checkCudaErrors(cuCtxDestroy(context));\n" ^
271     | "reduce" ->
272     (* Fill in with VLC_Array *)
273     "{0};\n" ^
274     (* Initializes CUDA driver and loads needed function *)
275     "checkCudaErrors(cuCtxCreate(&context, 0,
276     ← device));\n" ^
277     "std::ifstream t(\"" ^ Utils.idtos
278     ← fcall.applied_kernel_function ^ ".ptx\");\n" ^
279     "if (!t.is_open()) {\n" ^
280     " std::cerr << \"\" ^ Utils.idtos
281     ← fcall.applied_kernel_function ^ ".ptx not found\n\";\n" ^
282     ^
283     "return 1;\n" ^
284     "}\n" ^
285     "std::string " ^ Utils.idtos
286     ← fcall.applied_kernel_function ^ "_str" ^
287     "((std::istreambuf_iterator<char>(t)),
288     ← std::istreambuf_iterator<char>());\n" ^
289     "checkCudaErrors(cuModuleLoadDataEx(&cudaModule, "
290     ← (Utils.idtos fcall.applied_kernel_function) ^ "_str" ^
291     ← ", 0, 0, 0));\n" ^
292     "checkCudaErrors(cuModuleGetFunction(&function,
293     ← cudaModule, \"\" ^ (Utils.idtos
294     ← fcall.applied_kernel_function) ^ "_str" ^ "\"\") );\n" ^
295     (* Copies over constants *)

```

```

275     generate_list generate_constant_on_gpu "\n"
276     ↵ fcall.constants ^ "\n" ^
277     (* Allocates GPU pointers for input and result array *)
278     let rec get_kernel_names a_info_list name_list =
279       match a_info_list with
280         | [] -> name_list
281         | hd::tl -> get_kernel_names tl
282       ↵ (Utils.idtos(hd.kernel_name) :: name_list)
283       in
284       let kernel_names = (get_kernel_names
285         ↵ fcall.input_arrays_info []) in
286         generate_list generate_device_ptr "\n" kernel_names ^
287         "\n" ^
288         generate_device_ptr
289         ↵ (Utils.idtos((fcall.return_array_info).kernel_name)) ^
290         "\n" ^
291         (* Allocations memory and copies input arrays over to
292          GPU memory *)
293         generate_mem_alloc_host_to_device fcall ^ "\n" ^
294         generate_mem_cpy_host_to_device fcall
295
296
297     (* Sets Kernel params and other information needed to
298      call cuLaunchKernel *)
299     generate_kernel_params fcall.input_arrays_info ^ "\n"
300   ^
301   "unsigned int blockSizeX = 1;\n" ^
302   "unsigned int blockSizeY = 1;\n" ^
303   "unsigned int blockSizeZ = 1;\n" ^
304   "unsigned int gridSizeX = 1;\n" ^
305   "unsigned int gridSizeY = 1;\n" ^
306   "unsigned int gridSizeZ = 1;\n" ^
307   (* Launches kernel *)
308   "checkCudaErrors(cuLaunchKernel(function, gridSizeX,
309   gridSizeY, gridSizeZ, blockSizeX, blockSizeY,
310   blockSizeZ, 0, NULL, KernelParams, NULL));\n" ^
311   (* Copies result array back to host *)
312   "checkCudaErrors(cuMemcpyDtoH(c, " ^
313   ↵ Utils.idtos((fcall.return_array_info).host_name) ^ ",",
314   ↵ sizeof(" ^ generate_variable_type
315   ↵ ((fcall.return_array_info).variable_type) ^ ") * " ^
316   ↵ string_of_int fcall.array_length ^ "));\n" ^
317   (* Cleanup *)
318   generate_list generate_mem_cleanup "\n"
319   ↵ fcall.input_arrays_info ^ "\n" ^
320   generate_mem_cleanup fcall.return_array_info ^ "\n" ^
321   generate_list generate_mem_cleanup "\n" fcall.constants
322   ^ "\n" ^
323   "checkCudaErrors(cuModuleUnload(cudaModule));\n" ^
324   "checkCudaErrors(cuCtxDestroy(context));\n"
325   | _ -> raise
326   ↵ Exceptions.Unknown_higher_order_function_call
327   ↵ in sprintf "%s" higher_order_function_call_string
328
329
330
331

```

```

311
312 let generate_variable_statement vstatement =
313   let vstatement_string = match vstatement with
314     | Declaration(d) ->
315       (generate_vdecl d) ^ ";"^"\n"
316     | Assignment (id, e) ->
317       (generate_id id) ^ "=" ^ (generate_expression e) ^
318       ";"^"\n"
319     | Initialization(d,e) ->
320       (generate_vdecl d) ^ "=" ^ (generate_expression e) ^
321       ";"^"\n"
322   (*      / _ -> raise Exceptions.Unknown_variable_statement
323   *)
324   in sprintf "%s" vstatement_string
325
326 (* Generates statements *)
327 let rec generate_statement statement =
328   let statement_string = match statement with
329     | Variable_Statement(vsmtm) ->
330       generate_variable_statement vsmtm
331     | Expression(e) ->
332       (generate_expression e) ^ ";"^"\n"
333     | Block(stmt_list) -> generate_list generate_statement
334     -->
335     "" stmt_list
336     | If(e,stmt1,stmt2) ->
337       (match stmt2 with
338         | Block([]) -> "if(" ^ (generate_expression e) ^
339           ")" ^ (generate_statement stmt1) ^ ")"^"\n"
340         | _ -> "if(" ^ (generate_expression e) ^ "){"^"\n" ^
341           (generate_statement stmt1) ^ "};"^"\n" ^
342           (generate_statement stmt2) ^ ")"^"\n"
343     | While(e,stmt) -> "while(" ^ (generate_expression e) ^
344           ")"^"\n" ^
345           (generate_statement stmt) ^ ")"^"\n"
346     | For(stmt1,e,stmt2,stmt3) -> "for(" ^
347           (generate_statement stmt1) ^ (generate_expression e) ^
348           ";" ^ (generate_statement stmt2) ^ ")"^"\n" ^
349           (generate_statement stmt3) ^ ")"^"\n"
350     | Return(e) ->
351       "return" ^ (generate_expression e) ^ ";"^"\n"
352     | Return_Void ->
353       "return;"^"\n"
354     | Continue ->
355       "continue;"^"\n"
356     | Break ->
357       "break;"^"\n"
358   (*      / _ -> raise Exceptions.Unknown_type_of_statement *)
359   in sprintf "%s" statement_string
360
361 (* Generates function declarations *)
362 let generate_fdecl f =
363   let fdecl_string =
364     (generate_variable_type f.c_fdecl_return_type) ^ " " ^

```

```

353     (generate_id f.c_fdecl_name) ^ "(" ^
354     (generate_list generate_param ", " f.c_fdecl_params) ^
355     → ")" { \n ^
356     (generate_list generate_statement "\n" f.c_fdecl_body)
357     → ^ " }\n"
358     in
359     sprintf "%s" fdecl_string
360
361 (* Writing out to CUDA file *)
362 let write_cuda filename cuda_program_string =
363   let file = open_out (filename ^ ".cu") in
364   fprintf file "%s" cuda_program_string
365
366 (* Generates the full CUDA file *)
367 let generate_cuda_file filename program =
368   let cuda_program_body =
369     (generate_list generate_variable_statement "") ^
370     (Utils.triple fst (program)) ^
371     (generate_list generate_fdecl "") ^
372     (Utils.triple trd (program))
373
374   in
375   let cuda_program_string = sprintf "\n\
376   #include <stdio.h>\n\
377   #include <stdlib.h>\n\
378   #include \"cuda.h\"\n\
379   #include <iostream>\n\
380   #include <vlc.hpp>\n\
381   CUdevice device;\n\
382   CUmodule cudaModule;\n\
383   CUcontext context;\n\
384   CUfunction function;\n\
385   %s" cuda_program_body in
386   write_cuda filename cuda_program_string
387
388 (* Generate program *)
389 let generate_program cuda_filename program =
390   generate_cuda_file cuda_filename program;
391   Codegen_ptx.generate_ptx_function_files program

```

codegen_ptx.ml

```
1  open Sast
2  (* open Exceptions *)
3  (* For sprintf *)
4  open Printf
5  (*-----*
6  ↳ KERNEL CODE GENERATION
7  ↳ -----*)
8  (*
9  let generate_kernel_fdecl kernel_f =
10   Environment.combine [
11     Generator(generate_variable_type
12     ↳ kernel_f.kernel_r_type);
13     Verbatim(" ");
14     Generator(generate_id kernel_f.kernel_name);
15     Verbatim("(");
16     Generator(generate_parameter_list
17     ↳ kernel_f.kernel_params);
18     Verbatim("\n");
19     Generator(generate_statement_list
20     ↳ kernel_f.kernel_body);
21     Verbatim("}\n");
22   ]
23   let rec generate_nonempty_kernel_fdecl_list
24     ↳ kernel_fdecl_list =
25     match kernel_fdecl_list with
26     | kernel_fdecl :: [] -> Environment.combine
27     ↳ [Generator(generate_kernel_fdecl kernel_fdecl)]
28     | kernel_fdecl :: tail ->
29       Environment.combine [
30         Generator(generate_kernel_fdecl kernel_fdecl);
31         Verbatim("\n\n");
32         Generator(generate_nonempty_kernel_fdecl_list tail)
33       ]
34     | [] -> raise (Empty_kernel_fdecl_list)
35   and generate_kernel_fdecl_list kernel_fdecl_list =
36     match kernel_fdecl_list with
37     | [] -> Environment.combine [Verbatim("")]
38     | decl :: tail -> Environment.combine
39     ↳ [Generator(generate_nonempty_kernel_fdecl_list
40     ↳ kernel_fdecl_list)]
41   *)
42
43
44
45
46  (*-----* Duplicated in
47  ↳ codegen_c-----*)
48
49  (* Generate id *)
50  let generate_id id =
51    sprintf "%s" (Utils.idtos(id))
```

```

41 (* Calls generate_func for every element of the list and
42    → concatenates results with specified concat symbol
43    Used if you need to generate a list of something - e.x.
44    → list of statements, list of params *)
45 let generate_list generate_func concat mylist =
46   let list_string = String.concat concat (List.map
47     → generate_func mylist) in
48   sprintf "%s" list_string
49
50 (*-----
51
52 let generate_ptx_binary_operator operator =
53   let op = match operator with
54     | Ptx_Add -> "add"
55     | Ptx_Subtract -> "sub"
56     | Ptx_Multiply -> "mul"
57     | Ptx_Divide -> "div"
58     | Ptx_Modulo -> "rem"
59   in
60   sprintf "%s" op
61
62 let generate_ptx_data_type data_type =
63   let t = match data_type with
64     | U16 -> ".u16"
65     | U32 -> ".u32"
66     | U64 -> ".u64"
67     | S16 -> ".s16"
68     | S32 -> ".s32"
69     | S64 -> ".s64"
70   in
71   sprintf "%s" t
72
73 let generate_ptx_variable_type vtype =
74   let v = ""
75   (* TODO *)
76   in
77   sprintf "%s" v
78
79 let generate_ptx_vdecl dtype vtype id =
80   let v =
81     ".param " ^ generate_ptx_data_type dtype ^ " "
82     → generate_ptx_variable_type vtype
83     ^ " " ^ generate_id id
84   in
85   sprintf "%s" v
86
87 let generate_ptx_register_decl declaration =
88   let r = match declaration with
89     | Register_Declaration(dtype, name, size) -> ".reg "
90     → generate_ptx_data_type dtype
91     ^ " " ^ name ^ "<" ^ string_of_int size ^ "> ; \n"
92   in
93   sprintf "%s" r

```

