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 here[1].
If you do not have Homebrew, install it by running the script:

```
$ /user/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 here[2]

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. Paremeters 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'-'z' 'A'-'Z' '_' ] ['a'-'z' 'A'-'Z' '_'
↪   '1'-'9']*
```

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, multilplication, 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 evalutates 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, greather-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 *ith* 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 boggled 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 *arg*1 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 fplit
 ↪    { FLOATING_POINT_LITERAL(float_of_string fplit) }
  | (letter | '_')(letter | digit | '_')* as id {
 ↪  IDENTIFIER(id) }
  | eof       { get_eof() }

(* Blocks for comments *)
and single_line_comment = parse
```

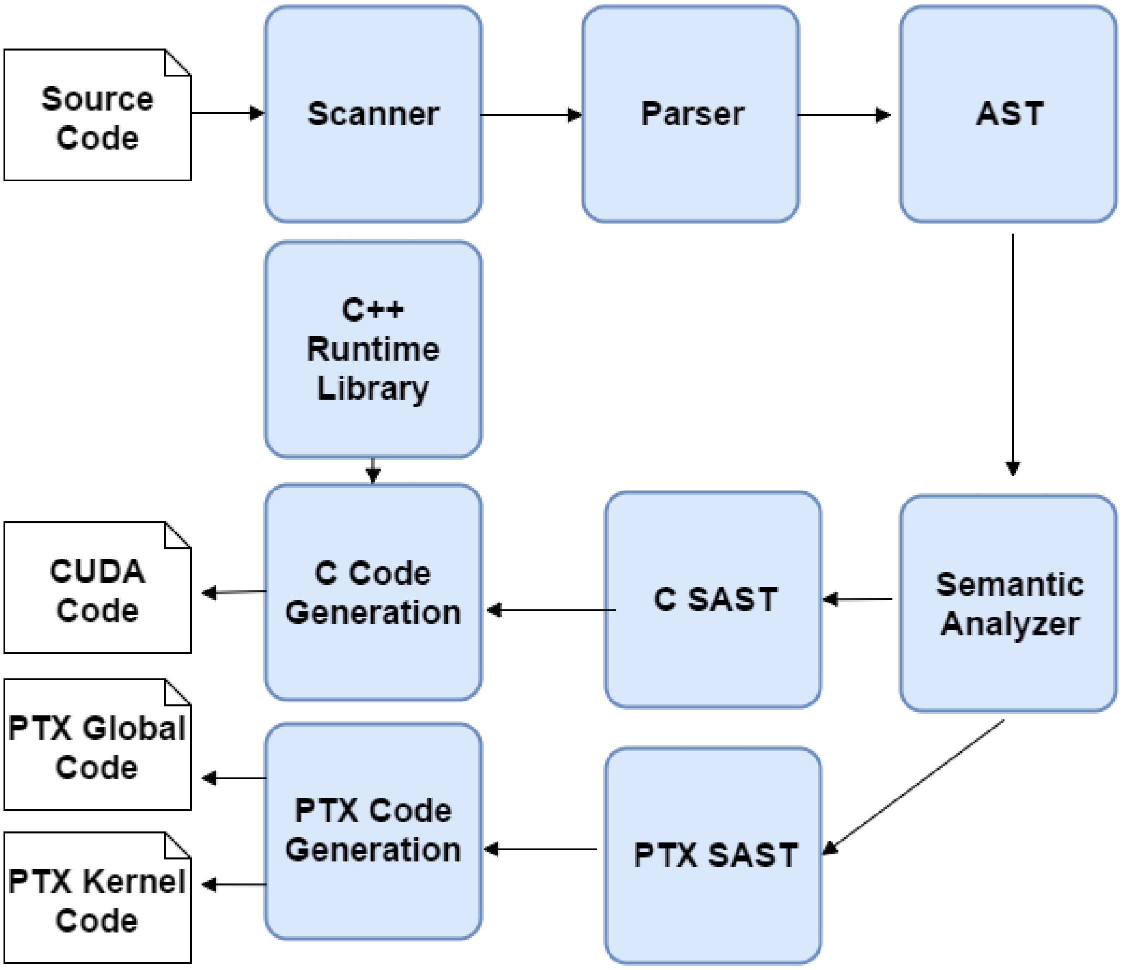
```
  | newline                { indent lexbuf }
  | eof                  { get_eof() }
  | _                  { single_line_comment lexbuf }

and multi_line_comment = parse
  | newline                { multi_line_comment lexbuf }
  | "*/"                 { token 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 DEDENT_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 featuares 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 <krl2130@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 <krl2130@columbia.edu>
Date:   Fri May 13 08:02:33 2016 -0400

    final

commit 4c38175a8ce46539d6065308348d4daeb7d5d5c4
Author: Kellie Ren Lu <krl2130@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 <krl2130@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 864559b1f978922b8061669bfcbe1c9c8fd1cfdf
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 a4e89135ce301eb643820501962b8c7dcd989c16
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 4ff0c9ee0ad4294f8f0c6cdbfd04486dccd0decd
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 initializatiosn 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 55242b5d5917345e3a78cf766f0edd11f006b1ed
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 assignm

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 919fcd8ea51cbccab037597bc08a93a35eb58273
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 9c79ccb3dbe642a8bcbc7d6a3e554eda04d7b6ab
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 8f5ee0aabd0bf4dc284bac534aafdaedf381212e
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 <krl2130@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 <krl2130@columbia.edu>
Date:   Mon May 9 17:03:36 2016 -0400

    vlc_array finished

commit 443fb4c1d4a129043acdfb9ff34b003c5572b563
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Mon May 9 15:47:10 2016 -0400

    changed vlc array

commit 4e89fb40123f8ec86f438761c78068e044bc65d1
Author: Kellie Ren Lu <krl2130@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 b041393cefc9ca819f5e073ddad6b080a5ff8b44
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 d7481e19e70c16fc0593c907cde764aefe729946
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 cf800b4c15215bbef091cee119d4d9ef15c15291
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 a2fcb3102d350986e5bc51799db56ac90f60319c
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 955c6490db401f842f7f05498fda13cb831c7692
Merge: 036a0e2 5a03087
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Thu Apr 28 17:46:42 2016 -0400

    merge with master

commit 036a0e25855cdcd6918aad62fca6c9a58d176145
Author: Kellie Ren Lu <krl2130@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 <krl2130@columbia.edu>
Date:   Thu Apr 28 09:59:41 2016 -0400

    fixed shift reduce conflict in parser

commit 5f1d744c18f7a74ad104335f7ed88b90e15e5bda
Author: Kellie Ren Lu <krl2130@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 <krl2130@columbia.edu>
Date:   Tue Apr 26 01:44:00 2016 -0400

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

commit 4e65179d2cde7da4c2ebd2bd99e6cff8ea732e43
Author: Kellie Ren Lu <krl2130@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 <krl2130@columbia.edu>
Date:   Sun Apr 10 21:28:38 2016 -0400

    readded ptx that was accidentally deleted

commit 345932517f38f9c29c0500c11b80a6c1eb2fe660
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Sun Apr 10 19:54:57 2016 -0400

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

commit 3903f60107bc48206d36c9c22e7dd027ead2c07e
Merge: 643f188 5e5abda
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Sat Apr 9 15:01:19 2016 -0400

    merge with diana

commit 643f1885f42180fab3fbd4cf00d07b79b57c04fb
Author: Kellie Ren Lu <krl2130@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 b13c985e43e25b3cf6fa1ee9efcf8533120a86d0
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 <krl2130@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 <krl2130@columbia.edu>
Date:   Thu Apr 7 14:19:17 2016 -0400

    added multidimensional arrays

commit 2af0b2a9365799b30b35333e46f6bb97197ee3aa
Merge: 9b09150 472cbe9
Author: Kellie Ren Lu <krl2130@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 <krl2130@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 fd2ed93d9cf9c3eead41c0445f2fe196efc6498e
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 71971f831719bddd5a70a993faef458e2586bc92
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 064cd6549c63b8ac6a1b6fcaf43d36e45e218aa3
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 d58606aeefe9dea595be512895bd3dce314ab52a
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 <krl2130@columbia.edu>
Date:   Mon Apr 4 22:42:20 2016 -0400

    compiles

commit 47ebdf0c55850fb952d62829174fb4bf990e1860
Merge: 346a82d 360fa9e
Author: Kellie Ren Lu <krl2130@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 <krl2130@columbia.edu>
Date:   Mon Apr 4 12:15:19 2016 -0400

    small syntax fixes

commit 360fa9e2acf7ea50165ecb0c4009f98cbd490494
Author: Kellie Ren Lu <krl2130@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 a6211f99053ae983b02a3116449e7963eaab5a7d
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Sun Apr 3 08:42:16 2016 -0400

    working parser and scanner

commit 9b064df0596a2371dbc5f9d39b84c3f284472d6f
Author: Kellie Ren Lu <krl2130@columbia.edu>
```

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

    diana

commit 54a73568db4b81c96e80778da1cbb0cf43eea3e9
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Sun Apr 3 03:43:40 2016 -0400

    before altering new parser

commit b8d0b5562a88a9ac9a44f37f2883ef7d4633555d
Author: Kellie Ren Lu <krl2130@columbia.edu>
Date:   Sat Apr 2 19:17:55 2016 -0400

    new parser implementation

commit d77fc630cd0127425410e939b6da8a4c7d035611
Author: Kellie Ren Lu <krl2130@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 <krl2130@columbia.edu>
Date:   Sun Mar 27 14:15:00 2016 -0400

    updated sample programs

commit 6c3cab724a23cd0c22607a98beb75b5f1c2c9456
Author: Kellie Ren Lu <krl2130@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 <krl2130@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 ff775f52111abe1e376b633d2fa61b39d636c4fd
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 5de939d8c9c8c4d7bfbae1edaa851b606d538afd
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 <krl2130@columbia.edu>
Date:   Sun Mar 6 21:40:30 2016 -0500

    solved merge conflicts

commit 1b22e9f997589381534e8d6553a240ac4bf8b875
Author: Kellie Ren Lu <krl2130@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 <krl2130@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 <krl2130@columbia.edu>
Date:   Sun Mar 6 19:43:22 2016 -0500

    finished lrm

commit 81940b2d557d9477806b5deb29dca311802fe39c
Merge: da86a4f b51cfca
Author: Kellie Lu <krl2130@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 da86a4fbccbfb54fb3263dfbc29d19250fd1ff88
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

```
214 lines (175 sloc) 7.61 KB
open Ast
(* Contains sast type definitions for conversions during semantic analysis *)
(* -----------------------------------------PTX types -----------------------------------------*)
type ptx_data_movement =
  | Ptx_Move | Ptx_Load | Ptx_Store

type ptx_binary_operator =
    | Ptx_Add | Ptx_Subtract | Ptx_Multiply | Ptx_Divide | Ptx_Modulo

type ptx_data_type =
  | U16 | U32 | U64 | S16 | S32 | S64

(* should use this as our information about global/param etc.*)
type ptx_variable_type =
  | Ptx_Primitive of ptx_data_type
  | Ptx_Array of ptx_variable_type * int           (* 'int' refers to the length of the array *)
  | Ptx_Pointer of ptx_variable_type * int          (* 'int' refers to size of memory pointed by the pointer *)

type ptx_register_decl =
  | Register_Declaration of ptx_data_type * string * int   (* type, name, number of registers *)

type ptx_register =
  | Register of string * int                       (* register name,  register number *)
(* Not sure what this is   | Typed_Register of ptx_data_type * string * int      (* type, register name, register number *) *)
(* Implement later  | Special_Register of string       (* register name *) *)

```

```
type ptx_parameter =
  | Parameter_register of ptx_register
  | Parameter_constant of int
  | Parameter_variable of Ast.identifier


type ptx_expression =
  | Ptx_reg_declaration of ptx_register_decl
  | Ptx_movement of ptx_data_movement * ptx_data_type * ptx_variable_type * ptx_parameter * ptx_parameter
  | Ptx_Binop of ptx_binary_operator * ptx_data_type * ptx_parameter * ptx_parameter * ptx_parameter
  | Ptx_Return
(*      | Ptx_Array_Literal of ptx_expression list
  | Ptx_Function_Call of Ast.identifier * ptx_expression list
  | Ptx_Identifier_Expression of Ast.identifier
 *)

type ptx_subroutine = {
  routine_name                : Ast.identifier;
  routine_expressions              : ptx_expression list;
}

type ptx_statement =
(*      | Ptx_Initialization of ptx_vdecl * ptx_expression *)
(*      | Ptx_Assignment of Ast.identifier * ptx_expression *)
    | Ptx_expression of ptx_expression
    | Ptx_subroutine of ptx_subroutine

type ptx_function_type =
  | Global
  | Device

type ptx_constant =
{
  ptx_constant_name               : Ast.identifier;
  ptx_constant_variable_type          : ptx_variable_type;
}

type ptx_variable_space =
  | Global
  | Local
  | Shared

type ptx_vdecl =
    | Ptx_Vdecl of ptx_data_type * ptx_variable_space(* need something about global/ptrs here*) ptx_variable_type * Ast.identifier
```

```
(* ptx fdecl is the entire file
  it seems it really only needs to be composed of a few parts - a name, a variable declaration list
  and a statement list
  register_decl list should go inside body generated from semantic analyzer
*)
type ptx_fdecl = {
  (* Global or Device *)
  ptx_fdecl_type                  : ptx_function_type; (* probably not needed *)

  (* Name of the function *)
  ptx_fdecl_name                  : Ast.identifier;