```

89
90 let generate_ptx_register register =
91   let r = match register with
92     | Register(s, i) -> "%" ^ s ^ string_of_int i
93   in
94   sprintf "%s" r
95
96 let generate_ptx_parameter parameter =
97   let p = match parameter with
98     | Parameter_register(r) -> generate_ptx_register(r)
99     | Parameter_constant(c) -> string_of_int c
100    | Parameter_variable(v) -> "[" ^ generate_id v ^ "]"
101  in
102  sprintf "%s" p
103
104 let generate_ptx_expression expression =
105   let e = match expression with
106     | Ptx_reg_declaration(r) -> generate_ptx_register_decl(r)
107     | Ptx_Binop(o, t, p1, p2, p3) ->
108       → generate_ptx_binary_operator(o) ^
109       → generate_ptx_data_type(t)
110       ^ " " " " ^ generate_ptx_parameter(p1) ^ ", " ^
111       → generate_ptx_parameter(p2) ^ ", "
112       ^ generate_ptx_parameter(p3) ^ "; \n"
113     | Ptx_Return -> "ret; \n"
114   in
115   sprintf "%s" e
116
117 let generate_ptx_subroutine subroutine =
118   let s =
119     generate_id subroutine.routine_name ^ ": \n" ^
120     generate_list generate_ptx_expression ""
121     → subroutine.routine_expressions
122   in
123   sprintf "%s" s
124
125 let generate_ptx_statement statement =
126   let s = match statement with
127     | Ptx_expression(e) -> generate_ptx_expression(e)
128     | Ptx_subroutine(s) -> generate_ptx_subroutine(s)
129   in
130   sprintf "%s" s
131
132 (* Generates the ptx function string *)
133 (* Fill in once you have the generation for other ptx types
134   → in the last *)
135 (*
136   should look like this
137   .entry <function name>(
138     <param list>
139   ) {
140     <statement list>

```

```

137     }
138   *)
139
140
141 (* Writing out to PTX file *)
142 let write_ptx filename ptx_string =
143   let file = open_out (filename ^ ".ptx") in
144   fprintf file "%s" ptx_string
145
146
147 (* Before each program include
148 // Generated by Vlc
149 .version 3.1
150 .target sm_20
151 .address_size 64
152 *)
153 (* Generates the ptx function string *)
154 let generate_ptx_function f =
155   let ptx_function_body =
156     ".visible .entry " ^ f.ptx_fdecl_name ^ "(" ^
157     (generate_list generate_ptx_vdecl ", "
158     f.ptx_fdecl_params) ^ ")\\n" ^
159     "{ " ^
160     (generate_list generate_ptx_register_decl "\\n"
161     f.register_declarations) ^ "\\n" ^
162     (generate_list generate_ptx_statement ""
163     f.ptx_fdecl_body) ^
164     "}"
165   in
166   let ptx_function_string = sprintf "
167   .version 3.1
168   .target sm_20
169   .address_size 64
170   %s" ptx_function_body
171   in
172   sprintf "%s" ptx_function_body
173
174 (* Main function for generating all ptx files*)
175 let generate_ptx_function_files program =
176   let ptx_function_list = Utils.triple_snd(program) in
177   let rec generate_ptx_files ptx_func_list =
178     match ptx_func_list with
179     | [] -> ()
180     | hd::tl ->
        write_ptx (Utils.idtos(hd.ptx_fdecl_name))
        (generate_ptx_function hd);
        generate_ptx_files tl
181   in generate_ptx_files ptx_function_list

```

vlc.ml

```
1 type action = Tokens | Ast | Compile | Sast | Run
2
3 let _ =
4   if Array.length Sys.argv < 2 then
5     print_string (
6       "Usage: vlc [mode] <VLC program file>\n" ^
7       "\t-t: prints tokens read in by scanner\n" ^
8       "\t-a: prints ast as a program\n" ^
9       "\t-s: prints sast as a program\n" ^
10      "\t-c: compiles VLC program to CUDA C file and PTX
11      \n"
12    ) else
13      let action = List.assoc Sys.argv.(1) [ (" -t ", Tokens);
14                                         (" -a ", Ast);
15                                         (" -s ", Sast);
16                                         (" -c ", Compile);
17                                         (" -r ", Run) ] and
18      filename = Sys.argv.(2) in
19      print_endline filename;
20      (* let base_filename = List.hd (Str.split (Str.regexp
21      ".vlc")) (List.hd (List.rev (Str.split (Str.regexp "/"
22      filename)))) in
23      *) let file_in = open_in filename in
24        let lexbuf = Lexing.from_channel file_in in
25        let token_list = Processor.get_token_list lexbuf in
26        let program = Processor.parser token_list in
27        let sast = Semant.analyze program in
28        match action with
29          | Tokens ->
30            print_string (Utils.token_list_to_string
31            token_list)
32          | Ast ->
33            print_string (Utils.program_to_string program)
34          | Sast ->
35            print_string (Utils.sast_to_string sast)
36          | Compile ->
37            Codegen_c.generate_program filename sast
38          | Run ->
39            Codegen_c.generate_program filename sast
40            Sys.command ("nvcc -" ^ filename ^ " " ^
41            filename ^ ".cu");
42            Sys.command ("./" ^ filename); *)
43
```

Makefile

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

Interfaces

ast.ml

```
1  type binary_operator =
2    | Add | Subtract | Multiply | Divide | Modulo
3    (*      / Plus_Equal | Subtract_Equal | Multiply_Equal | 
4      ↵ Divide_Equal *)
5    (*      / Exp | Dot | Matrix_Multiplication *)
6    | And | Or | Xor
7    | Equal | Not_Equal | Greater_Than | Less_Than |
8      ↵ Greater_Than_Equal | Less_Than_Equal
9    | Bitshift_Right | Bitshift_Left
10   type unary_operator =
11     | Not | Negate
12     | Plus_Plus | Minus_Minus
13
14
15  type data_type =
16    | String
17    | Byte
18    | Unsigned_Byte
19    | Integer
20    | Unsigned_Integer
21    | Long
22    | Unsigned_Long
23    | Float
24    | Double
25    | Boolean
26    | Void
27
28  type variable_type =
29    | Primitive of data_type
30    | Array of variable_type * int (* variable type, size
31      ↵ *)
32    (* / Struct of variable_type list * expression list * int
33      ↵ *)
34
35  type vdecl =
36    Variable_Declaration of variable_type * identifier
37
38  type expression =
39    | Function_Call of identifier * expression list
40    | Higher_Order_Function_Call of
41      ↵ higher_order_function_call
42    | String_Literal of string
43    | Integer_Literal of int
44    | Boolean_Literal of bool
```

```

42     | Floating_Point_Literal of float
43     | Array_Literal of expression list
44     | Identifier_Literal of identifier
45     | Cast of variable_type * expression
46     | Binop of expression * binary_operator * expression
47     | Unop of expression * unary_operator
48     | Array_Accessor of expression * expression list (*
49     → Array, indexes *)
50   and constant =
51     | Constant of identifier * expression
52   and higher_order_function_call = {
53     higher_order_function_type : ...
54     → identifier; (* Map or reduce *)
55     kernel_function_name : ...
56     → identifier;
57     constants : ...
58     → constant list; ...
59     input_arrays : ...
60     → expression list; (* Check in semantic analyzer that
61     → type is array*)
62   }
63
64   type variable_statement =
65     | Declaration of vdecl
66     | Initialization of vdecl * expression
67     | Assignment of expression * expression
68
69   type statement =
70     | Variable_Statement of variable_statement
71     | Expression of expression
72     | Block of statement list (* Used for if, else, for,
73     → while blocks *)
74     | If of expression * statement * statement (*
75     → expression-condition, statement-if block,
76     → statement-optional else block *)
77     | While of expression * statement
78     | For of statement * expression * statement * statement
79     | Return of expression
80     | Return_Void
81     | Continue
82     | Break
83
84   type fdecl = {
85     is_kernel_function : bool;
86     → (* Host or Kernel *)
87     return_type :
88     → variable_type;
89     name :
90     → identifier;
91     params : vdecl
92     → list;

```

```
81      body :  
82      ↵ statement list;  
83  }  
84 (* Program Definition *)  
85 type program = variable_statement list * fdecl list
```

sast.ml

```
1  open Ast
2  (* Contains sast type definitions for conversions during
   ↳ semantic analysis *)
3  (* -----PTX types
   ↳ -----*)
4  type ptx_data_movement =
5    | Ptx_Move | Ptx_Load | Ptx_Store
6
7  type ptx_binary_operator =
8    | Ptx_Add | Ptx_Subtract | Ptx_Multiply | Ptx_Divide |
   ↳ Ptx_Modulo
9
10 type ptx_data_type =
11   | U16 | U32 | U64 | S16 | S32 | S64
12
13 (* should use this as our information about global/param
   ↳ etc.*)
14 type ptx_variable_type =
15   | Ptx_Primitive of ptx_data_type
16   | Ptx_Array of ptx_variable_type * int          (* 'int'
   ↳ refers to the length of the array *)
17   | Ptx_Pointer of ptx_variable_type * int         (* 'int'
   ↳ refers to size of memory pointed by the pointer *)
18
19 type ptx_register_decl =
20   | Register_Declaration of ptx_data_type * string * int
   ↳      (* type, name, number of registers *)
21
22 type ptx_register =
23   | Register of string * int          (* register
   ↳ name, register number *)
24 (* Not sure what this is | Typed_Register of ptx_data_type
   ↳ * string * int      (* type, register name, register
   ↳ number *) *)
25 (* Implement later | Special_Register of string
   ↳                  (* register name *) *)
26
27 type ptx_parameter =
28   | Parameter_register of ptx_register
29   | Parameter_constant of int
30   | Parameter_variable of Ast.identifier
31
32
33 type ptx_expression =
34   | Ptx_reg_declaration of ptx_register_decl
35   | Ptx_movement of ptx_data_movement * ptx_data_type *
   ↳ ptx_variable_type * ptx_parameter * ptx_parameter
36   | Ptx_Binop of ptx_binary_operator * ptx_data_type *
   ↳ ptx_parameter * ptx_parameter * ptx_parameter
37   | Ptx_Return
```

```

38      (*      / Ptx_Array_Literal of ptx_expression list
39      / Ptx_Function_Call of Ast.identifier * ptx_expression
40      ↳ list
41      / Ptx_Identifier_Expression of Ast.identifier
42      *)
43
43 type ptx_subroutine = {
44     routine_name           : Ast.identifier;
45     routine_expressions    : ptx_expression list;
46 }
47
48 type ptx_statement =
49     (*      / Ptx_Initialization of ptx_vdecl * ptx_expression
50     ↳ *)
51     (*      / Ptx_Assignment of Ast.identifier * ptx_expression
52     ↳ *)
53     | Ptx_expression of ptx_expression
52     | Ptx_subroutine of ptx_subroutine
54
54 type ptx_function_type =
55     | Global
56     | Device
57
58 type ptx_constant =
59 {
60     ptx_constant_name       : Ast.identifier;
61     ptx_constant_variable_type : ptx_variable_type;
62 }
63
64 type ptx_variable_space =
65     | Global
66     | Local
67     | Shared
68
69 type ptx_vdecl =
70     | Ptx_Vdecl of ptx_data_type * ptx_variable_space (*
71     ↳ need something about global/ptrs here*)
71     ↳ ptx_variable_type * Ast.identifier
72
73 (* ptx fdecl is the entire file
74   it seems it really only needs to be composed of a few
75   ↳ parts - a name, a variable declaration list
75   and a statement list
76   register_decl list should go inside body generated from
76   ↳ semantic analyzer
77 *)
78 type ptx_fdecl = {
79     (* Global or Device *)
80     ptx_fdecl_type          : ptx_function_type; (*
80     ↳ probably not needed *)
81
82     (* Name of the function *)

```

```

83     ptx_fdecl_name           : Ast.identifier;
84
85     (* Expected parameters of the function *)
86     ptx_fdecl_params         : ptx_vdecl list;
87
88     (* List of constants that function needs to know - aka
89      ↳ variables that aren't in scope of function when it goes
89      ↳ through semantic analyzer
89      If this constant list doesn't match the constant list
89      ↳ of the higher order function, throw error in semant.ml
89      ↳ *)
90     ptx_consts                : ptx_constant list;
91     (* Declares the virtual registers that are needed for the
91      ↳ function *)
92     register_decls            : ptx_register_decl list;
93     (* Statements within the function *)
94     ptx_fdecl_body             : ptx_statement list;
95 }

96
97
98
99
100
101
102 (* -----C types
102   ↳ -----*)
103
104 (*-----*
104   ↳ Unnecessary?????????-----*
104   ↳ *)
105 type c_binary_operator =
106   | Add | Subtract | Multiply | Divide | Modulo
107   (*   / Plus_Equal | Subtract_Equal | Multiply_Equal |
107   ↳ Divide_Equal *)
108   (*   / Exp | Dot | Matrix_Multiplication *)
109   | And | Or | Xor
110   | Equal | Not_Equal | Greater_Than | Less_Than |
110   ↳ Greater_Than_Equal | Less_Than_Equal
111   | Bitshift_Right | Bitshift_Left
112 type c_unary_operator =
113   | Not | Negate
114   | Plus_Plus | Minus_Minus
115
116 type c_data_type =
117   | String
118   | Byte
119   | Unsigned_Byte
120   | Integer
121   | Unsigned_Integer
122   | Long
123   | Unsigned_Long
124   | Float
125   | Double

```

```

126     | Boolean
127     | Void
128
129 type c_variable_type =
130     | Primitive of c_data_type
131     | Array of c_variable_type * int
132 (* / Struct of variable_type list * expression list * int
133   ↵ *)
134
135 type c_vdecl =
136     Variable_Declaration of c_variable_type *
137   ↵ Ast.identifier
138
139 (*
140   -----Necessary-----
141   ↵ *)
142
143 type c_kernel_variable_info = {
144     variable_type      : c_variable_type;
145     host_name         : Ast.identifier;
146     kernel_name       : Ast.identifier;
147 }
148
149 type c_higher_order_function_call = {
150     (* Map or reduce *)
151     higher_order_function_type      : Ast.identifier;
152     (* Name of kernel function that is called from host
153       (would be kernel function corresponding to map/reduce)
154       *)
155     applied_kernel_function        : Ast.identifier;
156     (* List of constants passed into map and reduce *)
157     constants                     : c_kernel_variable_info list;
158     (* Size of input and return arrays *)
159     array_length                  : int;
160     (* Input array information
161       --If an array has no name (just simply passed in as
162         something like {1,2,3}) then it is given a temporary
163         generated name *)
164     input_arrays_info              : c_kernel_variable_info
165     list; (* type, host name, kernel name *)
166     (* Return array information *)
167     return_array_info             :
168     c_kernel_variable_info; (* type, host name, kernel
169     name*)
170 }
171
172 (* Type for calling defg functions directly from host *)
173 type c_kernel_function_call = {
174     (* Name of the function that is called from the host *)
175     kernel_function               : Ast.identifier;
176     (* Input array information
177       --If an array has no name (just simply passed in as
178         something like {1,2,3}) then it is given a temporary
179         generated name *)

```

```

167     input_args_info           : c_kernel_variable_info
168     ↳ list; (* type, host name, kernel name *)
169     (* Return array information *)
170     return_arg_info          :
171     ↳ c_kernel_variable_info; (* type, host name, kernel
172     ↳ name*)
173 }
174
175 type c_expression =
176   | Function_Call of Ast.identifier * c_expression list
177   | Higher_Order_Function_Call of
178     ↳ c_higher_order_function_call
179   | Kernel_Function_Call of c_kernel_function_call
180   | String_Literal of string
181   | Integer_Literal of int
182   | Boolean_Literal of bool
183   | Floating_Point_Literal of float
184   | Array_Literal of c_expression list
185   | Identifier_Literal of Ast.identifier
186   | Cast of c_variable_type * c_expression
187   | Binop of c_expression * c_binary_operator *
188   ↳ c_expression
189   | Unop of c_expression * c_unary_operator
190   | Array_Accessor of c_expression * c_expression list (*
191   ↳ Array, indexes *)
192   | Ternary of c_expression * c_expression * c_expression
193   ↳ (* expression if true, condition, expression if false
194   ↳ *)
195
196 type c_variable_statement =
197   | Declaration of c_vdecl
198   | Initialization of c_vdecl * c_expression
199   | Assignment of Ast.identifier * c_expression
200
201 type c_statement =
202   | Variable_Statement of c_variable_statement
203   | Expression of c_expression
204   | Block of c_statement list (* Used for if, else, for,
205   ↳ while blocks *)
206   | If of c_expression * c_statement * c_statement (*
207   ↳ expression-condition, statement-if block,
208   ↳ statement-optional else block *)
209   | While of c_expression * c_statement
210   | For of c_statement * c_expression * c_statement *
211   ↳ c_statement
212   | Return of c_expression
213   | Return_Void
214   | Continue
215   | Break
216
217 type c_fdecl = {
218   c_fdecl_return_type      : c_variable_type;

```

```
207     c_fdecl_name          : Ast.identifier;
208     c_fdecl_params        : c_vdecl list;
209     c_fdecl_body          : c_statement list;
210 }
211
212 (* Overall Program *)
213 type program = c_variable_statement list * ptx_fdecl list *
214   ↳ c_fdecl list
```

exceptions.ml

```
1 (* Collection of exceptions for different parts of the
   ↳ compiler *)
2
3 (*-----Scanner-----)
4 exception Bad_dedent
5 (*-----Parser-----)
6 exception Array_parsing_error
7 exception Invalid_data_type of string
8
9 exception Lexing_error of string (* Unused atm *)
10 exception Parsing_error of string (* Unused atm *)
11 (*-----Processor-----)
12 exception Missing_eof
13 (*-----Utils-----)
14 (*-----Semantic
   ↳ Analyzer-----*)
15 exception Cannot_infer_expression_type
16 exception Exception of string
17 exception Already_declared
18 exception Name_not_found of string
19 exception Invalid_environment
20 exception Variable_not_found_in_scope
21 exception Function_not_defined
22 exception Cannot_pop_empty_variable_scope_stack
23 exception Variable_already_declared
24 exception Not_an_array_expression
25 exception Type_mismatch of string
26 exception Empty_array_expression_list
27 exception Variable_not_declared
28 (*-----Codegen
   ↳ C-----*)
29 exception Unknown_variable_type
30 exception Unknown_operator
31 exception Unknown_data_type
32 exception Unknown_type_of_param
33 exception Unknown_higher_order_function_call
34 exception Unknown_type_of_vdecl
35 exception Unknown_type_of_expression
36 exception Unknown_variable_statement
37 exception Unknown_type_of_statement
38 (*-----Codegen
   ↳ PTX-----*)
```

utils.ml

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

```

50 |   Mult      -> "★"
51 |   Div       -> "/"
52 |   Equal     -> "==""
53 |   Neq        -> "!="
54 |   Less       -> "<""
55 |   Leq        -> "<="
56 |   Greater   -> ">""
57 |   Geq        -> ">="
58 |   And         -> "and"
59 |   Not         -> "not"
60 |   Or          -> "or"
61 |   Mod         -> "%"

62
63 let rec string_of_bracket_expr = function
64 []
65 |   head :: tail -> "[" ^ (string_of_expr head) ^ "]"
66 |   ↵ (string_of_bracket_expr tail)
67 and string_of_array_primitive = function
68 []
69 |   [last]      -> (string_of_expr last)
70 |   head :: tail -> (string_of_expr head) ^ ", "
71 |   ↵ (string_of_array_primitive tail)
72 and string_of_expr = function
73 |   Int_Lit(i)    -> string_of_int i
74 |   Boolean_Lit(b)  -> if b then "true" else "false"
75 |   Float_Lit(f)   -> string_of_float f
76 |   String_Lit(s)  -> "\"" ^ (String.escaped s) ^ "\""
77 |   Char_Lit(c)   -> Char.escaped c
78 |   This          -> "this"
79 |   Id(s)         -> s
80 |   Binop(e1, o, e2) -> (string_of_expr e1) ^ " "
81 |   ↵ (string_of_op o) ^ " " ^ (string_of_expr e2)
82 |   Assign(e1, e2)   -> (string_of_expr e1) ^ " = "
83 |   ↵ (string_of_expr e2)
84 |   Noexpr         -> ""
85 |   ObjAccess(e1, e2) -> (string_of_expr e1) ^ "."
86 |   ↵ (string_of_expr e2)
87 |   Call(f, el)    -> f ^ "(" ^ String.concat ", "
88 |   ↵ (List.map string_of_expr el) ^ ")"
89 |   ArrayPrimitive(el) -> "|"
90 |   ↵ (string_of_array_primitive el) ^ "|"
91 |   Unop(op, e)    -> (string_of_op op) ^ "(" ^
92 |   ↵ string_of_expr e ^ ")"
93 |   Null           -> "null"
94 |   ArrayCreate(d, el) -> "new " ^ string_of_datatype d
95 |   ↵ string_of_bracket_expr el
96 |   ArrayAccess(e, el) -> (string_of_expr e) ^
97 |   ↵ (string_of_bracket_expr el)
98 |   ObjectCreate(s, el) -> "new " ^ s ^ "(" ^
99 |   ↵ String.concat ", "
100 |   ↵ (List.map string_of_expr el) ^ ")"
101 |   Delete(e)      -> "delete (" ^ (string_of_expr e) ^
102 |   ↵ ")"

```