  (* Expected parameters of the function *)
  ptx_fdecl_params                : ptx_vdecl list;

  (* List of constants that function needs to know - aka variables that aren't in scope of function when it goes through semantic analyzer
    If this constant list doesn't match the constant list of the higher order function, throw error in semant.ml *)
  ptx_consts                    : ptx_constant list;
  (* Declares the virtual registers that are needed for the function *)
  register_decls                  : ptx_register_decl list;
  (* Statements within the function *)
  ptx_fdecl_body                  : ptx_statement list;
}






(* -----------------------------------------C types -----------------------------------------*)

(*----------------------------------Unnecessary?????????----------------------------------*)
type c_binary_operator =
    | Add | Subtract | Multiply | Divide | Modulo
(*     | Plus_Equal | Subtract_Equal | Multiply_Equal | Divide_Equal  *)
(*     | Exp | Dot | Matrix_Multiplication *)
    | And | Or | Xor
    | Equal | Not_Equal | Greater_Than | Less_Than | Greater_Than_Equal | Less_Than_Equal
    | Bitshift_Right | Bitshift_Left
type c_unary_operator =
    | Not | Negate
    | Plus_Plus | Minus_Minus

type c_data_type =
  | String
    | Byte
    | Unsigned_Byte
    | Integer
    | Unsigned_Integer
    | Long
    | Unsigned_Long
    | Float
    | Double
    | Boolean
    | Void

type c_variable_type =
  | Primitive of c_data_type
  | Array of c_variable_type * int
(*   | Struct of variable_type list * expression list * int
 ↪ *)

type c_vdecl =
    Variable_Declaration of c_variable_type *
 ↪ Ast.identifier

(*
 ↪ ----------------------------------Necessary-----------------------------
 ↪ *)

type c_kernel_variable_info = {
  variable_type       : c_variable_type;
  host_name         : Ast.identifier;
  kernel_name       : Ast.identifier;
}

type c_higher_order_function_call = {
  (* Map or reduce *)
  higher_order_function_type         : Ast.identifier;
  (* Name of kernel function that is called from host
 ↪ (would be kernel function corresponding to map/reduce)
 ↪ *)
    applied_kernel_function            : Ast.identifier;
  (* List of constants passed into map and reduce *)
  constants                 : c_kernel_variable_info list;
  (* Size of input and return arrays *)
  array_length               : int;
  (* Input array information
    --If an array has no name (just simply passed in as
 ↪ something like {1,2,3}) then it is given a temporary
 ↪ generated name *)
  input_arrays_info            : c_kernel_variable_info
 ↪ list; (* type, host name, kernel name *)
```

```
    (* Return array information *)
    return_array_info                    :
 ↪  c_kernel_variable_info; (* type, host name, kernel
 ↪  name*)
}

(* Type for calling defg functions directly from host *)
type c_kernel_function_call = {
  (* Name of the function that is called from the host *)
  kernel_function             : Ast.identifier;
  (* Input array information
    --If an array has no name (just simply passed in as
 ↪  something like {1,2,3}) then it is given a temporary
 ↪  generated name *)
  input_args_info              : c_kernel_variable_info
 ↪  list; (* type, host name, kernel name *)
    (* Return array information *)
    return_arg_info                    :
 ↪  c_kernel_variable_info; (* type, host name, kernel
 ↪  name*)
}

type c_expression =
      | Function_Call of Ast.identifier * c_expression list
      | Higher_Order_Function_Call of
 ↪  c_higher_order_function_call
      | Kernel_Function_Call of c_kernel_function_call
      | String_Literal of string
      | Integer_Literal of int
      | Boolean_Literal of bool
      | Floating_Point_Literal of float
      | Array_Literal of c_expression list
      | Identifier_Literal of Ast.identifier
      | Cast of c_variable_type * c_expression
      | Binop of c_expression * c_binary_operator *
 ↪  c_expression
      | Unop of c_expression * c_unary_operator
      | Array_Accessor of c_expression * c_expression list (*
 ↪  Array, indexes *)
      | Ternary of c_expression * c_expression * c_expression
 ↪  (* expression if true, condition, expression if false
 ↪  *)

type c_variable_statement =
      | Declaration of c_vdecl
      | Initialization of c_vdecl * c_expression
      | Assignment of Ast.identifier * c_expression

type c_statement =
      | Variable_Statement of c_variable_statement
      | Expression of c_expression
      | Block of c_statement list (* Used for if, else, for,
 ↪  while blocks *)
```

```
    | If of c_expression * c_statement * c_statement (* expression-condition, statement-if block, statement-optional else block *)
    | While of c_expression * c_statement
    | For of c_statement * c_expression * c_statement * c_statement
    | Return of c_expression
    | Return_Void
    | Continue
    | Break

type c_fdecl = {
    c_fdecl_return_type     : c_variable_type;
    c_fdecl_name           : Ast.identifier;
    c_fdecl_params         : c_vdecl list;
    c_fdecl_body           : c_statement list;
}

(* Overall Program *)
type program = c_variable_statement list * ptx_fdecl list * c_fdecl list
```

parser.mly

```
%{ open Ast;; (*open Exceptions;;*)


    (* Converts keywords to appropriate datatype *)
    let string_to_data_type = function
  | "string" -> String
    | "bool" -> Boolean
    | "void" -> Void
    | "ubyte" -> Unsigned_Byte
    | "byte" -> Byte
    | "uint" -> Unsigned_Integer
  | "int" -> Integer
    | "ulong" -> Unsigned_Long
    | "long" -> Long
    | "float" -> Float
    | "double" -> Double
  | dtype -> raise (Exceptions.Invalid_data_type dtype)

%}

%token LPAREN RPAREN LBRACKET RBRACKET LCURLY RCURLY INDENT DEDENT COLON TERMINATOR EOF COMMA
%token DEF DEFG RETURN CONSTS TILDA
%token <int> DEDENT_EOF, DEDENT_COUNT

%token ADD SUBTRACT MULTIPLY DIVIDE MODULO
%token PLUS_PLUS MINUS_MINUS
%token BITSHIFT_RIGHT BITSHIFT_LEFT
%token AND OR NOT XOR
%token EQUAL NOT_EQUAL GREATER_THAN GREATER_THAN_EQUAL LESS_THAN LESS_THAN_EQUAL
%token IF ELSE WHILE FOR
%token CONTINUE BREAK

%token ASSIGNMENT

%token <int> INTEGER_LITERAL
%token <string> STRING_LITERAL
%token <float> FLOATING_POINT_LITERAL
%token <bool> BOOLEAN_LITERAL

%token <string> IDENTIFIER
%token <string> DATATYPE

%nonassoc ELSE NOELSE
%right ASSIGNMENT
%left IF
%left LBRACKET RBRACKET
%left EQUAL NOT_EQUAL GREATER_THAN GREATER_THAN_EQUAL LESS_THAN LESS_THAN_EQUAL
```

```
%left AND NOT OR XOR
%left BITSHIFT_RIGHT BITSHIFT_LEFT
%left ADD SUBTRACT PLUS_PLUS MINUS_MINUS
%left MULTIPLY DIVIDE MODULO
%right NEGATE

%start program
%type <Ast.program> program

%%

program:
    |   /* nothing */ { [], [] } /* variable statements, function declarations */
    | program variable_statement TERMINATOR { List.rev ($2 :: List.rev (fst $1)), snd $1 }
    | program fdecl { fst $1, List.rev($2 :: List.rev(snd $1))   }

identifier:
    | IDENTIFIER { Identifier($1)}

/* Kernel and host function declarations */
fdecl:
    | variable_type DEF identifier LPAREN parameter_list RPAREN COLON indent_block
        {{
            is_kernel_function = false;
            return_type = $1;
            name = $3;
            params = $5;
            body = $8;
        }}
    | variable_type DEFG identifier LPAREN parameter_list RPAREN COLON indent_block
        {{
            is_kernel_function = true;
            return_type = $1;
            name = $3;
            params = $5;
```

```

 ↪  body = $8;

 ↪  }}

/* Constant parameters for higher order function calls */
constant:
      | identifier ASSIGNMENT expression
 ↪  {Constant($1,$3)}

constant_list:
      |  /* nothing */
 ↪  { [] }
      | nonempty_constant_list
 ↪  { $1 }

nonempty_constant_list:
      | constant COMMA nonempty_constant_list
 ↪  {$1 :: $3}
      | constant
 ↪  { [$1] }


/* Higher order function calls */
higher_order_function_call:
      | TILDA identifier LPAREN identifier COMMA CONSTS
 ↪  LPAREN constant_list RPAREN COMMA
 ↪  nonempty_array_expression_list RPAREN

 ↪  {{

 ↪  higher_order_function_type = $2;

 ↪  kernel_function_name = $4;

 ↪  constants = $8;

 ↪  input_arrays = $11;

 ↪  }}
      | TILDA identifier LPAREN identifier COMMA
 ↪  nonempty_array_expression_list RPAREN

 ↪  {{

 ↪  higher_order_function_type = $2;

 ↪  kernel_function_name = $4;

 ↪  constants = [];

 ↪  input_arrays = $6;

 ↪  }}
```

```



/* Parameters for normal host functions and kernel
 ↪ functions */
vdecl:
      | variable_type identifier
 ↪  { Variable_Declaration($1,$2)}

nonempty_parameter_list:
      | vdecl
 ↪  { [$1] }
      | nonempty_parameter_list COMMA vdecl
 ↪  {$3 :: $1}

parameter_list:
      | /* nothing */
 ↪  { [] }
      | nonempty_parameter_list
 ↪  { $1 }



/* Statements */
variable_statement:
      | vdecl TERMINATOR
 ↪  { Declaration($1) }
      | assignment_expression ASSIGNMENT expression
 ↪ TERMINATOR                { Assignment( $1, $3 ) }
      | vdecl ASSIGNMENT expression TERMINATOR
 ↪  { Initialization ($1, $3) }

for_statement:
      | assignment_expression ASSIGNMENT expression
 ↪  { Variable_Statement(Assignment($1,$3 ))}
      | vdecl ASSIGNMENT expression
 ↪  { Variable_Statement(Initialization($1,$3))}

statement:
      | expression TERMINATOR
 ↪  { Expression($1) }
      | RETURN expression TERMINATOR
 ↪  { Return($2) }
      | RETURN TERMINATOR
 ↪  { Return_Void }
      | CONTINUE TERMINATOR
 ↪  { Continue }
      | BREAK TERMINATOR
 ↪  { Break }
      | IF LPAREN expression RPAREN COLON indent_block %prec
 ↪ NOELSE                               { If($3,
 ↪ Block($6), Block([])) }
```

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

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

    /* Group of statements */
indent_block:
    | /* nothing */ { [] }
    | INDENT nonempty_statement_list DEDENT { $2 }



/* Expressions */
expression:
    | identifier LPAREN expression_list RPAREN { Function_Call($1,$3) }
    | higher_order_function_call { Higher_Order_Function_Call($1)}

    | LPAREN expression RPAREN { $2 }

    | STRING_LITERAL { String_Literal($1) }
    | INTEGER_LITERAL { Integer_Literal($1) }
    | BOOLEAN_LITERAL { Boolean_Literal($1) }
    | FLOATING_POINT_LITERAL { Floating_Point_Literal($1) }
    | array_literal { $1 }
    | identifier { Identifier_Literal($1)}

    | expression AND expression { Binop($1, And, $3) }
    | expression OR  expression { Binop($1, Or, $3) }
    | expression XOR expression { Binop($1, Xor, $3) }
    | NOT expression { Unop($2, Not) }

    | expression EQUAL expression { Binop($1, Equal, $3) }
    | expression NOT_EQUAL expression { Binop($1, Not_Equal, $3 )}
    | expression GREATER_THAN expression { Binop($1, Greater_Than, $3 )}
    | expression GREATER_THAN_EQUAL expression { Binop($1, Greater_Than_Equal, $3 )}
    | expression LESS_THAN expression { Binop($1, Less_Than, $3) }
    | expression LESS_THAN_EQUAL expression { Binop($1, Less_Than_Equal, $3)}

    | SUBTRACT expression { Unop($2,Negate) }
    | expression ADD expression { Binop($1, Add, $3) }
    | expression PLUS_PLUS { Unop($1,Plus_Plus)}
    | expression MINUS_MINUS { Unop($1, Minus_Minus)}
    | expression SUBTRACT expression { Binop($1, Subtract, $3) }
    | expression MULTIPLY expression { Binop($1, Multiply, $3) }
    | expression DIVIDE expression { Binop($1, Divide, $3) }
    | expression MODULO expression { Binop($1, Modulo, $3)}
    | expression BITSHIFT_RIGHT expression { Binop($1, Bitshift_Right,$3) }
    | expression BITSHIFT_LEFT expression { Binop($1, Bitshift_Left,$3) }
    | variable_type LPAREN expression RPAREN { Cast($1, $3)}

    | expression IF LPAREN expression RPAREN ELSE expression            { Ternary($1,$4,$7) }
    | array_expression nonempty_array_accessor_list { Array_Accessor($1,$2) }


nonempty_expression_list:
    | expression COMMA nonempty_expression_list { $1 :: $3 }
    | expression { [$1] }

```

```
expression_list:
    | /* nothing */
 ↪  { [] }
    | nonempty_expression_list
 ↪  { $1 }


array_accessor:
    | LBRACKET expression RBRACKET
 ↪  { $2 }

nonempty_array_accessor_list:
    | nonempty_array_accessor_list array_accessor
 ↪  { $2 :: $1 }
    | array_accessor
 ↪  { [$1] }

array_literal:
    | LCURLY nonempty_expression_list RCURLY
 ↪  { Array_Literal($2)}

array_expression:
    | identifier
 ↪  { Identifier_Literal($1) }
    | array_literal
 ↪  { $1 }


nonempty_array_expression_list:
    | array_expression
 ↪  { [$1] }
    | nonempty_array_expression_list COMMA array_expression
 ↪  { $3 :: $1 }

 /* Expressions that can be assigned on the right side of
 ↪  the assignment statement */
assignment_expression:
    | identifier
 ↪  { Identifier_Literal($1) }
    | array_expression nonempty_array_accessor_list
 ↪  { Array_Accessor($1,$2) }


/* Variable types  and Data types */
data_type:
    | DATATYPE
 ↪  { string_to_data_type $1 }
variable_type:
    | data_type
 ↪  { Primitive($1) }
    | data_type array_dimension_list
        {
            let rec create_array vtype dim_list=
        match dim_list with
          | [] -> raise (Exceptions.Array_parsing_error)
          | head::[] -> Array(Primitive(vtype),head)
          | head::tail -> Array((create_array vtype tail),head)
      in create_array $1 $2

    }
array_dimension:
  | LBRACKET INTEGER_LITERAL RBRACKET { $2 }

array_dimension_list:
  | array_dimension { [$1] }
  | array_dimension array_dimension_list { $1 :: $2 }
```