```

90 ; ;
91
92 let rec string_of_bracket_sexp = function
93 []
94 | head :: tail -> "[" ^ (string_of_sexp head) ^ "]" ^ "
95 and string_of_sarray_primitive = function
96 []
97 | [last] -> (string_of_sexp last)
98 | head :: tail -> (string_of_sexp head) ^ ", " ^
99 and string_of_sexp = function
100 SInt_Lit(i) -> string_of_int i
101 | SBoolean_Lit(b) -> if b then "true" else "false"
102 | SFLOAT_Lit(f) -> string_of_float f
103 | SString_Lit(s) -> "\"" ^ (String.escaped s) ^
104 | SChar_Lit(c) -> Char.escaped c
105 | SId(s, _) -> s
106 | SBinop(e1, o, e2, _) -> (string_of_sexp e1) ^ " " ^
107 | SAssign(e1, e2, _) -> (string_of_sexp e1) ^ " = "
108 | SNoexpr -> ""
109 | SObjAccess(e1, e2, _) -> (string_of_sexp e1) ^ ". " ^
110 | SCall(f, el, _, _) -> f ^ "(" ^ String.concat ", " ^
111 | SArrayPrimitive(el, _) -> "| " ^
112 | SUnop(op, e, _) -> (string_of_op op) ^ "(" ^
113 | SNull -> "null"
114 | SArrayCreate(d, el, _) -> "new " ^
115 | SArrayAccess(e, el, _) -> (string_of_sexp e) ^
116 | SObjectCreate(s, el, _) -> "new " ^ s ^ "(" ^
117 | SDelete(e) -> "delete (" ^ (string_of_sexp
118 | e) ^ ")"
119 ; ;
120 let string_of_local_expr = function
121 Noexpr -> ""
122 | e -> " = " ^ string_of_expr e
123
124 (* Print statements *)
125
126 let rec string_of_stmt indent =
127 let indent_string = String.make indent '\t' in
128 let get_stmt_string = function
129

```

```

130 Block(stmts)           ->
131 indent_string ^ "{\n" ^
132   String.concat "" (List.map (string_of_stmt (indent+1))
133   ↵ stmts) ^
134   indent_string ^ "}\n"
135 | Expr(expr)           ->
136 indent_string ^ string_of_expr expr ^ ";"\n";
137 |
138 | Return(expr)         ->
139 indent_string ^ "return " ^ string_of_expr expr ^ ";"\n";
140 |
141 | If(e, s, Block([Expr(Noexpr)])) ->
142 indent_string ^ "if (" ^ string_of_expr e ^ ") \n" ^
143 (string_of_stmt (indent+1) s)
144 |
145 | If(e, s1, s2)        ->
146 indent_string ^ "if (" ^ string_of_expr e ^ ") \n" ^
147 string_of_stmt (indent+1) s1 ^
148 indent_string ^ "else\n" ^
149 string_of_stmt (indent+1) s2
150 |
151 | For(e1, e2, e3, s)   ->
152 indent_string ^ "for (" ^ string_of_expr e1 ^ " ; " ^
153   ↵ string_of_expr e2 ^ " ; " ^ string_of_expr e3 ^ ") \n" ^
154   ↵
155 string_of_stmt (indent) s
156 |
157 | While(e, s)          ->
158 indent_string ^ "while (" ^ string_of_expr e ^ ") \n" ^
159 string_of_stmt (indent) s
160 |
161 | Break                 -> indent_string ^ "break;\n"
162 | Continue              -> indent_string ^ "continue;\n"
163 | Local(d, s, e)        -> indent_string ^
164   ↵ string_of_datatype d ^ " " ^ s ^ string_of_local_expr e
165   ↵ ^ ";\n"
166 in get_stmt_string
167 |
168 let string_of_local_sexpr = function
169   SNoexpr    -> ""
170   | e           -> " = " ^ string_of_sexpr e
171 |
172 let rec string_of_sstmt indent =
173   let indent_string = String.make indent '\t' in
174   let get_stmt_string = function
175   |
176   | SBlock(stmts)       ->
177     indent_string ^ "{\n" ^
178       String.concat "" (List.map (string_of_sstmt (indent+1))
179       ↵ stmts) ^
180       indent_string ^ "}\n"

```

```

177 |   SExpr(expr, _)           ->
178 indent_string ^ string_of_expr expr ^ ";" \n;
179
180 |   SReturn(expr, _)        ->
181 indent_string ^ "return " ^ string_of_expr expr ^ ";" \n;
182
183 |   SIf(e, s, SBlock([SExpr(SNoexpr, _)]))    ->
184 indent_string ^ "if (" ^ string_of_expr e ^ ") \n" ^
185 (string_of_sstmt (indent+1) s)
186
187 |   SIf(e, s1, s2)         ->
188 indent_string ^ "if (" ^ string_of_expr e ^ ") \n" ^
189 string_of_sstmt (indent+1) s1 ^
190 indent_string ^ "else\n" ^
191 string_of_sstmt (indent+1) s2
192
193 |   SFor(e1, e2, e3, s)    ->
194 indent_string ^ "for (" ^ string_of_expr e1 ^ " ; " ^
195     ↳ string_of_expr e2 ^ " ; " ^ string_of_expr e3 ^
196     ↳ ") \n" ^
197 string_of_sstmt (indent) s
198
199 |   SWhile(e, s)          ->
200 indent_string ^ "while (" ^ string_of_expr e ^ ") \n" ^
201 string_of_sstmt (indent) s
202
203 |   SBreak                 -> indent_string ^ "break;\n"
204 |   SContinue              -> indent_string ^ "continue;\n"
205 |   SLocal(d, s, e)        -> indent_string ^
206     ↳ string_of_datatype d ^ " " ^ s ^ string_of_local_expr
207     ↳ e ^ ";" \n"
208 in get_stmt_string
209
210 (* Print Function *)
211
212 let string_of_fname = function
213   Constructor -> "constructor"
214 |   FName(s)  -> s
215
216 let string_of_formal = function
217   Formal(d, s) -> (string_of_datatype d) ^ " " ^ s
218 |   _             -> ""
219
220 let string_of_formal_name = function
221   Formal(_, s) -> s
222 |   _             -> ""
223
224 let string_of_func_decl fdecl =
225   "" ^ (string_of_scope fdecl.scope) ^ " " ^
226     ↳ (string_of_datatype fdecl.returnType) ^ " " ^
227     ↳ (string_of_fname fdecl.fname) ^ " " ^
228 (* Formals *)

```

```

223 " (" ^ String.concat ", " (List.map string_of_formal
224   ~ fdecl.formals) ^ ")") { \n" ^
225   (* body *)
226   String.concat "" (List.map (string_of_stmt 2) fdecl.body)
227   ~
228   "\t}\n\n"
229
230 (* Class Printing *)
231
232 let string_of_extends = function
233   NoParent -> ""
234   | Parent(s) -> "extends " ^ s ^ " "
235
236 let string_of_field = function
237   Field(s, d, id) -> (string_of_scope s) ^ " " ^ "
238   ~ (string_of_datatype d) ^ " " ^ id ^ ";" ; \n"
239
240 let string_of_cbody cbody =
241   String.concat "" (List.map (fun s -> "\t" ^ s) (List.map
242     ~ string_of_field cbody.fields)) ^
243   String.concat "" (List.map (fun s -> "\t" ^ s) (List.map
244     ~ string_of_func_decl cbody.constructors)) ^
245   String.concat "" (List.map (fun s -> "\t" ^ s) (List.map
246     ~ string_of_func_decl cbody.methods))
247
248 let string_of_class_decl cdecl =
249   "class " ^ cdecl.cname ^ " " ^ (string_of_extends
250     ~ cdecl.extends) ^ "{\n" ^
251     (string_of_cbody cdecl.cbody) ^
252   " }\n"
253
254 (* Include Printing *)
255
256 let rec string_of_include = function
257   Include(s) -> "include(" ^ s ^ ");\n"
258
259 (* Print whole program *)
260
261 let string_of_program = function
262   Program(includes, cdecls) ->
263   String.concat "" (List.map string_of_include includes) ^
264   ~ "\n" ^
265   String.concat "\n" (List.map string_of_class_decl cdecls)
266
267 (* Print AST tree representation *)
268
269 let includes_tree includes =
270   `List` (List.map (function Include s -> `String` s) includes)
271
272 let map_fields_to_json fields =
273   `List` (List.map (function Field(scope, datatype, s) ->
274     `Assoc` [
275       ("name", `String` s);
276       ("scope", `String` (string_of_scope scope));
277       ("datatype", `String` (string_of_datatype datatype));
278     ])
279

```

```

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

```

```

301 Block(stmts)           -> `Assoc [("block", `List (List.map
302   ↳ (map_stmt_to_json) stmts)) ]
303 | Expr(expr)          -> `Assoc [("expr", map_expr_to_json
304   ↳ expr) ]
305 | Return(expr)         -> `Assoc [("return",
306   ↳ map_expr_to_json expr) ]
307 | If(e, s1, s2)        -> `Assoc [("if", `Assoc [("cond",
308   ↳ map_expr_to_json e); ("ifbody", map_stmt_to_json s1)];
309   ↳ ("else", map_stmt_to_json s2)])
310 | For(e1, e2, e3, s)    -> `Assoc [("for", `Assoc
311   ↳ [("init", map_expr_to_json e1); ("cond",
312     ↳ map_expr_to_json e2); ("inc", map_expr_to_json e3);
313     ↳ ("body", map_stmt_to_json s))])
314 | While(e, s)          -> `Assoc [("while", `Assoc [("cond",
315   ↳ map_expr_to_json e); ("body", map_stmt_to_json s))])
316 | Break                 -> `String "break"
317 | Continue              -> `String "continue"
318 | Local(d, s, e)        -> `Assoc [("local", `Assoc
319   ↳ [("datatype", `String (string_of_datatype d)); ("name",
320     ↳ `String s); ("val", map_expr_to_json e))])
321
322 let map_methods_to_json methods =
323   `List (List.map (fun (fdecl:Ast.func_decl) ->
324     `Assoc [
325       ("name", `String (string_of_fname fdecl.fname));
326       ("scope", `String (string_of_scope fdecl.scope));
327       ("returnType", `String (string_of_datatype
328         ↳ fdecl.returnType));
329       ("formals", map_formals_to_json fdecl.formals);
330       ("body", `List (List.map (map_stmt_to_json) fdecl.body));
331     ]) methods)
332
333
334 let cdecls_tree cdecls =
335   let map_cdecl_to_json cdecl =
336     `Assoc [
337       ("cname", `String cdecl.cname);
338       ("extends", `String (string_of_extends cdecl.extends));
339       ("fields", map_fields_to_json cdecl.cbody.fields);
340       ("methods", map_methods_to_json cdecl.cbody.methods);
341       ("constructors", map_methods_to_json
342         ↳ cdecl.cbody.constructors)
343     ]
344   in
345   `List (List.map (map_cdecl_to_json) cdecls)
346
347
348 let print_tree = function
349   Program(includes, cdecls) ->
350     `Assoc [("program",
351       `Assoc([
352         ("includes", includes_tree includes);
353         ("classes", cdecls_tree cdecls)
354       ]))

```