semant.ml

```
open Ast
open Sast
(* open Utils *)
open Exceptions

(* Maps variable name to variable type and value *)
module Variable_Map = Map.Make(String);;
(* Maps function name to return type *)
module Function_Map = Map.Make(String);;

(* For generating names for the device pointers *)
let dev_name_counter = ref 0
(* For generating names for the host pointers *)
let host_name_counter = ref 0
(* For generating names for each ptx map function *)
let map_ptx_name_counter = ref 0
(* For generating names for each c map function *)
let map_c_name_counter = ref 0
(* For generating names for each reduce function *)
let reduce_c_name_counter = ref 0
(* For generating names for ptx reduce function *)
let reduce_ptx_name_counter = ref 0
(* For generating arg names*)
let arg_counter = ref 0
(* Generates names for ptx return values *)
let ptx_return_counter = ref 0
(* Generates names for subroutines*)
let subroutine_counter = ref 0

(* For generating register counters for datatypes *)
let signed_int_counter = ref 1
let signed_float_counter = ref 1
let predicate_counter = ref 1
let block_counter = ref 1
let pointer_counter = ref 1
(*-----------------------------------Generates Symbols Based on Counters-----------------------------------*)
let generate_device_pointer_name () =
    let name = "dev_ptr" ^ (string_of_int !dev_name_counter) in
    incr dev_name_counter;
    name

let generate_host_pointer_name () =
    let name = "host_ptr" ^ (string_of_int !host_name_counter) in
    incr host_name_counter;
    name

let generate_map_c_function_name () =
    let name = "map_c" ^ (string_of_int !map_c_name_counter) in
    incr map_c_name_counter;
    name

let generate_map_ptx_function_name () =
    let name = "map_ptx" ^(string_of_int !map_ptx_name_counter) in
    incr map_ptx_name_counter;
    name

let generate_reduce_c_function_name () =
    let name = "red_c" ^ (string_of_int !reduce_c_name_counter) in
    incr reduce_c_name_counter;
    name
let generate_reduce_ptx_function_name () =
    let name = "red_ptx" ^ (string_of_int !reduce_ptx_name_counter) in
    incr reduce_ptx_name_counter;
    name

let generate_arg_name () =
    let name = "arg" ^ (string_of_int !arg_counter) in
    incr arg_counter;
    name
let generate_ptx_return_name () =
    let name = "func_ret" ^ (string_of_int !ptx_return_counter) in
    incr arg_counter;
    name
let generate_subroutine_name () =
    let name = "SUB_ROUT" ^ (string_of_int !subroutine_counter) in
    incr subroutine_counter;
    name

let get_signed_int_counter () =
    let orig = !signed_int_counter in
    incr(signed_int_counter);
    orig

let get_signed_float_counter () =
    let orig = !signed_float_counter in
    incr(signed_float_counter);
    orig

let get_predicate_counter () =
    let orig = !predicate_counter in
    incr(predicate_counter);
    orig

let get_pointer_counter () =
    let orig = !pointer_counter in
    incr(pointer_counter);
    orig
(*-----------------------------------Types for Semantic Analysis-----------------------------------*)
(* Three types of functions *)
type cuda_function_type  =
  | Kernel_Global
  | Kernel_Device
  | Host

(* Stores information about a function *)
type function_info = {
  (* Host, kernel_device, kernel_global *)
  function_type                   : cuda_function_type;
  (* Name of function *)
  function_name                   : Ast.identifier;
  (* Function return type and arguments *)
  function_return_type            : Ast.variable_type;
  function_args                   : (Ast.variable_type) list;
  (* Functions that are called within this function - needs to be specifically noted for gpu and ptx functions *)
  dependent_functions             : Ast.identifier list;
  (* Unknown ,possibly constant variables -> for kernel_device and kernel_global *)
  unknown_variables               : Ast.identifier list;
}

type variable_info = {
  vtype : Ast.variable_type;
  register_number :int;
}

(* Stores information about the environment *)
type environment = {
  (* Variables that have been declared in the environment - stores variable name, variable type *)
  variable_scope_stack                              : variable_info Variable_Map.t list;
  (* List of kernel functions that have been declared in the environment  - info from function_info record *)
  kernel_function_map                               : function_info Function_Map.t;
  (* List of host functions that have been declared in the environment - info from function_info record *)
  host_function_map                                 : function_info Function_Map.t;
  (* Bool specifying whether environment is being evaluated on the gpu *)
  is_gpu_env                                        : bool;
  (*List of functions for higher order functions *)
  hof_c_function_list                               : Sast.c_higher_order_fdecl list;
  hof_ptx_function_list                           : Sast.ptx_higher_order_fdecl list;
  (* Contains list of ptx_identifiers *)
  expression_stack                                : Sast.ptx_literal list list;
  return_lit                                      : Sast.ptx_literal;
}

(*-----------------------------------Helper functions to check variables and functions in the environment -----------------------------------*)

let builtin_functions = ["print";]


(* Checks if function is a builtin function *)
(* Used to check function declarations to make sure they aren't declaring anything with the same name *)
let is_builtin_function id =
  List.exists (fun function_name -> function_name = id) builtin_functions


(* Creates a function_info record with information *)
let create_function_info ftype rtype args df uv name = {
  function_type                       = ftype;
  function_name                       = Identifier(name);
  function_return_type                = rtype;
  function_args                       = args;
  dependent_functions                 = df;
  unknown_variables                   = uv;
}


(* Function for adding initializing host function map, adds builtin functions to host function map*)
let init_host_function_map =
  let fmap = Function_Map.empty in
  let rec add_functions fmap function_list =
    match function_list with
      | [] -> fmap
      | f_info::tl -> add_functions (Function_Map.add (Utils.idtos(f_info.function_name)) f_info fmap) tl
  in
  let print_function = {
      function_type = Host;
      function_name = Ast.Identifier("print");
      function_return_type = Ast.Primitive(Ast.Void);
      function_args = [Ast.Primitive(Ast.String)];
      dependent_functions = [];
      unknown_variables = [];
  }
  in
  let random_function = {
      function_type = Host;
      function_name = Ast.Identifier("random");
      function_return_type = Ast.Primitive(Ast.Integer);
      function_args = [];
      dependent_functions = [];
      unknown_variables = [];
  }
in
  (*  let create_built_in_function = (create_function_info
 ↪  Host (Ast.Primitive(Ast.Void)) [] [] []) in
  let builtin_function_info_structs = List.map
 ↪  create_built_in_function builtin_functions in *)
  add_functions fmap [print_function;random_function]

let make_ptx_id name reg num write_reg is_ptr =
  {
    var_name    = name; (* Name as passed as a param or
 ↪  declared *)
    reg_name    = reg; (* Register name it is stored in *)
    reg_num     = num;
    write_reg   = write_reg;
    is_ptr      = is_ptr;
  }

(* Creates a new environment *)
let init_env = {
  variable_scope_stack        = Variable_Map.empty :: [];
  kernel_function_map         = Function_Map.empty;
  host_function_map           = init_host_function_map;
  is_gpu_env                  = false;
  (* Two lists that stores the new higher order functions
 ↪  we need to add*)
  hof_c_function_list         = [];
  hof_ptx_function_list       = [];
  expression_stack            = [[]];
  return_lit                  =
 ↪  Sast.Ptx_Identifier_Literal(make_ptx_id (Ast.Identifier
 ↪  "") "" 0 true true )
}


(* Updates the environment *)
let update_env vscope_stack kfmap hfmap is_gpu hof_c_list
 ↪  hof_ptx_list ptx_e_stack rlit= {
  variable_scope_stack        = vscope_stack;
  kernel_function_map         = kfmap;
  host_function_map           = hfmap;
  is_gpu_env                  = is_gpu;
  hof_c_function_list         = hof_c_list;
  hof_ptx_function_list       = hof_ptx_list;
  expression_stack            = ptx_e_stack;
  return_lit                  = rlit;
}
```

```
(* Pushes a new scope on top of the  variable_scope_stack
 ↪  *)
let push_scope env =
    update_env (Variable_Map.empty ::
 ↪  env.variable_scope_stack) env.kernel_function_map
 ↪  env.host_function_map env.is_gpu_env
 ↪  env.hof_c_function_list env.hof_ptx_function_list
 ↪  env.expression_stack env.return_lit

(* Pops a scope from the top of the variable_scope_stack *)
let pop_scope env =
    match env.variable_scope_stack with
      | [] -> raise
 ↪  Exceptions.Cannot_pop_empty_variable_scope_stack
      | local_scope :: tail ->
          update_env tail env.kernel_function_map
 ↪  env.host_function_map env.is_gpu_env
 ↪  env.hof_c_function_list env.hof_ptx_function_list
 ↪  env.expression_stack env.return_lit

let update_scope updated_scope env =
    let env = pop_scope env in
    update_env (updated_scope::env.variable_scope_stack)
 ↪  env.kernel_function_map env.host_function_map
 ↪  env.is_gpu_env env.hof_c_function_list
 ↪  env.hof_ptx_function_list env.expression_stack
 ↪  env.return_lit

let update_kernel_fmap f_info env =
    let new_kfmap = Function_Map.add
 ↪  (Utils.idtos(f_info.function_name)) f_info
 ↪  env.kernel_function_map in
    update_env env.variable_scope_stack new_kfmap
 ↪  env.host_function_map env.is_gpu_env
 ↪  env.hof_c_function_list env.hof_ptx_function_list
 ↪  env.expression_stack env.return_lit

let update_host_fmap f_info env =
    let new_hfmap = Function_Map.add
 ↪  (Utils.idtos(f_info.function_name)) f_info
 ↪  env.host_function_map in
    update_env env.variable_scope_stack
 ↪  env.kernel_function_map new_hfmap env.is_gpu_env
 ↪  env.hof_c_function_list env.hof_ptx_function_list
 ↪  env.expression_stack env.return_lit

let update_hof_lists hof_c_fdecl hof_ptx_fdecl env =
    update_env env.variable_scope_stack
 ↪  env.kernel_function_map env.host_function_map
 ↪  env.is_gpu_env
 ↪  (List.rev(hof_c_fdecl::List.rev(env.hof_c_function_list)))
 ↪  (List.rev(hof_ptx_fdecl::List.rev(env.hof_ptx_function_list)))
 ↪  env.expression_stack env.return_lit

let update_return_lit ptx_lit env =
    update_env env.variable_scope_stack
 ↪  env.kernel_function_map env.host_function_map
 ↪  env.is_gpu_env env.hof_c_function_list
 ↪  env.hof_ptx_function_list  env.expression_stack ptx_lit

let pop_expression_stack env =
    match env.expression_stack with
      | [] -> env
      | hd::tl -> update_env env.variable_scope_stack
 ↪  env.kernel_function_map env.host_function_map
 ↪  env.is_gpu_env  env.hof_c_function_list
 ↪  env.hof_ptx_function_list tl env.return_lit

let push_expression_stack env =
  update_env env.variable_scope_stack
 ↪  env.kernel_function_map env.host_function_map
 ↪  env.is_gpu_env  env.hof_c_function_list
 ↪  env.hof_ptx_function_list ([]::env.expression_stack)
 ↪  env.return_lit

let update_expression_stack lit env =
  let update = lit::(List.hd env.expression_stack) in
  let env = pop_expression_stack env in
  update_env env.variable_scope_stack
 ↪  env.kernel_function_map env.host_function_map
 ↪  env.is_gpu_env  env.hof_c_function_list
 ↪  env.hof_ptx_function_list
 ↪  (update::env.expression_stack) env.return_lit

(* Retrieves nth element from head list of expression stack
 ↪  *)
let get_from_expression_stack nth env =
  if nth > (List.length (List.hd env.expression_stack)) ||
 ↪  nth < 0 then raise
 ↪  Exceptions.Invalid_expression_stack_access
  else match env.expression_stack with
      | [] -> raise
 ↪  Exceptions.Cannot_access_empty_expression_stack
      | hd::tl -> List.nth (List.hd env.expression_stack)
 ↪  nth

(* Checks if variable has been declared - is valid - in the
 ↪  scope *)
let is_variable_in_scope id env =
    let rec check_scopes scope_stack =
      match scope_stack with
        | [] -> false
        | [scope] ->
          if env.is_gpu_env then false
          else (Variable_Map.mem id scope)
        | scope :: larger_scopes ->
          if (Variable_Map.mem id scope) then true
          else check_scopes larger_scopes
      in check_scopes env.variable_scope_stack


(* Searches variable in scope for CUDA C and returns its
  ↪  type *)
let get_variable_type id env =
  let rec check_scopes scope_stack =
    match scope_stack with
      | [] -> raise (Exceptions.Variable_not_found_in_scope
  ↪  ( id))
      | scope::larger_scopes ->
          if Variable_Map.mem id scope then
            (Variable_Map.find id scope).vtype
          else
            check_scopes larger_scopes
  in check_scopes env.variable_scope_stack

let get_variable_info id env =
  let rec check_scopes scope_stack =
    match scope_stack with
      | [] -> raise (Exceptions.Variable_not_found_in_scope
  ↪  (  id))
      | scope::larger_scopes ->
          if Variable_Map.mem id scope then
            Variable_Map.find id scope
          else
            check_scopes larger_scopes
  in check_scopes env.variable_scope_stack

let update_variable_register id reg_num env =
  let old_info = get_variable_info id env in
  let new_info = { vtype = old_info.vtype; register_number
  ↪  = reg_num;} in
  let new_vmap = Variable_Map.add id new_info (List.hd
  ↪  env.variable_scope_stack) in
  update_scope new_vmap env

(* Helper function that returns checks types are the same
  ↪  *)
let same_types t1 t2 = (t1 = t2)

(* Checks if function is valid in the environment *)
let is_function_in_scope id env =
  if env.is_gpu_env = true then (Function_Map.mem id
  ↪  env.kernel_function_map)
  else (Function_Map.mem id env.host_function_map) || (Function_Map.mem id env.kernel_function_map)


(* Searches for function called in function call and returns information about the function *)
let get_function_info id env =
    if env.is_gpu_env = true then
        (if(Function_Map.mem id env.kernel_function_map) then
            (Function_Map.find id env.kernel_function_map)
        else raise (Exceptions.Function_not_defined (id)))
    else
        (if (Function_Map.mem id env.host_function_map) then
            (Function_Map.find id env.host_function_map)
        else if (Function_Map.mem id env.kernel_function_map) then
            (Function_Map.find id env.kernel_function_map)
        else raise (Exceptions.Function_not_defined (id)))

(* ----------------------------------- Functions for Checking Ast -----------------------------------*)
(* Checks a variable declaration and initialization to ensure variable hasn't already been declared *)
let check_already_declared id env =
  if ((is_variable_in_scope id env) = true) then true else false

(* Helper function that performs type inference for expressions *)
let rec infer_type expression env=
  let f type1 type2 =
    match type1 with
      | Some(t) -> (if t = type2 then Some(t)
                    else raise (Exceptions.Type_mismatch "wrong types"))
      | None -> Some(type2) in
  let match_type expression_list =
    let a = List.fold_left f None expression_list in
      match a with
        | Some(t) -> t
        | None -> raise Exceptions.Empty_array_expression_list in
  match expression with
    | Ast.String_Literal(_) -> Ast.Primitive(Ast.String)
    | Ast.Integer_Literal(_) -> Ast.Primitive(Ast.Integer)
    | Ast.Floating_Point_Literal(_) -> Ast.Primitive(Ast.Float)
    | Ast.Boolean_Literal(_) -> Ast.Primitive(Ast.Boolean)
    | Ast.Array_Literal(expr_list) ->
       let f expression = infer_type expression env in
      Ast.Array(match_type (List.map f expr_list),(List.length expr_list))
    | Ast.Identifier_Literal(id) ->
        if(check_already_declared (Utils.idtos id) env) = false then raise (Exceptions.Variable_not_found_in_scope (( Utils.idtos id)) )
        else (get_variable_type (Utils.idtos id) env)
    | Ast.Binop(e1,op,e2) ->
        (match op with
          | Ast.And | Ast.Or | Ast.Xor -> Ast.Primitive(Ast.Boolean)
          | _ -> if (same_types (infer_type e1 env) (infer_type e2 env)) = true then infer_type e1 env
                  else (raise (Exceptions.Type_mismatch("Binop types don't match")))
        )
    | Ast.Cast(vtype,e) -> vtype
    | Ast.Unop(e,unop) -> infer_type e env
    | Ast.Array_Accessor(e1,e_list,is_lvalue) ->
        (* Check e1 is an array *)
          (match infer_type e1 env with
            | Ast.Array(t,n) -> ()
            | _ -> (raise (Exceptions.Not_an_array_expression))
          );
        (* Check valid access *)
        let rec get_array_type arr dim_list =
          match dim_list with
            | [] -> raise Exceptions.Empty_array_access
            | hd::[] ->
              (match arr with
                | Ast.Array(t,n) -> t
                | _ -> raise Invalid_array_expression
              )
            | hd::tl ->
              ( match arr with
                | Ast.Array(t,n) -> get_array_type t tl
                | _ -> raise Exceptions.Invalid_array_expression
              )
        in get_array_type (infer_type e1 env) e_list
    | Ast.Ternary(e1,e2,e3) ->
        if(same_types (infer_type e1 env) (infer_type e2 env)) = true then infer_type e1 env else (raise (Exceptions.Type_mismatch("Ternary doesn't return same type")))
    | Ast.Higher_Order_Function_Call(hof) ->
      let f_info = get_function_info (Utils.idtos hof.kernel_function_name) env in
      let vtype = infer_type (List.hd hof.input_arrays) env in
        let length = match vtype with
```

```
          | Ast.Primitive(p) -> raise Exceptions.Invalid_array_expression
          | Ast.Array(t,n) -> n
        in
      Ast.Array(f_info.function_return_type,length)
    | Ast.Function_Call(id,e_list) ->
        let f_info = get_function_info (Utils.idtos id) env in
        f_info.function_return_type


(* Check that array has only one dimension - used for certain operations *)
let is_one_layer_array expression env =
    match expression with
    | Ast.Array_Literal(e_list) as array_literal ->
        let arr = infer_type array_literal env in
          (match arr with
          | Ast.Array(vtype,size) -> if size > 1 then false else true
          | _ -> raise Exceptions.Not_an_array_expression)
    | _ -> raise Exceptions.Not_an_array_expression


(* Helper function that returns a list of dimensions for an array variable type *)
let rec get_array_dimensions vtype dimensions =
  match vtype with
  | Ast.Array(t,n) ->
      get_array_dimensions t (List.rev(n::List.rev(dimensions)))
  | Ast.Primitive(p) -> dimensions
(*   | _ -> raise Exceptions.Unknown_variable_type *)




(* ----------------------------------- Functions for converting ast to sast (Also performs advanced checking) -----------------------------------*)
(* Converts a list of something to another list *)
let rec convert_list func ast_list sast_list env =
  match ast_list with
    | [] -> sast_list,env
    | hd::tl ->
      let sast_type, env = func hd env in
      convert_list func tl (List.rev (sast_type::List.rev(sast_list))) env


(* Generates a register for every variable type, keeps a counter for the types as well *)
let generate_reg vtype =
    match vtype with
        | Ast.Primitive(Ast.Integer) ->
 ↪  "%si",(get_signed_int_counter())
        | Ast.Primitive(Ast.Float) -> "%fl",
 ↪  get_signed_float_counter()
        | Ast.Primitive(Ast.Boolean) -> "%pr",
 ↪  get_predicate_counter()
        | Ast.Primitive(Ast.String) -> raise
 ↪  Exceptions.NO_STRINGS_ALLOWED_IN_GDECL
        | Ast.Primitive(Ast.Void) -> raise
 ↪  Exceptions.Void_type_in_gdecl
        | Ast.Array(vtype, size) -> "%ptr",
 ↪  get_pointer_counter()

(*Gets the string of the register type*)
let get_reg_type vtype =
    match vtype with
        | Ast.Primitive(Ast.Integer) -> "%si"
        | Ast.Primitive(Ast.Float) -> "%fl"
        | Ast.Primitive(Ast.Boolean) -> "%pr"
        | Ast.Primitive(Ast.String) -> raise
 ↪  Exceptions.NO_STRINGS_ALLOWED_IN_GDECL
        | Ast.Primitive(Ast.Void) -> raise
 ↪  Exceptions.Void_type_in_gdecl
        | Ast.Array(vtype, size) -> "%ptr"

(* Checks statement order - nothing follows a return ,
 ↪  break, or continue in a block*)
let rec good_statement_order stmt_list =
  match stmt_list with
    | [] ->true
    | hd ::[] -> true
    | hd :: tl ->
        match hd with
          | Ast.Return_Void | Ast.Continue | Ast.Break ->
 ↪  false
          | Ast.Return(e)-> false
          | _ -> good_statement_order tl

    (* Binop *)
let convert_to_c_binop binop env =
  match binop with
    | Ast.Add -> Sast.Add,env
    | Ast.Subtract -> Sast.Subtract,env
    | Ast.Multiply -> Sast.Multiply,env
    | Ast.Divide -> Sast.Divide,env
    | Ast.Modulo -> Sast.Modulo,env
    | Ast.And -> Sast.And,env
    | Ast.Or -> Sast.Or,env
    | Ast.Xor -> Sast.Xor,env
    | Ast.Equal -> Sast.Equal,env
    | Ast.Not_Equal -> Sast.Not_Equal,env
    | Ast.Greater_Than -> Sast.Greater_Than,env
    | Ast.Less_Than -> Sast.Less_Than,env
    | Ast.Greater_Than_Equal -> Sast.Greater_Than_Equal,env
    | Ast.Less_Than_Equal -> Sast.Less_Than_Equal,env
    | Ast.Bitshift_Right -> Sast.Bitshift_Right,env
    | Ast.Bitshift_Left -> Sast.Bitshift_Left,env
    | Ast.Bitwise_Or -> Sast.Bitwise_Or,env
    | Ast.Bitwise_And -> Sast.Bitwise_And,env

let convert_to_ptx_binop binop env =
  match binop with
    | Ast.Add -> Sast.Ptx_Add,env
    | Ast.Subtract -> Sast.Ptx_Subtract,env
    | Ast.Multiply -> Sast.Ptx_Multiply,env
    | Ast.Divide -> Sast.Ptx_Divide,env
    | Ast.Modulo -> Sast.Ptx_Modulo,env
    | Ast.And -> Sast.Ptx_And,env
    | Ast.Or -> Sast.Ptx_Or,env
    | Ast.Xor -> Sast.Ptx_Xor,env
    | Ast.Equal -> Sast.Ptx_Equal,env
    | Ast.Not_Equal -> Sast.Ptx_Not_Equal,env
    | Ast.Greater_Than -> Sast.Ptx_Greater_Than,env
    | Ast.Less_Than -> Sast.Ptx_Less_Than,env
    | Ast.Greater_Than_Equal -> Sast.Ptx_Greater_Than_Equal,env
    | Ast.Less_Than_Equal -> Sast.Ptx_Less_Than_Equal,env
    | Ast.Bitshift_Right -> Sast.Ptx_Bitshift_Right,env
    | Ast.Bitshift_Left -> Sast.Ptx_Bitshift_Left,env
    | Ast.Bitwise_Or -> Sast.Ptx_Bitwise_Or,env
    | Ast.Bitwise_And -> Sast.Ptx_Bitwise_And,env


  (* Unop *)
let convert_to_c_unop unop env =
  match unop with
    | Ast.Not -> Sast.Not,env
    | Ast.Negate -> Sast.Negate,env
    | Ast.Plus_Plus -> Sast.Plus_Plus,env
    | Ast.Minus_Minus -> Sast.Minus_Minus,env

let convert_to_ptx_unop unop env =
  match unop with
    | Ast.Not -> Sast.Ptx_Not,env
    | Ast.Negate -> Sast.Ptx_Negate,env
    | Ast.Plus_Plus -> Sast.Ptx_Plus_Plus,env
    | Ast.Minus_Minus -> Sast.Ptx_Minus_Minus,env


(* Datatype *)
let convert_to_c_data_type dtype env =
  match dtype with
      | Ast.Integer -> Sast.Integer,env
      | Ast.Float -> Sast.Float,env
      | Ast.String -> Sast.String,env
      | Ast.Boolean -> Sast.Boolean,env
      | Ast.Void -> Sast.Void,env

let convert_to_ptx_data_type dtype env =
  match dtype with
    | Ast.Integer -> Sast.S32,env
    | Ast.Float -> Sast.F32,env
    | Ast.Boolean -> Sast.Pred,env
    | Ast.String -> raise Exceptions.NO_STRINGS_ALLOWED_IN_GDECL
    | Ast.Void -> Sast.Ptx_Void,env

    (* Variable Type *)
let rec convert_to_c_variable_type vtype env =
  match vtype with
      | Ast.Primitive(p) ->
          let c_p,env = convert_to_c_data_type p env in
          Sast.Primitive(c_p),env
      | Ast.Array(t,n) ->
          let array_dims = get_array_dimensions vtype [] in
          let inside,env = (match t with
            | Ast.Array(t,n) ->
                convert_to_c_variable_type t env
            | Ast.Primitive(p) ->
                let c_p,env= convert_to_c_data_type p env in
                Sast.Primitive(c_p),env
          ) in
          Sast.Array(inside, array_dims),env

(* TO IMPLEMENT *)
let rec convert_to_ptx_variable_type vtype env =
  match vtype with
      | Ast.Primitive(p) ->
          let p2,env = convert_to_ptx_data_type p env in
          Sast.Ptx_Primitive(p2),env
      | Ast.Array(t,n) ->
          let array_dims = get_array_dimensions vtype [] in
          let c_t,env = convert_to_ptx_variable_type t env in
          Sast.Ptx_Array(c_t,array_dims),env

(* Variable Declarations *)
let convert_to_c_vdecl vdecl env  =
    match vdecl with
      | Ast.Variable_Declaration(vtype,id) ->
          if(check_already_declared (Utils.idtos(id)) env) = true then (raise (Exceptions.Variable_already_declared (Utils.idtos(id))))
          else
```

```
            let v_info = { vtype = vtype; register_number =
 ↪  0; } in
            let new_vmap = Variable_Map.add (Utils.idtos
 ↪  id) v_info (List.hd env.variable_scope_stack) in
            let env = update_scope new_vmap env in
            let c_vtype, env = convert_to_c_variable_type
 ↪  vtype env in
            Sast.Variable_Declaration(c_vtype,id),env

(*Statements are found as parameters in the defg*)
let convert_to_ptx_param vdecl env =
  match vdecl with
    | Ast.Variable_Declaration(vtype,id) ->
          if(check_already_declared (Utils.idtos id) env) =
 ↪  true then raise (raise
 ↪  (Exceptions.Variable_already_declared
 ↪  (Utils.idtos(id))))
          else
          (* Generate a register for each parameter - since
 ↪  its parameters, arrays are pointers *)
          let reg_name, reg_num = generate_reg vtype in let
 ↪  v_info = { vtype = vtype; register_number = reg_num;}
 ↪  in
          (* Update the variable in our variable map*)
          let new_vmap = Variable_Map.add (Utils.idtos id)
 ↪  v_info (List.hd env.variable_scope_stack) in
          let env = update_scope new_vmap env in
          (* Generates a PTX identifier that we want to use
 ↪  *)
          let is_array = match vtype with Ast.Primitive(p)
 ↪  -> false | Ast.Array(t,n) -> true in
          let ptx_id = make_ptx_id id reg_name reg_num
 ↪  false is_array in
          (match vtype with
              (* Convert these types to have state space
 ↪  param*)
              | Ast.Primitive(p) ->
 ↪  Sast.Ptx_Vdecl(Sast.Param,
 ↪  fst(convert_to_ptx_variable_type vtype env),ptx_id),env
              | Ast.Array(t,n) ->
 ↪  Sast.Ptx_Vdecl(Sast.Param,
 ↪  fst(convert_to_ptx_variable_type vtype env),ptx_id),env
          )


(*Statements are found in the body of the defg*)
let convert_to_ptx_vdecl vdecl env =
  match vdecl with
    | Ast.Variable_Declaration(vtype,id) ->
      if(check_already_declared (Utils.idtos id) env) =
 ↪  true then raise (raise
 ↪  (Exceptions.Variable_already_declared
 ↪  (Utils.idtos(id))))
```

```
   else
   (* Generate a register name for the variable and add
↪ it to our vmap*)
   let reg_name, reg_num = generate_reg vtype in let
↪ v_info = { vtype = vtype; register_number = reg_num;}
↪ in
   let new_vmap = Variable_Map.add (Utils.idtos id)
↪ v_info (List.hd env.variable_scope_stack) in
   let env = update_scope new_vmap env in
   let is_array = match vtype with Ast.Primitive(p) ->
↪ false | Ast.Array(t,n) -> true in
   let ptx_id = make_ptx_id id reg_name reg_num true
↪ is_array in
   (match vtype with
       (* Predicates can only be declared in
↪ register space *)
       | Ast.Primitive(Ast.Boolean) ->
↪ Sast.Ptx_Vdecl(Sast.Register,
↪ fst(convert_to_ptx_variable_type vtype env),ptx_id),env
       | Ast.Primitive(p)
↪ ->Sast.Ptx_Vdecl(Sast.Local,fst(convert_to_ptx_variable_type
↪ vtype env),ptx_id),env
       | Ast.Array(t,n) ->
↪ Sast.Ptx_Vdecl(Sast.Local,fst(convert_to_ptx_variable_type
↪ vtype env),ptx_id),env
   )

let same_types_list type_list =
 let main_type = (List.hd type_list) in
  let rec check_each_type main_type type_list =
   (match type_list with
    | [] -> true
    | hd::tl ->
     if(same_types main_type hd) then (check_each_type
↪ main_type tl)
     else raise
↪ Exceptions.Array_elements_not_all_same_type
   )
  in check_each_type main_type type_list

(* Creates list of sast structs storing information about
↪ constants for higher order function *)
let rec get_constants_info constant_list c_constant_list
↪ env =
 match constant_list with
 | [] -> c_constant_list
 | hd::tl ->
   (match hd with
    | Ast.Constant(id,e) ->
      let vtype = infer_type e env in
      (* Name of constant in defg gpu function*)
      let h_name =
↪ Ast.Identifier(generate_host_pointer_name ()) in
```

```
            (* Name of constant when input as an argument *)
            let a_name = Ast.Identifier(generate_arg_name ())in
            let v_type,env = convert_to_c_variable_type vtype env in
            (* Sast.type*)
            let constant_info = {
                variable_type = v_type;
                host_name = h_name;
                arg_name = a_name;
                kernel_name = id;
            } in get_constants_info tl (List.rev(constant_info::List.rev(c_constant_list))) env
      )

(* Creates list of sast structs storing information about info arrays from higher order function *)
let rec get_input_arrays_info input_arrays var_info_list env =