```

341 ) ]
342 (* Print SAST tree representation *)
343
344 let rec map_expr_to_json =
345   let datatype d = [("datatype", `String (string_of_datatype
346   ~d))] in
347   function
348     SInt_Lit(i)           -> 'Assoc [("int_lit", `Assoc
349   ~v: (["val", `Int i]) @ (datatype (Datatype(Int_t))))]
350   | SBoolean_Lit(b)      -> 'Assoc [("bool_lit", `Assoc
351   ~v: (["val", `Bool b]) @ (datatype (Datatype(Bool_t))))]
352   | SFloat_Lit(f)        -> 'Assoc [("float_lit", `Assoc
353   ~v: (["val", `Float f]) @ (datatype
354   (Datatype(Float_t))))]
355   | SString_Lit(s)       -> 'Assoc [("string_lit", `Assoc
356   ~v: (["val", `String s]) @ (datatype (Arraytype(Char_t,
357   1)))])
358   | SChar_Lit(c)         -> 'Assoc [("char_lit", `Assoc
359   ~v: (["val", `String (Char.escaped c)]) @ (datatype
360   (Datatype(Char_t))))]
361   | SId(s, d)            -> 'Assoc [("id", `Assoc
362   ~v: (["name", `String s]) @ (datatype d))]
363   | SBinop(e1, o, e2, d)  -> 'Assoc [("binop", `Assoc
364   ~v: (["lhs", map_expr_to_json e1]; ("op", `String
365   (string_of_op o)); ("rhs", map_expr_to_json e2)) @
366   (datatype d))]
367   | SAssign(e1, e2, d)    -> 'Assoc [("assign", `Assoc
368   ~v: (["lhs", map_expr_to_json e1]; ("op", `String "=");
369   ("rhs", map_expr_to_json e2)) @ (datatype d))]
370   | SNoexpr               -> 'Assoc [("noexpr", `Assoc
371   (datatype (Datatype(Void_t))))]
372   | SArrayCreate(t, el, d) -> 'Assoc [("arraycreate",
373   `Assoc ([("datatype", `String (string_of_datatype d));
374   ("args", `List (List.map map_expr_to_json el)))] @
375   (datatype d))]
376   | SArrayAccess(e, el, d) -> 'Assoc [("arrayaccess",
377   `Assoc ([("array", map_expr_to_json e); ("args", `List
378   (List.map map_expr_to_json el)))] @ (datatype d))]
379   | SObjAccess(e1, e2, d)  -> 'Assoc [("objaccess", `Assoc
380   ~v: (["lhs", map_expr_to_json e1]; ("op", `String "."));
381   ("rhs", map_expr_to_json e2)) @ (datatype d))]
382   | SCall(fname, el, d, i) -> 'Assoc [("call", `Assoc
383   ~v: (["name", `String fname]; ("params", `List (List.map
384   map_expr_to_json el)); ("index", `Int i)) @ (datatype
385   d)))]
386   | SObjectCreate(s, el, d) -> 'Assoc [("objectcreate",
387   `Assoc ([("type", `String s); ("args", `List (List.map
388   map_expr_to_json el)))] @ (datatype d))]
389   | SArrayPrimitive(el, d)  -> 'Assoc [("arrayprimitive",
390   `Assoc ([("expressions", `List (List.map
391   map_expr_to_json el)))] @ (datatype d)))]

```

```

363 |   SUnop(op, e, d)           -> 'Assoc [ ("Unop", 'Assoc
364 |     ("op", `String (string_of_op op)); ("operand",
364 |     map_sexpr_to_json e) ] @ (datatype d)) ]
364 |   SNull                   -> 'Assoc [ ("null", 'Assoc
364 |     (datatype (Datatype(Void_t))) ) ]
365 |   SDelete(e)             -> 'Assoc [ ("delete", 'Assoc
365 |     (["expr", map_sexpr_to_json e]) @ (datatype
365 |     (Datatype(Void_t)))) ]
366
367 let rec map_sstmt_to_json =
368   datatype d = [("datatype", `String (string_of_datatype
368   d)) ] in
369   function
370     SBlock sl           -> 'Assoc [ ("sblock", 'List
370     (List.map (map_sstmt_to_json) sl))]
371     | SExpr(e, d)         -> 'Assoc [ ("sexpr", 'Assoc
371     (["expr", map_sexpr_to_json e]) @ (datatype d)) ]
372     | SReturn(e, d)       -> 'Assoc [ ("sreturn", 'Assoc
372     (["return", map_sexpr_to_json e]) @ (datatype d)) ]
373     | SIf (e, s1, s2)     -> 'Assoc [ ("sif", 'Assoc
373     (["cond", map_sexpr_to_json e); ("ifbody",
373     map_sstmt_to_json s1)); ("selse", map_sstmt_to_json
373     s2) ]
374     | SFor (e1, e2, e3, s) -> 'Assoc [ ("sfor", 'Assoc
374     (["init", map_sexpr_to_json e1); ("cond",
374     map_sexpr_to_json e2); ("inc", map_sexpr_to_json e3);
374     ("body", map_sstmt_to_json s)) ]
375     | SWhile (e, s)        -> 'Assoc [ ("swhile", 'Assoc
375     (["cond", map_sexpr_to_json e); ("body",
375     map_sstmt_to_json s)) ]
376     | SBreak                 -> 'String "sbreak"
377     | SContinue              -> 'String "scontinue"
378     | SLocal(d, s, e)       -> 'Assoc [ ("slocal", 'Assoc
378     (["datatype", `String (string_of_datatype d)); ("name",
378     `String s); ("val", map_sexpr_to_json e)) ]
379
380   let string_of_func_type = function
381     User -> "user" | Reserved -> "reserved"
382
383   let map_sfdecl_to_json sfdecl =
384     'Assoc [ ("sfdecl", 'Assoc [
385       ("sfname", `String (string_of_fname sfdecl.sfname));
386       ("sreturnType", `String (string_of_datatype
386       sfdecl.sreturnType));
387       ("sformals", map_formals_to_json sfdecl.sformals);
388       ("sbody", 'List (List.map (map_sstmt_to_json)
388       sfdecl.sbody));
389       ("func_type", `String (string_of_func_type
389       sfdecl.func_type));
390     ] ) ]
391
392   let map_sfdecls_to_json sfdecls =
393     'List (List.map map_sfdecl_to_json sfdecls)

```

```

394
395 let map_scdecls_to_json scdecls =
396   `List(List.map (fun scdecl ->
397     `Assoc [ ("scdecl", `Assoc [
398       ("scname", `String scdecl.scname);
399       ("sfields", map_fields_to_json scdecl.sfields);
400       ("sfuncs", map_sfdecls_to_json scdecl.sfuncs);
401     ] )
402   ] )
403   ]
404   scdecls)
405
406 let map_sprogram_to_json sprogram =
407   `Assoc [ ("sprogram", `Assoc [
408     ("classes", map_scdecls_to_json sprogram.classes);
409     ("functions", map_sfdecls_to_json sprogram.functions);
410     ("main", map_sfdecl_to_json sprogram.main);
411     ("reserved", map_sfdecls_to_json sprogram.reserved);
412   ] ) ]
413
414 (* Print tokens *)
415
416 let string_of_token = function
417   LPAREN           -> "LPAREN"
418   | RPAREN          -> "RPAREN"
419   | LBRACE          -> "LBRACE"
420   | RBRACE          -> "RBRACE"
421   | SEMI            -> "SEMI"
422   | COMMA           -> "COMMA"
423   | PLUS             -> "PLUS"
424   | MINUS           -> "MINUS"
425   | TIMES            -> "TIMES"
426   | DIVIDE           -> "DIVIDE"
427   | ASSIGN           -> "ASSIGN"
428   | EQ               -> "EQ"
429   | NEQ              -> "NEQ"
430   | LT               -> "LT"
431   | LEQ              -> "LEQ"
432   | GT               -> "GT"
433   | GEQ              -> "GEQ"
434   | AND              -> "AND"
435   | OR               -> "OR"
436   | NOT              -> "NOT"
437   | DOT              -> "DOT"
438   | LBRACKET         -> "LBRACKET"
439   | RBRACKET         -> "RBRACKET"
440   | BAR              -> "BAR"
441   | IF               -> "IF"
442   | ELSE             -> "ELSE"
443   | FOR              -> "FOR"
444   | WHILE            -> "WHILE"
445   | RETURN           -> "RETURN"

```

```

446 |     INT          -> "INT"
447 |     FLOAT         -> "FLOAT"
448 |     BOOL          -> "BOOL"
449 |     CHAR          -> "CHAR"
450 |     VOID          -> "VOID"
451 |     NULL          -> "NULL"
452 |     TRUE          -> "TRUE"
453 |     FALSE         -> "FALSE"
454 |     CLASS          -> "CLASS"
455 |     CONSTRUCTOR    -> "CONSTRUCTOR"
456 |     PUBLIC         -> "PUBLIC"
457 |     PRIVATE        -> "PRIVATE"
458 |     EXTENDS        -> "EXTENDS"
459 |     INCLUDE        -> "INCLUDE"
460 |     THIS           -> "THIS"
461 |     BREAK          -> "BREAK"
462 |     CONTINUE       -> "CONTINUE"
463 |     NEW            -> "NEW"
464 |     INT_LITERAL(i) -> "INT_LITERAL(" ^ string_of_int i ^ "
465 |     " )"
465 |     ↵ FLOAT_LITERAL(f) -> "FLOAT_LITERAL(" ^ string_of_float
465 |     f ^ ")"
466 |     ↵ CHAR_LITERAL(c) -> "CHAR_LITERAL(" ^ Char.escaped c
466 |     ^ ")"
467 |     STRING_LITERAL(s) -> "STRING_LITERAL(" ^ s ^ ")"
468 |     ID(s)          -> "ID(" ^ s ^ ")"
469 |     DELETE          -> "DELETE"
470 |     MODULO          -> "MODULO"
471 |     EOF             -> "EOF"

472
473 let string_of_token_no_id = function
474 | LPAREN          -> "LPAREN"
475 | RPAREN          -> "RPAREN"
476 | LBRACE          -> "LBRACE"
477 | RBRACE          -> "RBRACE"
478 | SEMI            -> "SEMI"
479 | COMMA           -> "COMMA"
480 | PLUS             -> "PLUS"
481 | MINUS           -> "MINUS"
482 | TIMES            -> "TIMES"
483 | DIVIDE          -> "DIVIDE"
484 | ASSIGN           -> "ASSIGN"
485 | EQ               -> "EQ"
486 | NEQ              -> "NEQ"
487 | LT               -> "LT"
488 | LEQ              -> "LEQ"
489 | GT               -> "GT"
490 | GEQ              -> "GEQ"
491 | AND              -> "AND"
492 | OR               -> "OR"
493 | NOT              -> "NOT"

```