  match input_arrays with
    | [] -> var_info_list
    | hd::tl ->
      (
        match infer_type hd env with
          | Ast.Array(t,n) ->
              let h_name = Ast.Identifier(generate_host_pointer_name ())in
              let k_name = Ast.Identifier(generate_device_pointer_name ())in
              let a_name = Ast.Identifier(generate_arg_name ())in
              let vtype,env = convert_to_c_variable_type(infer_type hd env) env in
              let var_info = {
                  variable_type = vtype;
                  host_name = h_name;
                  kernel_name = k_name;
                  arg_name = a_name;
              }
            in get_input_arrays_info tl (List.rev(var_info::List.rev(var_info_list))) env
          | _ -> raise Exceptions.Nonarray_argument_passed_into_higher_order_function
      )

(* Creates sast struct storing information about return array from higher order function *)
let get_return_array_info kfunc_id length env =
  let f_info            = get_function_info kfunc_id env in
  let return_vtype,env  = convert_to_c_variable_type (Ast.Array(f_info.function_return_type,length)) env in
  let h_name              = Ast.Identifier(generate_host_pointer_name ())in
  let k_name              = Ast.Identifier(generate_device_pointer_name ())in
  let a_name              = Ast.Identifier(generate_arg_name ()) in
  let var_info            =  {
      variable_type       = return_vtype;
      host_name           = h_name;
      kernel_name         = k_name;
      arg_name            = a_name;
  } in
  var_info

(* Main function for creating the C map function (when we see a map function call) *)
let make_hof_c_fdecl hof_call env =
  let arr_length =
              let arr = infer_type (List.hd hof_call.input_arrays) env in
              (match arr with
              | Ast.Array(t,n) -> n
              | _ -> raise Exceptions.Not_an_array_expression)
  in
  match Utils.idtos(hof_call.hof_type) with
    | "map" ->
          let kfunc_name = Ast.Identifier(generate_map_ptx_function_name ()) in
         {
            higher_order_function_type = Ast.Identifier("map");
            higher_order_function_name = Ast.Identifier(generate_map_c_function_name ());
            applied_kernel_function = kfunc_name;
            higher_order_function_constants = get_constants_info hof_call.constants [] env;
            array_length = arr_length;
            input_arrays_info = get_input_arrays_info hof_call.input_arrays [] env;
            return_array_info = get_return_array_info (Utils.idtos(hof_call.kernel_function_name)) arr_length env;
            called_functions = [hof_call.kernel_function_name]
         }
    | "reduce" ->
          let kfunc_name = Ast.Identifier(generate_reduce_ptx_function_name ()) in
           {
              higher_order_function_type
 ↪  =  Ast.Identifier("reduce");
              higher_order_function_name
 ↪  =  Ast.Identifier(generate_reduce_c_function_name ());
              applied_kernel_function
 ↪  =  kfunc_name;
              higher_order_function_constants
 ↪  =  get_constants_info hof_call.constants [] env;
              array_length
 ↪  =  arr_length;
              input_arrays_info
 ↪  =  get_input_arrays_info hof_call.input_arrays [] env;
              return_array_info
 ↪  =  get_return_array_info
 ↪  (Utils.idtos(hof_call.kernel_function_name)) arr_length
 ↪  env;
              called_functions
 ↪  =  [hof_call.kernel_function_name]
           }
    | _ -> raise
 ↪  (Exceptions.Unknown_higher_order_function_call
 ↪  (Utils.idtos hof_call.hof_type))

(* TO IMPLEMENT
Converts c_kernel_variable_info to ptx_kernel_variable_info
 ↪  *)
let convert_to_register_declaration dtype id num_reg =
  {
      reg_type      = dtype;
      reg_id        = id;
      num_registers = num_reg;
  }

let hof_param_reg_counter = ref 0

let change_to_ptx_vdecl ckv_info  =
  let change_to_ptx_data_type sast_c_dtype =
    match sast_c_dtype with
      | Sast.Integer -> S32
      | Sast.Float   -> F32
      | Sast.Boolean -> Pred
      | Sast.Void -> Ptx_Void
      | _ -> raise Exceptions.NO_STRINGS_ALLOWED_IN_GDECL
  in
  let rec get_vtype sast_c_vtype =
    (match sast_c_vtype with
        | Sast.Primitive(p) ->
 ↪  Ptx_Primitive(change_to_ptx_data_type p)
        | Sast.Array(t,n) -> Ptx_Array(get_vtype t, n)
    )
  in
  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)))

(* Creates a ptx_fdecl based on the hof_c_fdecl*)
let make_hof_ptx_fdecl hof_c_fdecl hof env=
  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;
  ] in
  {
    ptx_higher_order_function_type                        =
↪ hof_c_fdecl.higher_order_function_type;
    ptx_higher_order_function_name                        =
↪ hof_c_fdecl.applied_kernel_function;
    ptx_applied_kernel_function                           =
↪ hof.kernel_function_name;
    ptx_higher_order_function_constants                   =
↪ List.map change_to_ptx_vdecl
↪ (hof_c_fdecl.higher_order_function_constants);
    ptx_array_length                                      =
↪ hof_c_fdecl.array_length;
    ptx_input_arrays_info                                 =
↪ List.map change_to_ptx_vdecl
↪ (hof_c_fdecl.input_arrays_info);
    ptx_return_array_info                                 =
↪ change_to_ptx_vdecl hof_c_fdecl.return_array_info;
    ptx_called_functions                                  =
↪ [hof.kernel_function_name];
    ptx_register_decls                                    =
↪ regs;
  }

let rec get_types args types env =
  match args with
    | [] -> types
    | hd::tl -> get_types tl (List.rev((infer_type hd
↪ env)::List.rev types)) env

let rec add_lists list_lists newlist=
  match list_lists with
  | [] -> newlist
  | hd::tl -> add_lists tl (newlist @ hd)
```

```

let rec convert_to_c_expression e env =
  let rec flatten_array e flattened_array env =
    (match e with
      | Ast.Array_Literal(e_list) ->
            (match List.hd e_list with
              | Ast.Array_Literal(e1_list) ->
                  let list_of_flattened_arrays = List.map
 ↪  (fun x-> flatten_array x [] env) e_list in
                  add_lists list_of_flattened_arrays []
              | _ ->  flattened_array @ (List.map (fun x ->
 ↪  fst(convert_to_c_expression x env)) e_list)
            )
      | _ -> raise Exceptions.Not_an_array_expression
    )
  in
    match e with
      | Ast.Function_Call(id,e_list) ->
        (* Check that function exists in environment *)
         if (is_function_in_scope (Utils.idtos id) env) =
 ↪  false then (raise (Exceptions.Function_not_defined
 ↪  (Utils.idtos id)));
        (* Check that function arguments match that of
 ↪  function declaration *)
        let f_info = (get_function_info (Utils.idtos id)
 ↪  env) in
        let f_arg_types = f_info.function_args in
              let check_args expected_arg_types f_args =
              List.map2 same_types expected_arg_types
 ↪  f_args in
        ignore(check_args f_arg_types (get_types e_list []
 ↪  env));
        (* Convert *)
        let c_e_list = List.map (fun x ->
 ↪  fst(convert_to_c_expression x env)) e_list
        in Sast.Function_Call(id,c_e_list),env
      | Ast.String_Literal(s) -> Sast.String_Literal(s),env
      | Ast.Integer_Literal(i) ->
 ↪  Sast.Integer_Literal(i),env
      | Ast.Boolean_Literal(b) ->
 ↪  Sast.Boolean_Literal(b),env
      | Ast.Floating_Point_Literal(f) ->
 ↪  Sast.Floating_Point_Literal(f),env
      | Ast.Array_Literal(e_list) ->
          (* Check all elements of the array are the same
 ↪  type *)
          let type_list = List.map (fun x -> infer_type x
 ↪  env) e_list in
          ignore(same_types_list type_list);
          (* Get array dimensions and pass to sast *)
          let arr = Ast.Array(infer_type (List.hd e_list)
 ↪  env ,List.length e_list) in
          let array_dim = get_array_dimensions arr [] in
```

```
     (* Convert *)
     let c_e_list = flatten_array e [] env in
     Sast.Array_Literal(c_e_list,array_dim),env
   | Ast.Identifier_Literal(id) ->
     if(check_already_declared (Utils.idtos id) env) =
↪ false then raise
↪ (Exceptions.Variable_not_found_in_scope ( Utils.idtos
↪ id))
     else Sast.Identifier_Literal(id),env
   | Ast.Cast(vtype, e) ->
     let c_vtype,env = convert_to_c_variable_type
↪ vtype env in
     let c_e,env = convert_to_c_expression e env in
     Sast.Cast(c_vtype,c_e),env
   | Ast.Unop(e,op) ->
     (match op with
      | Ast.Not ->
       if((infer_type e env)=
↪ Ast.Primitive(Ast.Boolean)) = false then raise
↪ (Exceptions.Type_mismatch("Must use boolean expression
↪ with boolean unop"))
       else
        let c_e,env = convert_to_c_expression e env
↪ in
        let c_op,env = convert_to_c_unop op env in
        Sast.Unop(c_e,c_op),env
      | _ ->
       if((infer_type e env) = (Ast.Primitive
↪ (Ast.String))) then raise
↪ (Exceptions.Cannot_perform_operation_on_string
↪ (Utils.unary_operator_to_string op))
       else
        let c_e,env = convert_to_c_expression e env
↪ in
        let c_op,env = convert_to_c_unop op env in
        Sast.Unop(c_e,c_op),env
     )
   | Ast.Ternary(e1,e2,e3) ->
    (*Check e1 and e3 match*)
    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 c_e1,env = convert_to_c_expression e1 env
↪ in
      let c_e2,env = convert_to_c_expression e2 env
↪ in
```

```
    let c_e3,env = convert_to_c_expression e3 env in
    Sast.Ternary(c_e1,c_e2,c_e3),env
 | Ast.Array_Accessor(e,e_list,is_lvalue) ->
  (* Check e is an array *)
   (match infer_type e env with
    | Ast.Array(t,n) -> ()
    | _ -> raise Exceptions.Not_an_array_expression);
   (* Check that e_list can access a*)
   ignore(List.map (fun x -> same_types (infer_type x env) (Ast.Primitive(Ast.Integer))) e_list);
   (* Convert *)
   let c_e,env = convert_to_c_expression e env  in
   let c_e_list = List.map (fun x -> fst(convert_to_c_expression x env)) e_list in
   let array_type = infer_type e env in
   let array_dims = get_array_dimensions array_type [] in
   let array_access = ((List.length array_dims) > (List.length e_list)) in
   Sast.Array_Accessor(c_e,c_e_list,is_lvalue,array_access),env
 | Ast.Binop(e1,op,e2) ->
   (* Check that expressions match *)
   if((same_types (infer_type e1 env) (infer_type e2 env)) = false) then raise (Exceptions.Type_mismatch "Binop doesn't match")
   else
    (match op with
     | Ast.And | Ast.Or | Ast.Xor ->
      (* Check that type is boolean if using boolean operator *)
      ignore(same_types (infer_type e1 env) (Ast.Primitive(Ast.Boolean)));
       let c_e1,env = convert_to_c_expression e1 env in
       let c_op,env = convert_to_c_binop op env in
       let c_e2,env = convert_to_c_expression e2 env in
       Sast.Binop(c_e1,c_op,c_e2),env
     | _ ->
      (* Check if type is string, array *)
       (match (infer_type e1 env) with
        | Ast.Primitive(t) -> if t = Ast.String then raise (Exceptions.Cannot_perform_operation_on_string (Utils.binary_operator_to_string op)) else ()
        | Ast.Array(t,n) -> raise (Exceptions.Cannot_perform_operation_on_array (Utils.binary_operator_to_string op))
       );
```

```
                  let c_e1,env = convert_to_c_expression e1 env in
                  let c_op,env = convert_to_c_binop op env in
                  let c_e2,env = convert_to_c_expression e2 env in
                  Sast.Binop(c_e1,c_op,c_e2),env
            )
      | Ast.Higher_Order_Function_Call(hof) ->
       (* Check that function exists in environment *)
        if (is_function_in_scope (Utils.idtos(hof.kernel_function_name)) env) = false then raise (Exceptions.Function_not_defined (Utils.idtos hof.kernel_function_name));
        (* Check that arrays are valid arrays *)
(*           let input_arrays = List.map (fun e -> infer_type e env) hof.input_arrays in
        let good_arrays = (List.iter same_types_list input_arrays) in *)
        (* Check that function arguments match that of function declaration *)
        let f_info = (get_function_info (Utils.idtos hof.kernel_function_name) env) in
        (* if f_info.function_type != Kernel_Device then raise (Exceptions.Higher_order_function_call_only_takes_defg_functions)
      else *)
        let expected_arg_types = f_info.function_args in
        let get_array_types arr =
            match arr with
                | Ast.Array(t,n) -> t
                | _ -> raise Exceptions.Invalid_input_argument_to_map
        in
        let f_arg_types = List.map get_array_types (get_types hof.input_arrays [] env) in
        let check_args expected_arg_types f_args =
        List.map2 same_types expected_arg_types f_args in
        ignore(same_types (List.length f_arg_types) (List.length expected_arg_types));
        ignore(check_args f_arg_types expected_arg_types);
        (*Check that constants match those unknown variables in the defg*)
        let retrive_constant_name c =
          match c with
            | Ast.Constant(id,e) -> Utils.idtos(id)
        in
        let hof_call_constants_names = List.map (retrive_constant_name) hof.constants in
        let hof_constants_names = List.map (fun x -> Utils.idtos(x)) f_info.unknown_variables in
          let rec check_constants hof_call_c hof_fdecl_c =
            match hof_fdecl_c with
            | [] -> true
            | hd::tl -> if (List.exists (fun s -> s = hd) hof_call_c) = false then raise Exceptions.Constants_missing_in_defg
              else check_constants hof_call_c tl
          in
        ignore(check_constants hof_call_constants_names hof_constants_names);
        (match Utils.idtos(hof.hof_type) with
            | "map" ->
                (*Add the c map function to the environment*)
                let hof_c_fdecl = make_hof_c_fdecl hof env in
                let hof_ptx_fdecl = make_hof_ptx_fdecl hof_c_fdecl hof env in
                let env = update_hof_lists hof_c_fdecl hof_ptx_fdecl env in
                (* Convert *)
                Sast.Function_Call(hof_c_fdecl.higher_order_function_name,(List.map (fun x -> fst(convert_to_c_expression x env)) (List.rev hof.input_arrays))),env
            (* | "reduce" ->
                in Sast.FunctionCall(c_ma) *)
            | _ -> raise (Exceptions.Unknown_higher_order_function_call (Utils.idtos(hof.hof_type))))

(* Stack Algorithm *)
(*
    PUSH EXPRESSION STACK
    RECURSE
    SAVE LAST PTX ID
    POP EXPRESSION STACK
    PUSH LAST SAVED ONTO HIGHER STACK
*)
(* FILL IN WITH SEMANTIC CHECKING !!!!!!!!!! *)

let rec bool_sum bool_list sum =
  match bool_list with
    | [] -> sum
    | hd::tl ->
        let num = (if hd = true then 1 else 0) in
        bool_sum (tl) (sum + num)

let rec is_constant expr =
    match expr with
    | Ast.Integer_Literal(i) -> true
    | Ast.Floating_Point_Literal(f) -> true
    | Ast.Boolean_Literal(f) -> true (*Maybe want to change*)
    | Ast.Array_Literal(e_list) -> ((bool_sum (List.map is_constant e_list) 0) = (List.length e_list))
    | _ -> false

let rec convert_to_ptx_expression e env =
  match e with
    | Ast.String_Literal(s) -> raise Exceptions.NO_STRINGS_ALLOWED_IN_GDECL;
    | Ast.Higher_Order_Function_Call(hof) -> raise Exceptions.No_Hof_Allowed
    | Ast.Integer_Literal(i) ->
          let env = update_expression_stack (Sast.Ptx_Signed_Integer(i)) env in
          Sast.Ptx_Block([Ptx_Empty]), env
    | Ast.Boolean_Literal(b) ->
          let env = update_expression_stack (Sast.Ptx_Predicate(if b = true then 1 else 0)) env in
          Sast.Ptx_Block([Ptx_Empty]),env
    | Ast.Floating_Point_Literal(f) ->
          let env = update_expression_stack (Sast.Ptx_Signed_Float f) env in
          Sast.Ptx_Block([Ptx_Empty]),env
    | Ast.Identifier_Literal(i) ->
          if(check_already_declared (Utils.idtos i) env) = false then raise (Exceptions.Variable_not_found_in_scope ( Utils.idtos i))
          else
          let v_info = get_variable_info (Utils.idtos i) env in
          let is_array = match v_info.vtype with | Ast.Primitive(p) -> false | Ast.Array(t,n) -> true in
          let ptx_lit = Sast.Ptx_Identifier_Literal(make_ptx_id i (get_reg_type (v_info.vtype)) v_info.register_number true is_array) in
          let env = update_expression_stack ptx_lit env in
          Sast.Ptx_Block([Ptx_Empty]),env
    | Ast.Binop(e1,o,e2) ->
          if((same_types (infer_type e1 env) (infer_type e2 env)) = false) then raise (Exceptions.Type_mismatch "Binop doesn't match")
          else
            (match o with
              | Ast.And | Ast.Or | Ast.Xor ->
                (* Check that type is boolean if using boolean operator *)
                ignore(same_types (infer_type e1 env) (Ast.Primitive(Ast.Boolean)));
              | _ ->
                (* Check if type is string, array *)
                  (match (infer_type e1 env) with
                    | Ast.Primitive(t) -> if t = Ast.String then raise (Exceptions.Cannot_perform_operation_on_string (Utils.binary_operator_to_string o)) else ()
                    | Ast.Array(t,n) -> raise (Exceptions.Cannot_perform_operation_on_array (Utils.binary_operator_to_string o))
                  );
              );
        let vtype = infer_type e1 env in
          (* Push stack *)
          let env = push_expression_stack env in
        let ptx_e1, env = convert_to_ptx_expression e1 env in
        let ptx_e2, env = convert_to_ptx_expression e2 env in
          (* We now have two values on our current stack, resolve *)
          (* For binop, we need to generate a third register to store value of addition*)
          let reg_name,reg_num = generate_reg vtype in
        let ptx_lit = Sast.Ptx_Identifier_Literal(make_ptx_id (Ast.Identifier("")) reg_name reg_num true false) in
              let ptx_binop,env = convert_to_ptx_binop o env in
              let ptx_vtype,env = convert_to_ptx_variable_type vtype env in
        let resolve = Sast.Ptx_Binop(ptx_binop,ptx_vtype,ptx_lit,get_from_expression_stack 1 env , get_from_expression_stack 0 env) in
            (* Pop stack *)
            let env = pop_expression_stack env in
            (* Push the ptx_lit on current stack *)
            let env = update_expression_stack ptx_lit env in

            let ptx_expr_block = [ptx_e1;ptx_e2;resolve] in
        Sast.Ptx_Block(ptx_expr_block),env
    | Ast.Unop(e,o) ->
          (match o with
            | Ast.Not ->
              if((infer_type e env)= Ast.Primitive(Ast.Boolean)) = false then raise (Exceptions.Type_mismatch("Must use boolean expression with boolean unop"))
            | _ -> ());
              if((infer_type e env) = (Ast.Primitive (Ast.String))) then raise (Exceptions.Cannot_perform_operation_on_string (Utils.unary_operator_to_string o))
              else
              let vtype = infer_type e env in
              (* Push stack *)
              let env = push_expression_stack env in
          let ptx_e,env = convert_to_ptx_expression e env in
              (* We now have a value on our current stack, resolve *)
              (* Unop requires a generated second register *)
              let reg_name,reg_num = generate_reg vtype in
          let ptx_lit = Sast.Ptx_Identifier_Literal(make_ptx_id (Ast.Identifier "") reg_name reg_num true false) in
              let ptx_unop,env = convert_to_ptx_unop o env in
              let ptx_vtype,env = convert_to_ptx_variable_type vtype env in
          let resolve = Sast.Ptx_Unop(ptx_unop,ptx_vtype,ptx_lit,get_from_expression_stack 0 env) in
          (* Pop stack *)
              let env = pop_expression_stack env in
          (* Push the ptx_lit on current stack *)
              let env = update_expression_stack ptx_lit env in
            let ptx_expr_block = [ptx_e;resolve] in
        Sast.Ptx_Block(ptx_expr_block),env
    | Ast.Array_Literal(e_list) ->
          (* Check all elements of the array are the same type *)
          let type_list = List.map (fun x -> infer_type x env) e_list in
          ignore(same_types_list type_list);
          (* Get array dimensions and pass to sast *)
          let arr = Ast.Array(infer_type (List.hd e_list) env ,List.length e_list) in
          let array_dim = get_array_dimensions arr [] in
        (* Check that all the expressions are primitives because PTX doesn't allow expressions *)
          let valid_array = (bool_sum (List.map is_constant e_list) 0) = List.length(e_list) in
          if (valid_array = false) then raise Exceptions.Defg_arrays_must_be_defined_with_constants
          else
            (* Now we know that the array list is only full of array lits, basically just convert all of them*)
            (* Push on stack *)
            let env = push_expression_stack env in
                let lit_list, env = convert_list convert_to_ptx_expression e_list [] env in
                (* For an array literal, we will push the entire thing onto the stack *)
                let rec get_elements stack alist = match stack with [] -> alist| hd::tl -> get_elements tl (hd::alist) in
                let array_lit = Sast.Ptx_Array_Literal ((get_elements(List.hd env.expression_stack) [])) in
            let env = update_expression_stack array_lit env in
            Sast.Ptx_Block([Ptx_Empty]),env
    | Ast.Function_Call(id, e_list) ->
        if (is_function_in_scope (Utils.idtos id) env) = false then (raise (Exceptions.Function_not_defined (Utils.idtos id)));
        (* Check that function arguments match that of function declaration *)
        let f_info = (get_function_info (Utils.idtos id) env) in
        let f_arg_types = f_info.function_args in
              let check_args expected_arg_types f_args = List.map2 same_types expected_arg_types f_args in
              ignore(check_args f_arg_types (get_types e_list [] env));
        let rtype = f_info.function_return_type in
        (* Push stack *)
        let env = push_expression_stack env in
           let lit_list, env = convert_list convert_to_ptx_expression e_list [] env in
           (* For a function call, need to define a return register *)
            let reg_name,reg_num = generate_reg rtype in
            let ptx_lit = Sast.Ptx_Identifier_Literal(make_ptx_id (Ast.Identifier "") reg_name reg_num true false) in
            let rec get_elements stack alist = match stack with [] -> alist| hd::tl -> get_elements tl (hd::alist) in
            let expr = match rtype with
              | Ast.Primitive(Ast.Void) -> Sast.Ptx_Empty_Call(id,(get_elements(List.hd env.expression_stack) []))
              | _ -> Sast.Ptx_Call(ptx_lit, id, (get_elements(List.hd env.expression_stack) []))
            in
          let env = pop_expression_stack env in
          let env = update_expression_stack ptx_lit env in
        expr,env
        (* Pop stack *)
          (* For function call, we resolve the expressions and then *)
    | Ast.Cast(vtype,e)-> raise Exceptions.Casting_not_allowed_in_defg
    | Ast.Array_Accessor(e,e_list,b)-> raise Exceptions.C'est_La_Vie
    | Ast.Ternary(e1,e2,e3) -> raise Exceptions.C'est_La_Vie
```

```
       (*  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
```

```
              | Ast.Identifier_Literal(id) ->
                  if (check_already_declared (Utils.idtos(id)) env) = false then raise (Exceptions.Name_not_found (Utils.idtos id))
                  else
                    (* Check same types*)
                    ignore(same_types (get_variable_type (Utils.idtos id) env) (infer_type e2 env));
                    (*Convert*)
                    let c_e1, env = convert_to_c_expression e1 env in
                    let c_e2, env = convert_to_c_expression e2 env in
                    Sast.Assignment(c_e1,c_e2),env
              | Ast.Array_Accessor(e,e_list,is_lvalue)->
                    (match e with
                      | Ast.Identifier_Literal(id) ->
                          if (check_already_declared (Utils.idtos id) env )= false then raise (Exceptions.Name_not_found (Utils.idtos id))
                          else
                            (* Check same types*)
                              let arr = get_variable_type (Utils.idtos id) env in ignore(same_types (get_array_el_type arr (List.length e_list)) (infer_type e2 env));
                              (*Convert*)
                              let c_e1, env = convert_to_c_expression e1 env in
                              let c_e2,env = convert_to_c_expression e2 env in
                              Sast.Assignment(c_e1,c_e2),env
                      | _ -> (raise Exceptions.Cannot_assign_expression)
                    )
              | _ -> raise Exceptions.Cannot_assign_expression

(* TO IMPLEMENT *)
(*Stack algorithm for conversion:
  Push stack
  Recurse on expression, Every expression will push a new stack, and when resolved will pop its stack
  Save last ptx_id we obtain from resolving -> this will be different for different expressions -> this is done in expressions
  Pop stack

  Push last ptx_id onto new stack
  Return Sast.datatype, new env
*)

let get_vdecl_parts vdecl =
    (match vdecl with
    | Ast.Variable_Declaration(t,i) -> i,t)

let convert_to_ptx_variable_statement vstmt env =
    match vstmt with
      | Ast.Declaration(vdecl) ->
          let ptx_vdecl,env =  convert_to_ptx_vdecl vdecl
 ↪  env in
          Sast.Ptx_Variable_Declaration(ptx_vdecl),env
      | Ast.Initialization(vdecl, e) ->
          let ptx_vdecl,env = convert_to_ptx_vdecl vdecl
 ↪  env in
                (* Push scope for expression stack *)
                  let env = push_expression_stack env in
              let vdecl_expr =
 ↪  Sast.Ptx_Variable_Declaration(ptx_vdecl) in
              (* Must save ptx value for vdecl on the stack
 ↪  *)
                    let id,vtype = get_vdecl_parts vdecl in
                    let v_info = get_variable_info
 ↪  (Utils.idtos id) env in
                    let ptx_lit =
 ↪  Sast.Ptx_Identifier_Literal(make_ptx_id id
 ↪  (get_reg_type v_info.vtype) v_info.register_number true
 ↪  false) in
                    let env = update_expression_stack
 ↪  ptx_lit env in
              let ptx_e,env = convert_to_ptx_expression e
 ↪  env in
              (* convert_to_ptx_expression has saved a
 ↪  value in the stack. Let us fetch it and resolve*)
              let resolve =
 ↪  Sast.Ptx_Move(fst(convert_to_ptx_variable_type vtype
 ↪  env),get_from_expression_stack 1 env,
 ↪  get_from_expression_stack 0 env) in
              (* Pop the stack *)
                    let env = pop_expression_stack env in
                    let expr_block =
 ↪  [vdecl_expr;ptx_e;resolve] in
            Sast.Ptx_Block(expr_block),env
      | Ast.Assignment(e1, e2) ->
          match e1 with
            | Ast.Identifier_Literal(id) ->
                (* Ast checking...*)
                if (check_already_declared
 ↪  (Utils.idtos(id)) env) = false then raise
 ↪  (Exceptions.Name_not_found (Utils.idtos id))
                else
                  ignore(same_types (get_variable_type
 ↪  (Utils.idtos id) env) (infer_type e2 env));
                   let env = push_expression_stack env in
              (* Must save ptx value for vdecl on the stack
 ↪  *)
```

```
                  let v_info = get_variable_info (Utils.idtos id) env in
                  (* Update vmap *)
                  let ptx_id = make_ptx_id id (get_reg_type v_info.vtype) v_info.register_number true false in
                  let env = update_variable_register (Utils.idtos id) ptx_id.reg_num env in
                  let ptx_lit = Sast.Ptx_Identifier_Literal(ptx_id) in
                  let env = update_expression_stack ptx_lit env in
            let ptx_e,env = convert_to_ptx_expression e2 env in
            (* convert_to_ptx_expression has saved a value in the stack. Let us fetch it and resolve*)
            let resolve = Sast.Ptx_Move(fst(convert_to_ptx_variable_type v_info.vtype env),get_from_expression_stack 1 env, get_from_expression_stack 0 env) in
            (* Pop the stack *)
                  let env = pop_expression_stack env in
                  let expr_block = [ptx_e;resolve] in
          Sast.Ptx_Block(expr_block),env
          | Ast.Array_Accessor(e,e_list,is_lvalue)-> raise Exceptions.C'est_La_Vie
                  (* (match e with
                  | Ast.Identifier_Literal(id) ->
                      if (check_already_declared (Utils.idtos id) env )= false then raise (Exceptions.Name_not_found (Utils.idtos id))
                      else
                        (* Check same types*)
                          let arr = get_variable_type (Utils.idtos id) env in ignore(same_types (get_array_el_type arr (List.length e_list)) (infer_type e2 env));

                          (* This case is weird becuase we know e is an identifier literal, and that it is an array, so we can gets its information to make a ptx_id from get_variable_info *)
                          (* Don't need to push pop  - special case for assign *)
                          (* We get the variable information for the array *)
                          let v_info = get_variable_info id env in
(* NEED TO RESOLVE*)      let arr_ptx_id = make_ptx_id id (get_reg_type v_info.vtype) v_info.register_number true in


```

```
                            We need to create a load
  ↪ statement
                            let reg_name, reg_num =
  ↪ generate_reg vtype in let new_v_info = { vtype =
  ↪ v_info.vtype; register_number = reg_num;} in
                            let new_vmap = Variable_Map.add
  ↪ (Utils.idtos id) new_v_info (List.hd
  ↪ env.variable_scope_stack) in
                            let env = update_scope new_vmap
  ↪ env in
                            let ptx_e = make_ptx_id id
  ↪ reg_name reg_num true in
                            (*Push expression stack*)
                            let env = push_expression_stack
  ↪ env in
                            let e1_stmt_block,env =
  ↪ convert_to_c_expression e1 env in
                            let e2_stmt_block,env =
  ↪ convert_to_c_expression e2 env in

  ↪ Sast.Load(Sast.Global,c_e1,c_e2),env
                            (*Pop expression stack*)
                            let env = pop_expression_stack
  ↪ env in
                            Sast.Block(),env
                    | _ -> (raise
  ↪ Exceptions.Cannot_assign_expression)
                  )   *)
            | _ -> raise
  ↪ Exceptions.Cannot_assign_expression



(* Converts global vstmt list into c vstmt list *)
let rec convert_to_c_variable_statement_list vstmt_list
  ↪ c_vstmt_list env =
     match vstmt_list with
      | [] -> (c_vstmt_list,env)
      | hd::tl ->
          let c_vstmt, env =
  ↪ convert_to_c_variable_statement hd env in
          convert_to_c_variable_statement_list tl
  ↪ (List.rev(c_vstmt::List.rev(c_vstmt_list))) env


let rec convert_to_c_statement stmt env =
  match stmt with
    | Ast.Variable_Statement(vstmt) ->
        let c_vstmt,env = convert_to_c_variable_statement
  ↪ vstmt env in
        Sast.Variable_Statement(c_vstmt),env
    | Ast.Expression(e) ->
        let c_e,env = convert_to_c_expression e env in
        Sast.Expression(c_e),env
      | Ast.If(e,stmt1,stmt2) ->
        (* Check that e is a boolean expression *)
        ignore(same_types (infer_type e env)
  ↪ (Ast.Primitive(Ast.Boolean)));
        (* Convert*)
        let c_e, env = convert_to_c_expression e env in
        let c_stmt1,env = convert_to_c_statement stmt1 env
  ↪ in
        let c_stmt2,env = convert_to_c_statement stmt2 env
  ↪ in
        Sast.If(c_e,c_stmt1,c_stmt2),env
      | Ast.While(e,stmt) ->
        ignore(same_types (infer_type e env)
  ↪ (Ast.Primitive(Ast.Boolean)));
        (* Check that e is a boolean expression *)
        let c_e, env = convert_to_c_expression e env in
        let c_stmt,env = convert_to_c_statement stmt env in
        Sast.While(c_e,c_stmt),env
      | Ast.For(stmt1,e,stmt2,stmt3) ->
        (* Check that stmt1 is an initialization expression
  ↪ *)
         (match stmt1 with
            | Ast.Variable_Statement(vstmt) ->
             (match vstmt with
              | Ast.Assignment(e1,e2) -> ()
              | Ast.Initialization(vdecl,e) -> ()
              | _ -> raise
  ↪ Exceptions.Invalid_statement_in_for)
            | _ -> raise
  ↪ Exceptions.Invalid_statement_in_for);

        (* Convert *)
        let env = push_scope env in
        let c_stmt1,  env = convert_to_c_statement  stmt1
  ↪ env  in
        (* Check that e is a boolean expression *)
        ignore(same_types (infer_type e env)
  ↪ (Ast.Primitive(Ast.Boolean)));
        let c_e,    env = convert_to_c_expression  e
  ↪ env  in
        let c_stmt2,  env = convert_to_c_statement  stmt2
  ↪ env  in
        let c_stmt3,  env = convert_to_c_statement  stmt3
  ↪ env  in
        let env = pop_scope env in
        Sast.For(c_stmt1,c_e,c_stmt2,c_stmt3),env
      | Ast.Return(e) ->
        let c_e, env = convert_to_c_expression e env in
        Sast.Return(c_e),env
      | Ast.Return_Void -> Sast.Return_Void,env
      | Ast.Continue -> Sast.Continue,env
      | Ast.Break -> Sast.Break,env
```

```
  | Ast.Block(stmt_list) ->
    (* Check that nothing follows a return , break, or continue in a block *)
    if (good_statement_order stmt_list) = false then raise Exceptions.Have_statements_after_return_break_continue
    else
    (* Convert *)
    let c_stmt_list,env = convert_list convert_to_c_statement stmt_list [] env in
     Sast.Block(c_stmt_list),env

let rec convert_to_ptx_statement stmt env =
 match stmt with
  | Ast.Variable_Statement(v) -> convert_to_ptx_variable_statement v env
  | Ast.Expression(e) -> convert_to_ptx_expression e env
  | Ast.Return_Void -> Sast.Ptx_Return_Void, env
  | Ast.Return(e) ->
    let vtype = infer_type e env in
    let env = push_expression_stack env in
     let ptx_e, env = convert_to_ptx_expression e env in
     let rlit = env.return_lit in
    let expr = Sast.Ptx_Store(Sast.Global,fst(convert_to_ptx_variable_type vtype env),rlit,get_from_expression_stack 0 env) in
    let expr_block = [expr;Sast.Ptx_Return_Void] in
    let env = pop_expression_stack env in
    Sast.Ptx_Block(expr_block),env
  | Ast.Block(stmt_list) ->
    let expr_block, env = convert_list convert_to_ptx_statement stmt_list [] env in
    Sast.Ptx_Block(expr_block),env
  | Ast.If(e, s1, s2) -> raise Exceptions.C'est_La_Vie
(*               let env = push_expression_stack env in
              let vtype = infer_type e env in
              let ptx_e,env = convert_to_ptx_expression e env in
              (*  Create a literal referencing the predicate *)
              let ptx_lit = Sast.Ptx_Identifier_Literal(make_ptx_id (get_reg_type vtype) (get_predicate_counter ()) true false) in
              let env = update_expression_stack ptx_lit env in
          let bool_expr = Sast.Block([ptx_e]) in
              let env = pop_expression_stack env in
          let branch = Sast.Branch(get_from_expression_stack 0 env,generate_subroutine_name()) in
              let

          Sast.Ptx_Block(expr_block),env *)
    | Ast.While(e, s) -> raise Exceptions.C'est_La_Vie
    | Ast.For(s1, e, s2, s3) -> raise Exceptions.C'est_La_Vie
    | Ast.Continue -> raise Exceptions.C'est_La_Vie
    | Ast.Break -> raise Exceptions.C'est_La_Vie



let convert_to_c_param vdecl env  =
    match vdecl with
      | Ast.Variable_Declaration(vtype,id) ->
          if(check_already_declared (Utils.idtos id) env) = true then raise (raise (Exceptions.Variable_already_declared (Utils.idtos(id))))
          else
            let v_info = {
              vtype = vtype;
              register_number = 0;
            }
            in
            let updated_scope = Variable_Map.add (Utils.idtos id) v_info (List.hd env.variable_scope_stack) in
            let env = update_scope updated_scope env in
            let c_vtype, env = convert_to_c_variable_type vtype env in
            Sast.Variable_Declaration(c_vtype,id),env

let rec check_rtype rtype body env =
    match body with
    | [] -> ()
    | hd::tl->
        match hd with
          | Ast.Return_Void -> if(rtype != Ast.Primitive(Ast.Void)) then raise Exceptions.Missing_return_type
                            else check_rtype rtype tl env
          | Ast.Return(e) ->
                            if (same_types (infer_type e env) rtype) = false then raise Exceptions.Return_type_doesn't_match
                            else check_rtype rtype tl env
          | _ -> check_rtype rtype tl env

(* Converts from fdecl to c_fdecl *)
let convert_to_c_fdecl fdecl env =
    if (is_function_in_scope (Utils.idtos fdecl.name) env) = true then (raise Exceptions.Function_already_declared)
    else
      let vdecl_to_param vdecl =
        match vdecl with
          | Ast.Variable_Declaration(vtype,id) -> vtype
      in
      (* Add to function map*)
      (let host_func_info = {
          function_type = Host;
          function_name = fdecl.name;
          function_return_type = fdecl.return_type;
          function_args = List.map vdecl_to_param
 ↪ fdecl.params;
          dependent_functions = [];
          unknown_variables = [];
      }
      in
      let env = update_host_fmap host_func_info env in
      (* Push new scope for function *)

      let env = push_scope env in
      (* Do conversion while passing enviroment *)
      let return_type,  env    = convert_to_c_variable_type
 ↪ fdecl.return_type env in
      let params,       env    = convert_list
 ↪ convert_to_c_param          (List.rev fdecl.params)   []
 ↪ env in
      let body,         env    = convert_list
 ↪ convert_to_c_statement     fdecl.body    [] env in
      let c_fdecl = {
        c_fdecl_return_type = return_type;
        c_fdecl_name = fdecl.name;
        c_fdecl_params = params;
        c_fdecl_body = body;
      }
      in
      check_rtype fdecl.return_type fdecl.body env;
      (* Pop the variable scope for the function *)
      let env = pop_scope env in
      c_fdecl, env)


let convert_rtype_to_ptx_vdecl rtype env =
  let rname = Ast.Identifier( generate_ptx_return_name ())
 ↪ in
  let reg_name,reg_num = generate_reg rtype in
  let is_array = match rtype with | Ast.Primitive(p) ->
 ↪ false | Ast.Array(t,n) -> true in
  let ptx_id = make_ptx_id rname reg_name reg_num false
 ↪ false in
  let env = update_return_lit
 ↪ (Sast.Ptx_Identifier_Literal(ptx_id)) env in
  (Sast.Ptx_Vdecl(Sast.Param,fst
 ↪ (convert_to_ptx_variable_type rtype env), ptx_id)),env