```

494 |     DOT          -> "DOT"
495 |     LBRACKET    -> "LBRACKET"
496 |     RBRACKET    -> "RBRACKET"
497 |     BAR          -> "BAR"
498 |     IF           -> "IF"
499 |     ELSE          -> "ELSE"
500 |     FOR          -> "FOR"
501 |     WHILE         -> "WHILE"
502 |     RETURN        -> "RETURN"
503 |     INT           -> "INT"
504 |     FLOAT          -> "FLOAT"
505 |     BOOL          -> "BOOL"
506 |     CHAR          -> "CHAR"
507 |     VOID          -> "VOID"
508 |     NULL          -> "NULL"
509 |     TRUE          -> "TRUE"
510 |     FALSE         -> "FALSE"
511 |     CLASS         -> "CLASS"
512 |     CONSTRUCTOR   -> "CONSTRUCTOR"
513 |     PUBLIC        -> "PUBLIC"
514 |     PRIVATE       -> "PRIVATE"
515 |     EXTENDS      -> "EXTENDS"
516 |     INCLUDE       -> "INCLUDE"
517 |     THIS          -> "THIS"
518 |     BREAK         -> "BREAK"
519 |     CONTINUE      -> "CONTINUE"
520 |     NEW           -> "NEW"
521 |     INT_LITERAL(i) -> "INT_LITERAL"
522 |     FLOAT_LITERAL(f) -> "FLOAT_LITERAL"
523 |     CHAR_LITERAL(c) -> "CHAR_LITERAL"
524 |     STRING_LITERAL(s) -> "STRING_LITERAL"
525 |     ID(s)         -> "ID"
526 |     DELETE        -> "DELETE"
527 |     MODULO        -> "MODULO"
528 |     EOF            -> "EOF"
529
530 let token_list_to_string_endl token_list =
531 let rec helper last_line_number = function
532 (token, curr)::tail ->
533 let line = curr.lineno in
534 (if line != last_line_number then "\n" ^ string_of_int line
535   ~^ ". " else " ") ^
536 string_of_token token ^ helper line tail
537 | [] -> "\n"
538 in helper 0 token_list
539
540 let token_list_to_string token_list =
541 let rec helper = function
542 (token, line)::tail ->
543 string_of_token_no_id token ^ " " ^ helper tail
544 | [] -> "\n"
545 in helper token_list

```

processor.ml

```
1  open Parser
2  (* open Exceptions *)
3
4  let last_token = ref EOF
5
6  (* Gets the original raw tokens from the scanner *)
7  let get_tokens lexbuf =
8    let rec next lexbuf token_list =
9      match Scanner.token lexbuf with
10      | DEDENT_EOF(c) as eof -> eof :: token_list
11      | _ as token -> token :: (next lexbuf token_list)
12  in next lexbuf []
13
14 (* Replaces DEDENT_COUNT with DEDENTS *)
15 let rec get_tokens_with_dedents original_token_list
16   ~new_token_list=
17   let rec fill_dedent count mylist =
18     if count <= 0 then mylist
19     else
20       fill_dedent (count-1)
21   ~(List.rev(DEDENT::List.rev(mylist)))
22   in
23   if (List.length(original_token_list)) != 0 then
24     match (List.hd original_token_list) with
25     | DEDENT_COUNT(c) ->
26       let templ = (List.rev (TERMINATOR::(List.rev
27   ~new_token_list))) in
28       let temp = fill_dedent c templ in
29       get_tokens_with_dedents (List.tl
30   ~original_token_list) temp
31     | DEDENT_EOF(c) ->
32       let templ = (List.rev (TERMINATOR::(List.rev
33   ~new_token_list))) in
34       let temp = fill_dedent c templ in
35       List.rev(EOF::(List.rev temp));
36       | _ as token -> get_tokens_with_dedents (List.tl
37   ~original_token_list) (List.rev (token :: (List.rev
38   ~new_token_list)))
39   else
40     new_token_list
41
42 (* Removes opening TERMINATOR if it is there *)
43 let filter_opening_whitespace token_list =
44   match token_list with
45   | [] -> []
46   | hd::tail -> if (hd = TERMINATOR) then tail else
47   ~token_list
48
49 (* Function that uses above three functions to get the
50   ~ final list of tokens *)
```

```

42 let get_token_list lexbuf =
43   let original_token_list = get_tokens lexbuf in
44   let new_token_list = get_tokens_with_dedents
45     ↳ original_token_list [] in
46   let filtered_token_list = filter_opening_whitespace
47     ↳ new_token_list
48 in filtered_token_list
49
50 (* Parse function *)
51 let parser token_list =
52   let token_list = ref(token_list) in
53   let tokenizer _ =
54     match !token_list with
55       | head :: tail ->
56         last_token := head;
57         token_list := tail;
58         head
59       | [] -> raise (Exceptions.Missing_eof) in
60 let program = Parser.program tokenizer
61   ↳ (Lexing.from_string "") in
62 program

```

Library files

vlc.hpp

```

1 #ifndef VLC_H
2 #define VLC_H
3
4
5 // Defines the default block and grid size
6 #ifndef BLOCK_SIZE
7 #define BLOCK_SIZE 1024
8 #endif
9
10 #ifndef GRID_SIZE
11 #define GRID_SIZE 32
12 #endif
13 // Include statements
14 #include <stdlib.h>
15 #include <iostream>
16 #include <stdarg.h>
17
18 // Useful Macros for CUDA
19 // #define min(a, b) (((a) < (b)) ? (a) : (b))
20 // #define max(a, b) (((a) > (b)) ? (a) : (b))
21
22 // CUDA Error checking function
23 // void checkCudaErrors(CUresult err) {
24 //   assert(err == CUDA_SUCCESS);

```

```

25 // }
26
27 /* Why this class exists:
28 - For ensuring that we don't have any arrays allocated on
29   the stack and all are allocated on the heap
30   ( can get messy with memory otherwise )
31 - To bypass C/C++ not being able to do things like the
32   following assignment
33     size_t a[5];
34     size_t b[5] = {1,2,3,4,5};
35     a=b;
36
37           !!size_t[5] not assignable error!!
38 */
39
40 // VLC Array class
41 template <class T>
42 class VLC_Array {
43 private:
44     size_t num_values; // Tells us how many values are in
45   → the array in total. Ex. would be 4 if [2][2] array
46     T* values; // Posize_ter to values in array
47
48 public:
49     // Constructors and Destructors
50     VLC_Array();
51     VLC_Array(size_t num_values, size_t
52       → num_dimensions,...);
53     VLC_Array(size_t num_values, size_t
54       → num_dimensions, size_t total_args...);
55     VLC_Array(size_t num_values, T*values, size_t
56       → num_dimensions, size_t
57       → *dimensions);                                     // For
58       → declarations
59       → // For declarations and initializations like size_t
60       → a[5] = {1,2,3,4,5}
61     VLC_Array(const VLC_Array<T> &vlcarray); // For
62       → assignments like size_t a[1] = {5}, size_t b[1]={7},
63       → a=b
64     ~VLC_Array();
65
66     /* Class Accessors and Getters */
67     T*      get_values() const; // Returns the posize_ter
68   → to VLC's size_ternal array
69     size_t* get_dimensions() const; // Returns the
70   → posize_ter to VLC's dimensions
71     size_t  get_num_dimensions () const; // Returns number
72   → of dimensions
73     size_t  size() const; // Returns length of first
74   → dimension

```

```

62     size_t total_elements() const; // Returns total
→   elements in the array
63
64     /* Element Accessors and Getters */
65     T get_element_value(size_t number_accessing_dims, ... )
→   const;
66     VLC_Array<T> get_array_value_host(size_t
→   number_accessing_dims, ... ) const;
67     T* get_array_value_kernel(size_t
→   number_accessing_dims, ... ) const;
68     void set_element_value(T new_value, size_t
→   number_accessing_dims, ... );
69     void set_array_value(const VLC_Array<T> &array, size_t
→   number_accessing_dims, ... );
70     VLC_Array<T> operator=(const VLC_Array<T> &vlcarray);
71
72 };
73
74 ----- Regular constructors -----*/
75 template <class T>
76 VLC_Array<T>::VLC_Array() {
77     this->num_values = 0;
78     this->values = NULL;
79     this->num_dimensions = 0;
80     this->dimensions = NULL;
81 }
82
83
84 template <class T>
85 VLC_Array<T>::VLC_Array(size_t num_values, T*values, size_t
→   num_dimensions, size_t *dimensions) {
86     this->num_values = num_values;
87     this->num_dimensions = num_dimensions;
88
89     T *values_copy = (T*)calloc(num_values, sizeof(T));
90     for(size_t i = 0; i < num_values; i++) {
91         values_copy[i] = values[i];
92     }
93
94     size_t *dims_copy = (size_t*)calloc(
→   num_dimensions, sizeof(size_t));
95     for(size_t j = 0; j < num_dimensions; j++) {
96         dims_copy[j] = dimensions[j];
97     }
98
99     this->values = values_copy;
100    this->dimensions = dims_copy;
101 }
102
103 //Declarations
104 template <class T>
105 VLC_Array<T>::VLC_Array(size_t num_values, size_t
→   num_dimensions, ... ) {

```

```

106  /* Assign the dimensions and values */
107  this->num_dimensions = num_dimensions;
108  this->num_values = num_values;
109
110  this->dimensions = (size_t *)calloc(
111      num_dimensions, sizeof(size_t));
112  this->values = (T*)calloc( num_values, sizeof(T));
113
114  /* Now access the values that are passed in */
115  std::cout<<num_dimensions<<std::endl;
116  va_list args;
117  va_start(args, num_dimensions);
118  for(size_t i = 0; i < num_dimensions; i++) {
119      this->dimensions[i] = va_arg(args, size_t);
120  }
121  va_end(args);
122
123 // Declarations, Assignments by value
124 template <class T>
125 VLC_Array<T>::VLC_Array(size_t num_values, size_t
126     num_dimensions, size_t total_args...) {
127     /* Assign the dimensions and values */
128     this->num_dimensions = num_dimensions;
129     this->num_values = num_values;
130
131     this->dimensions = (size_t
132         *)calloc(num_dimensions, sizeof(size_t));
133     this->values = (T*)calloc( num_values, sizeof(T));
134
135     /* Now access the values that are passed in */
136     va_list args;
137     va_start(args, total_args);
138     for(size_t i = 0; i < num_dimensions; i++) {
139         this->dimensions[i] = va_arg(args, size_t);
140         for(size_t j = 0; j < num_values;
141             j++) { this->values[j] = va_arg(args, T); }
142     }
143     va_end(args);
144
145 // Assignments to other arrays
146 template <class T>
147 VLC_Array<T>::VLC_Array(const VLC_Array<T> &vlcarray) {
148     /* For now, make a deep copy every time. Can optimize
149     later */
150     this->num_values = vlcarray.total_elements();
151     this->num_dimensions = vlcarray.get_num_dimensions();
152
153     this->values = (T *)calloc(this->num_values, sizeof(T));
154     this->dimensions = (size_t *)calloc(
155         this->num_dimensions, sizeof(size_t));
156
157     /* Now access the values that are passed in */

```

```

150    for(size_t j = 0; j < this->num_values; j++) {
151        this->values[j] = vlcarray.get_values() [j];
152    }
153
154    // Destructor
155    template <class T>
156    VLC_Array<T>::~VLC_Array() {
157        free(this->values);
158        free(this->dimensions);
159    }
160
161    /*----- Accessing Functions
162    -----*/
163
164    // Get Element Value
165    // Accesses element of the array - must check num_accessing
166    // = num_dims in semant
167    template <class T>
168    T VLC_Array<T>::get_element_value(size_t
169        number_accessing_dims,...) const{
170        size_t index = 1;
171        size_t corr_dim;
172        printf("num_access_dim%zu\n",number_accessing_dims );
173        va_list dims;
174        va_start(dims,number_accessing_dims);
175        for(size_t i=0; i < number_accessing_dims;i ++){
176            index = va_arg(dims,size_t) * index;
177            printf("index right now is %zu\n",index);
178            corr_dim = this-> dimensions[i];
179            printf("dim right now is%zu\n",corr_dim);
180            index = i * corr_dim + index;
181        }
182        printf("%zu\n",index);
183        va_end(dims);
184        return this->values[index];
185    }
186
187    // Get Array Value In Host
188    // Accesses an array of the array - must check
189    // num_accessing < num_dims in semant
190    template <class T>
191    VLC_Array<T> VLC_Array<T>::get_array_value_host(size_t
192        number_accessing_dims,...) const{
193        // Get where new array starts
194        size_t index = 1;
195        size_t corr_dim;
196        va_list dims;
197        va_start(dims,number_accessing_dims);
198        for(size_t i=0; i < number_accessing_dims; i++){
199            index = va_arg(dims,size_t) * index;
200            corr_dim = this-> dimensions[i];

```

```

195     index = i * corr_dim + index;
196 }
197 va_end(dims);
198
199 // Get all the elements in this new array
200 size_t num_elements = 1;
201 for(size_t i = this->num_dimensions -
202     number_accessing_dims; i < this->num_dimensions;i++) {
203     num_elements = num_elements * this->dimensions[i];
204 }
205
206 // Set values
207 size_t num_dimensions = this->num_dimensions -
208     number_accessing_dims;
209 size_t *new_dimensions =
210     &(this->dimensions[this->num_dimensions -
211     number_accessing_dims]);
212 size_t num_values = num_elements;
213 size_t *new_values = this->values[index];
214
215 // Return a VLC_Array
216 return
217     VLC_Array(num_values,new_values,num_dimensions,new_dimensions);
218 }
219
220 // Get Array Value In Kernel
221 // Accesses an array of the array - must check
222 //    num_accessing < num_dims in semant
223 template <class T>
224 T* VLC_Array<T>::get_array_value_kernel(size_t
225     number_accessing_dims,...) const{
226     // Get where new array starts
227     size_t index = 1;
228     size_t corr_dim;
229     va_list dims;
230     va_start(dims,number_accessing_dims);
231     for(size_t i=0; i < number_accessing_dims; i++) {
232         index = va_arg(dims,size_t) * index;
233         corr_dim = this-> dimensions[i];
234         index = i * corr_dim + index;
235     }
236     va_end(dims);
237
238     // Get all the elements in this new array
239     size_t num_elements = 1;
240     for(size_t i = this->num_dimensions -
241         number_accessing_dims; i < this->num_dimensions;i++) {
242         num_elements = num_elements * this->dimensions[i];
243     }
244
245     // Set values
246     size_t num_dimensions = this->num_dimensions -
247         number_accessing_dims;

```

```

240     size_t *new_dimensions =
241         &(this->dimensions[this->num_dimensions -
242             number_accessing_dims]);
243     size_t num_values = num_elements;
244     size_t *new_values = this->values[index];
245
246     // Return a VLC_Array
247     return
248         VLC_Array(num_values, new_values, num_dimensions, new_dimensions);
249
250     }
251
252     // Set Element Value
253     // Sets value for element of an array
254     template <class T>
255     void VLC_Array<T>::set_element_value(T new_value, size_t
256         number_accessing_dims, ...) {
257         // Get where new array starts
258         size_t index = 1;
259         size_t corr_dim;
260         va_list dims;
261         va_start(dims, number_accessing_dims);
262         for(size_t i=0; i < number_accessing_dims; i++) {
263             index = va_arg(dims, size_t) * index;
264             corr_dim = this->dimensions[i];
265             index = i * corr_dim + index;
266         }
267         va_end(dims);
268         printf("new value is %d\n", new_value);
269         this->get_values() [index] = new_value;
270     }
271
272     // Set Array Value
273     // Sets value for an array of an array
274     template <class T>
275     void VLC_Array<T>::set_array_value(const VLC_Array<T>
276         &array, size_t number_accessing_dims, ...) {
277         // Get where new array starts
278         size_t index = 1;
279         size_t corr_dim;
280         va_list dims;
281         va_start(dims, number_accessing_dims);
282         for(size_t i=0; i < number_accessing_dims; i++) {
283             index = va_arg(dims, size_t) * index;
284             corr_dim = this->dimensions[i];
285             index = i * corr_dim + index;
286         }
287         va_end(dims);
288
289         //Get number of elements to replace
290         size_t num_elements = 1;
291         for(size_t i = this->num_dimensions -
292             number_accessing_dims; i < this->num_dimensions; i++) {
293             num_elements = num_elements * this->dimensions[i];

```

```

288     }
289     // Copy values
290     for(size_t i = 0; i < num_elements; i++) {
291         this->values[int(index + i)] =
292             array.get_element_value(1, i);
293     }
294     //Operator =
295     template <class T>
296     VLC_Array<T> VLC_Array<T>::operator=(const VLC_Array<T>
297         &vlcarray) {
298         if(this == &vlcarray) {
299             return *this;
300         }
301         /* For now, make a deep copy every time. Can optimize
302            later */
303         num_values = vlcarray.total_elements();
304         num_dimensions = vlcarray.get_num_dimensions();
305
306         values = (T*)calloc(sizeof(T) *
307             vlcarray.total_elements());
308         dimensions = (size_t *)calloc(sizeof(size_t) *
309             vlcarray.get_num_dimensions());
310
311         /* Now access the values that are passed in */
312         for(size_t j = 0; j < this->num_values; j++) {
313             this->values[int(j)] =
314                 vlcarray.get_values()[int(j)];
315             for(size_t i = 0; i < this->num_dimensions; i++) {
316                 this->dimensions[int(i)] =
317                     vlcarray.get_dimensions()[int(i)];
318             }
319         }
320         return *this;
321     }
322
323     template <class T>
324     size_t VLC_Array<T>::get_values() const { return this->values; }
325
326     template <class T>
327     size_t* VLC_Array<T>::get_dimensions() const { return
328         this->dimensions; }
329
330     template <class T>
331     size_t VLC_Array<T>::get_num_dimensions() const { return
332         this->num_dimensions; }
333
334     template <class T>
335     size_t VLC_Array<T>::size() const { return
336         this->dimensions[0]; }
337
338     template <class T>
339     size_t VLC_Array<T>::total_elements() const { return
340         this->num_values; }
341
342 #endif

```

Tests

test.sh

```
1  #!/bin/bash
2  (set -o igncr) 2>/dev/null && set -o igncr; # this comment
   ↳ is required
3
4  # Regression testing script for VLC
5  # Step through a list of files
6  # Compile, run, and check the output of each
   ↳ expected-to-work test
7  # Compile and check the error of each expected-to-fail
   ↳ test
8
9  NVCC="nvcc"
10
11 VLC="sudo vlc -c"
12
13 globallog=./tests/test.log
14 rm -f $globallog
15 error=0
16 globalerror=0
17 NC='\033[0m'
18 GREEN='\033[0;32m'
19 CYAN='\033[0;36m'
20 keep=0
21 pass=0
22 fail=0
23 Usage() {
24     echo "Usage: test.sh [options] [.mc files]"
25     echo "-k      Keep intermediate files"
26     echo "-h      Print this help"
27     exit 1
28 }
29
30 SignalError() {
31     if [ $error -eq 0 ] ; then
32         echo "FAILED"
33         error=1
34     fi
35     echo " $1"
36 }
37
38 # Compare <outfile> <reffile> <difffile>
39 # Compares the outfile with reffile. Differences, if any,
   ↳ written to difffile
40 Compare() {
41     generatedfiles="$generatedfiles $3"
42     echo diff -b $1 $2 ">" $3 1>&2
43     diff -b "$1" "$2" > "$3" 2>&1 || {
```

```

44     SignalError "$1 differs"
45     echo "FAILED $1 differs from $2" 1>&2
46   }
47 }
48
49 # Run <args>
50 # Report the command, run it, and report any errors
51 Run() {
52   echo $* 1>&2
53   eval $* || {
54     SignalError "$1 failed on $*"
55     return 1
56   }
57 }
58
59 # RunFail <args>
60 # Report the command, run it, and expect an error
61 RunFail() {
62   echo $* 1>&2
63   eval $* && {
64     SignalError "failed: $* did not report an error"
65     return 1
66   }
67   return 0
68 }
69
70 Check() {
71   error=0
72   basename=`echo $1 | sed 's/.*\\\///'
73   `           s/.vlc//` `
74   reffile=`echo $1 | sed 's/.vlc$//` `
75   basedir="`echo $1 | sed 's/\\/[^\//]*$//` `."
76
77   echo -n "$basename..."
78
79   echo 1>&2
80   echo ##### Testing $basename " 1>&2
81
82   generatedfiles=""
83
84   generatedfiles="$generatedfiles ./tests/${basename}.cu
85   ./tests/${basename}.out ./tests/${basename}" &&
86   Run "$VLC" $1 > null" &&
87   Run "$NVCC" "./tests/${basename}.cu -o
88   ./tests/${basename} && ./tests/${basename}" ">""
89   "./tests/${basename}.out" &&
90   Compare ./tests/${basename}.out ./${reffile}.out
91   ./tests/${basename}.diff
92
93   if [ $error -eq 0 ] ; then
94     if [ $keep -eq 0 ] ; then
95       rm -f $generatedfiles
96     fi
97     echo "OK"

```

```
145      esac
146 done
147
148 shift `expr $OPTIND - 1`
149
150 if [ $# -ge 1 ]
151 then
152     files=$@
153 else
154     files="tests/test-*.vlc tests/fail-*.vlc"
155 fi
156
157 for file in $files
158 do
159     case $file in
160         *test-*)
161             Check $file 2>> $globallog
162             ;;
163         *fail-*)
164             CheckFail $file 2>> $globallog
165             ;;
166         *)
167             echo "unknown file type $file"
168             globalerror=1
169             ;;
170     esac
171 done
172 echo ""
173 echo -e "Tests Passed: $pass"
174 echo -e "Tests Failed: $fail"
175 exit $globalerror
```

test-arithmetic_ops.vlc

```
1 int def vlc():
2     int a = 2
3     int b = 4
4
5     int c = a + b
6     int d = b - a
7     int e = a * b
8     int f = b / a
9     int g = b % a
10
11    print("success")
12    return 0
```

test-arithmetic_ops.cu

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "cuda.h"
4 #include <iostream>
5 #include "vlc.hpp"
6 #include <stdarg.h>
7 CUdevice device;
8 CUmodule cudaModule;
9 CUcontext context;
10 CUfunction function;
11 int vlc() {
12     int a=2;
13     int b=4;
14     int c=a + b;
15     int d=b - a;
16     int e=a * b;
17     int f=b / a;
18     int g=b % a;
19     printf("success");
20     return 0;
21 }
22
23
24 int main(void) { return vlc(); }
```

test-print_hello_world.vlc

```
1 string helloworld
2
3 int def vlc():
4     helloworld = "Hello world!"
5     print(helloworld)
6     return 0
```

test-print_hello_world.cu

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "cuda.h"
4 #include <iostream>
5 #include "vlc.hpp"
6 #include <stdarg.h>
7 CUdevice device;
8 CUmodule cudaModule;
9 CUcontext context;
10 CUfunction function;
11 char * helloworld;
12 int vlc(){
13     helloworld="Hello world!";
14     printf(helloworld);
15     return 0;
16 }
17
18 int main(void) { return vlc(); }
```

test-statements.vlc