```

```

let convert_to_ptx_fdecl fdecl env =
     if (is_function_in_scope (Utils.idtos fdecl.name) env) = true then (raise Exceptions.Function_already_declared)
    else
      let vdecl_to_param vdecl =
        match vdecl with
          | Ast.Variable_Declaration(vtype,id) -> vtype
      in
      (* Add to function map*)
      (let kernel_func_info = {
          function_type = Kernel_Device;
          function_name = fdecl.name;
          function_return_type = fdecl.return_type;
          function_args = List.map vdecl_to_param fdecl.params;
          dependent_functions = [];
          unknown_variables = [];
      }
      in
      let env = update_kernel_fmap kernel_func_info env in
      (* Push new scope for function *)
      let env = push_scope env in
      (* Convert sections of the function *)
      let return_type, env      = convert_to_ptx_variable_type fdecl.return_type env in
      let params,env            = convert_list convert_to_ptx_param (List.rev fdecl.params) [] env in
      let output, env           = convert_rtype_to_ptx_vdecl fdecl.return_type env in
      let body,         env     = convert_list convert_to_ptx_statement fdecl.body   [] env in
      let registers,    env     =  [
        convert_to_register_declaration (S32)  "si" !signed_int_counter;
        convert_to_register_declaration (F32)  "fl" !signed_float_counter;
        convert_to_register_declaration (Pred)  "pr" !predicate_counter;
      ],  env in
      check_rtype fdecl.return_type fdecl.body env;
      (* Create function item *)
      let ptx_fdecl = {
        ptx_fdecl_type = Sast.Device_Function;
        ptx_fdecl_name = fdecl.name;
        ptx_fdecl_input_params = params;
        ptx_fdecl_return_param = output;
        register_decls = registers;
        ptx_fdecl_body = body;
      }
      in
      (* Pop the variable scope for the function *)
      let env = pop_scope env in
```

```
   ptx_fdecl, env)

(* Converts a list of function declarations to ptx and c
 ↪ functions *)
let rec convert_fdecl_list fdecl_list ptx_fdecl_list
 ↪ c_fdecl_list env =
  match fdecl_list with
   | [] -> (ptx_fdecl_list,c_fdecl_list,env)
   | hd::tl ->
    ( match hd.is_kernel_function with
     | false ->
       let c_fdecl, env = convert_to_c_fdecl hd env
 ↪ in
       convert_fdecl_list tl ptx_fdecl_list
 ↪ (List.rev(c_fdecl::List.rev(c_fdecl_list))) env
     | true ->
       let ptx_fdecl, env = convert_to_ptx_fdecl hd
 ↪ env in
       convert_fdecl_list tl
 ↪ (List.rev(ptx_fdecl::List.rev(ptx_fdecl_list)))
 ↪ c_fdecl_list env
    )

(* Main function for converting ast to sast *)
let convert ast env =
  let vstmt_list,env                      = convert_list
 ↪ convert_to_c_variable_statement (fst(ast)) [] env in
  let ptx_fdecl_list,c_fdecl_list, env    =
 ↪ convert_fdecl_list (snd(ast)) [] [] env in
  let sast                                =
 ↪ (vstmt_list,ptx_fdecl_list,(env.hof_ptx_function_list),(env.hof_c_funct
 ↪ in
  sast

(* Main function for Sast *)
let analyze ast =
 let env = init_env in
 let sast = convert ast env in
 sast
```