```
1 int defg add(int x, int y):
2     return scale * (x + y)
3
4 int defg vector_add(int a, int b):
5     int index = a
6     if(index == 1):
7         index = 5
8
9     for (int i = 0, i < 2, i = i + 1):
10        print(i)
11
12    while(i < 3):
13        print(i)
14
15    index = 5 if (i == 2) else 2
16    a[4]
17
18
19
20 int def vlc():
21     int[5] a = {1,2,3,4,5}
22     int[5] b = {1,2,3,4,5}
23
24     int[5] c = ~map(add, consts(scale = 4), a, b)
25     int[5] d = vector_add(a, b)
26
27     print(d)
28
29     return 0
```

test-statements.cu

1

fail-Already_declared.vlc

```
1 int a = 5
2 int a = 6
3
4 int def main():
5     return 1
```

fail-Already_declared.vlc.err

```
1 Fatal error: exception Failure("int_of_string")
```

fail-Array_elements_not_all_same_type.vlc

```
1 int[5] a = [1.0, 1]
2
3 int def main():
4     return 1
```

fail-Array_elements_not_all_same_type.vlc.err

```
1 Fatal error: exception Failure("int_of_string")
```

fail-bad_array_initialization.vlc

```
1 /*Number of elements in array does not match specified
   ↵ size*/
2 int [5] a = [1]
3
4 int def vlc():
5     return 0
```

fail-bad_array_initialization.vlc.err

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

fail-bad_return_type.vlc

```
1 /*Return statement doesn't match type*/
2 int defg add(int x, int y):
3     float index = 1.0
4     return index
5
6 int def vlc():
7     return 0
```

fail-bad_return_type.vlc.err

```
1 Fatal error: exception Failure("int_of_string")
```

fail-Boolean_condition.vlc

```
1 int def main():
2     float a = 1.0
3     if(a):
4         print("hi")
5     return 1
```

fail-Boolean_condition.vlc.err

```
1 Fatal error: Conditional_must_be_a_boolean
```

fail-Cannot_perform_operation_on_array.vlc

```
1 int def main():
2     int[5] a = [1, 2, 3, 4, 5]
3     int c = a + 3
4     return 1
```

fail-Cannot_perform_operation_on_array.vlc.err

```
1 Fatal error: Cannot_perform_operation_on_array
```

fail-Cannot_perform_operation_on_string.vlc

```
1 int def main():
2     string hi = "hello"
3     if(█hi):
4         print("wrong")
5     return 1
```

fail-Cannot_perform_operation_on_string.vlc.err

```
1 Fatal error: exception Failure("int_of_string")
```

fail-Constants_missing_in_defg.vlc

```
1 int defg test(int a):
2     scale = 3
3     return a
4
5 int def main():
6     int[5] a = [1, 2, 3, 4, 5]
7     int[5] b = ~map(test, a)
```

fail-Constants_missing_in_defg.vlc.err

```
1 Fatal error: Constants_missing_in_defg
```

fail-defg_reinitialize.vlc

```
1 /*Constant input re-initialized in defg*/
2 int defg add(int x, int y):
3     int scale = 5
4     return scale * (x + y)
5
6 int def vlc():
7     int[5] a = {1,2,3,4,5}
8     int[5] b = {1,2,3,4,5}
9     int[5] c = ~map(add, consts(scale = 4), a, b)
10
11    return 0
```

fail-defg_reinitialize.vlc.err

```
1 Fatal error: exception Exceptions.Already_declared
```

fail-Empty_array_access.vlc

```
1 int def main():
2     int[5] a
3     a[3] = 6
4     return 1
```

fail-Empty_array_access.vlc.err

```
1 Fatal error: exception Exceptions.Empty_array_access
```

fail-Function_already_declared.vlc

```
1 int def one(int i):  
2     return 1  
3  
4 int def one(int i):  
5     return 1  
6  
7 int def main():  
8     return 1
```

fail-Function_already_declared.vlc.err

```
1 Fatal error: exception Exceptions.Function_already_declared
```

fail-Function_not_defined.vlc

```
1 int def main():
2     int b = 5
3     float c = 5.0
4     b = c
5     return 1
```

fail-Function_not_defined.vlc.err

```
1 Fatal error: Function_not_defined
```

fail-Have_statements_after_break.vlc

```
1 int def main():
2     int i
3     for (i = 0, i < 6, i++):
4         break
5     int a = 5
6     return 1
```

fail-Have_statements_after_break.vlc.err

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

fail-Have_statements_after_return.vlc

```
1 int def main():
2     return 1
3     int a = 5
```

fail-Have_statements_after_return.vlc.err

```
1 Fatal error: exception
   ~ Exceptions.Have_statements_after_return_break_continue
```

fail-Higher_order_function_only_takes_defg.vlc

```
1 int def test(int b):
2     return 1
3
4 int def main():
5     int[3] a = [0, 0, 0]
6     int[3] c = ~map(test, a)
7     return 1
```

fail-Higher_order_function_only_takes_defg.vlc.err

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

fail-Invalid_accessor_value.vlc

```
1 int def main():
2     int[1] a = [0]
3     a[2]
4     return 1
```

fail-Invalid_accessor_value.vlc.err

```
1 Fatal error: exception Exceptions.Invalid_accessor_value
```

fail-multidec.vlc

```
1 /*Multiple declarations of the same variable*/
2 int a = 5
3 int a = 6
4
5 int def vlc():
6     return 0
```

fail-multidec.vlc.err

```
1 Fatal error: exception Exceptions.Variable_already_declared
```

fail-Name_not_found.vlc

```
1 a = 5
2
3 int def main()
4     return 1
```

fail-Name_not_found.vlc.err

```
1 Fatal error: exception Exceptions.Name_not_found
```

fail-nomain.vlc

```
1 /*No main*/
2 int defg test():
3     return 0
```

fail-nomain.vlc.err

```
1 Fatal error: exception Exceptions.No_main_function
```

fail-No_strings_allowed_in_gdecl.vlc

```
1 int defg test(string a):  
2     return a  
3  
4 int def main():  
5     return 1
```

fail-No_strings_allowed_in_gdecl.vlc.err

```
1 Fatal error: exception  
    ~ Exceptions.NO_STRINGS_ALLOWED_IN_GDECL
```

fail-nomain.vlc

```
1 /*No main*/
2 int defg test():
3     return 0
```

fail-nomain.vlc.err

```
1 Fatal error: exception Exceptions.No_main_function
```

fail-Nonarray_passed_into_gdecl.vlc

```
1 int defg test(int i):
2     return 1
3
4 int def main():
5     int a = a
6     int c = ~map(a)
7     return 1
```

fail-Nonarray_passed_into_gdecl.vlc.err

```
1 Fatal error:
   → Nonarray_argument_passed_into_higher_order_function
```

fail-nonconstdefg.vlc

```
1 /*Constant inputs to defg not constant*/
2 int defg add(int x, int y):
3     scale = 5
4     return scale * (x + y)
5
6 int def vlc():
7     return 0
```

fail-nonconstdefg.vlc.err

```
1 Fatal error: exception Exceptions.Non_constant_constants
```

fail-Out_of_scope.vlc

```
1 /*Scope fails*/
2 int def vlc():
3     for (int i = 0, i < 2, i = i + 1):
4         int k = 1
5         k = 8
```

fail-Out_of_scope.vlc.err

```
1 Fatal error: exception
   ~ Exceptions.Variable_not_found_in_scope
```

fail-predefined_defg.vlc

```
1 /*Print function in defg*/
2 int defg vector_add(int a, int b):
3     print(a)
4     return a
5
6 int def vlc():
7     return 0
```

fail-predefined_defg.vlc.err

```
1 Fatal error: Invalid_function_in_defg
```

fail-Type_mismatch.vlc

```
1 int def main():
2     int b = 5
3     float c = 5.0
4     b = c
5     return 1
```

fail-Type_mismatch.vlc.err

```
1 Fatal error: exception Failure("int_of_string")
```

fail-undefined_defg.vlc

```
1 /*Defg undefined*/
2 int def vlc():
3     int[5] a = {1,2,3,4,5}
4     int[5] b = {1,2,3,4,5}
5     int[5] c = ~map(add, consts(scale = 4), a, b)
6
7     return 0
```

fail-undefined_defg.vlc.err

```
1 Fatal error: exception
   ~ Exceptions.Function_not_defined("add")
```

fail-uninitialized_variable.vlc

```
1 /*Uninitialized variable*/
2 helloworld = "Hello world!"
```

fail-uninitialized_variable.vlc.err

```
1 Fatal error: exception
  ~ Exceptions.Name_not_found("helloworld")
```

fail-unmatching_args.vlc

```
1 /*PTX arguments do not match*/
2 int defg add(float x, int y):
3     return (x + y)
4
5 int def vlc():
6     return 0
```

fail-unmatching_args.vlc.err

```
1 Fatal error: exception Exceptions.Unmatching_PTX_args
```

fail-unsupported_defg_args.vlc

```
1 /*Unsupported PTX type as argument*/
2 string defg add(string x, string y):
3     return x
4
5 int def vlc():
6     return 0
```

fail-unsupported_defg_args.vlc.err

```
1 Fatal error: exception
   ↳ Exceptions.NO_STRINGS_ALLOWED_IN_GDECL
```

fail-Variable_not_declared.vlc

```
1 b = 5
2
3 int def main():
4     return 1
```

fail-Variable_not_declared.vlc.err

```
1 Fatal error: exception Exceptions.Name_not_found("b")
```

fail-Variable_not_found_in_scope.vlc

```
1 int def main():
2     int i
3     for (i=0, i<5, i++):
4         int j = j + i
5
6     return j
```

fail-Variable_not_found_in_scope.vlc.err

```
1 Fatal error: exception Exceptions.Name_not_found("a")
```

fail-Void_type_in_gdecl.vlc

```
1 int defg test(string a):  
2     return a  
3  
4 int def main():  
5     return 1
```

fail-Void_type_in_gdecl.vlc.err

```
1 Fatal error: exception Exceptions.Void_type_in_gdecl
```

fail-wrong_array_type2.vlc

```
1 /*Array type not supported*/
2 string[2] wrong_array = ["hi", "hello"]
3
4 int def vlc():
5     return 0
```

fail-wrong_array_type2.vlc.err

```
1 Fatal error: exception Exceptions.Array_type_not_supported
```

fail-wrong_array_types.vlc

```
1 /*Different types in array declaration*/
2 int[2] wrong_array = [1.0, 1]
3
4 int def vlc():
5     return 0
```

fail-wrong_array_types.vlc.err

```
1 Fatal error: exception
   → Exceptions.Array_elements_not_all_same_type
```