codegen_c.ml

```
open Sast
(* open Exceptions *)
(* open Codegen_ptx *)

(* For sprintf *)
open Printf

(*-------------------------------------Generating Functions-------------------------------------*)

(* 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 *)
let generate_list generate_func concat mylist =
  let list_string = String.concat concat (List.map generate_func mylist) in
  sprintf "%s" list_string

(* Generate operators *)
let generate_binary_operator operator  =
  let op = match operator with
    | Add -> "+"
    | Subtract -> "-"
    | Multiply -> "*"
    | Divide -> "/"
    | Modulo -> "%"
    | And -> "&&"
    | Or -> "||"
    | Xor -> "^"
    | Equal -> "=="
    | Not_Equal -> "!="
    | Greater_Than -> ">"
    | Less_Than -> "<"
    | Greater_Than_Equal -> ">="
    | Less_Than_Equal -> "<="
    | Bitshift_Right -> ">>"
    | Bitshift_Left -> "<<"
  in
  sprintf "%s" op

let generate_unary_operator operator =
  let op = match operator with
    | Not -> "!"
    | Negate -> "-"
    | Plus_Plus -> "++"
    | Minus_Minus -> "--"
  in sprintf "%s" op
```

```
(* Generate data type*)
let generate_data_type dtype =
    let data_type = match dtype with
        | String -> "char *"
        | Unsigned_Byte -> "unsigned char"
        | Byte -> "signed char"
        | Unsigned_Integer -> "unsigned int"
        | Integer -> "int"
        | Unsigned_Long -> "unsigned long"
        | Long -> "long"
        | Float -> "float"
        | Double -> "double"
        | Boolean -> "bool"
        | Void -> "void"
    in sprintf "%s" data_type

(* Generate variable type *)
let rec generate_variable_type variable_type  =
  let vtype = match variable_type with
    | Primitive(p) -> generate_data_type p
    | Array(t,n) ->
      (match t with
        | Array(t1,n1) -> generate_variable_type t1
        | Primitive(p) -> generate_data_type p)
  in sprintf "%s" vtype

(* Generate id *)
let generate_id id  =
  let id_string = Utils.idtos(id) in sprintf "%s" id_string
(*   match id_string with
    | "print" -> sprintf "printf"
    | _ as identifier -> sprintf identifier *)

 (* Generates CUDA device pointer *)
 let generate_device_ptr ptr_name =
    sprintf "CUdeviceptr " ^ ptr_name ^ ";"

 (* Generates CUDA memory allocation from host to device *)
 (* Fill in with VLC_Array*)
 let generate_mem_alloc_statement_host_to_device arr_info
 ↪  arr_length=
    sprintf "checkCudaErrors(cuMemAlloc(&" ^
 ↪  Utils.idtos(arr_info.kernel_name) ^ ", sizeof(" ^
 ↪  (generate_variable_type arr_info.variable_type) ^ ")*"
 ↪  ^ string_of_int arr_length ^ "));"

 let generate_mem_alloc_host_to_device fcall =
    let rec create_list mylist length element = if length >
 ↪  0 then create_list (element::mylist) (length-1) element
 ↪  else mylist in
    let mem_alloc_string =
    String.concat "\n" (List.map2 generate_mem_alloc_statement_host_to_device fcall.input_arrays_info (create_list [] (List.length fcall.input_arrays_info) fcall.array_length)) in
    sprintf "%s" mem_alloc_string

(* Generates CUDA copying from host to device*)
 let generate_mem_cpy_statement_host_to_device arr_info arr_length =
    let mem_cpy_string  =
      "checkCudaErrors(cuMemcpyHtoD("^ Utils.idtos(arr_info.kernel_name) ^", " ^ Utils.idtos(arr_info.host_name) ^ ", sizeof(" ^ (generate_variable_type arr_info.variable_type) ^ ")*" ^ string_of_int arr_length ^ "));\n" in
    sprintf "%s" mem_cpy_string

 let generate_mem_cpy_host_to_device fcall =
    let rec create_list mylist length element = if length > 0 then create_list (element::mylist) (length-1) element else mylist in
    let mem_cpy_string = String.concat "\n" (List.map2 generate_mem_cpy_statement_host_to_device fcall.input_arrays_info (create_list [] (List.length fcall.input_arrays_info) fcall.array_length)) in
    sprintf "%s" mem_cpy_string

(* Generates CUDA statement for kernel params *)
 let generate_kernel_params arr_info =
    let rec get_kernel_names a_info_list name_list =
      match a_info_list with
        | [] -> name_list
        | hd::tl -> get_kernel_names tl (hd.kernel_name::name_list)
    in
    let kernel_names = (get_kernel_names arr_info []) in
    let kernel_param_string = generate_list generate_id ", &" kernel_names in
    sprintf "void *KernelParams[] = { &" ^ kernel_param_string ^ "};"

(* Generate CUDA memory cleanup *)
let generate_mem_cleanup arr_info =
    sprintf "checkCudaErrors(cuMemFree("^ Utils.idtos(arr_info.kernel_name) ^ "));"

(* Generates variable declaration statements *)
let generate_vdecl d  =
   match d with
  | Variable_Declaration(vtype,id) ->
      match vtype with
        | Array(t,n) -> sprintf "vlcarray fill"
            (* Fill in with VLC_Array*)
            (* let array_dimensions= (get_array_dimensions t [n]) in
            Environment.combine  [
                Generator(generate_variable_type t);
                Verbatim(" ");
                Generator(generate_id d.name);
                (* Get the array dimensions *)
                Verbatim("[");
                Verbatim(String.concat "][" (List.map string_of_int array_dimensions));
                Verbatim("]")
            ] *)
        | Primitive(p) ->
            let param_string = (generate_data_type p) ^ " " ^ (generate_id id) in
            sprintf "%s" param_string
(*          | _ -> raise Exceptions.Unknown_variable_type
  | _ -> raise Exceptions.Unknown_type_of_vdecl *)

let generate_param d =
  match d with
    | Variable_Declaration(vtype,id) ->
      match vtype with
        | Array(t,n) -> sprintf "vlcarray fill"
            (* Fill in with VLC_Array*)
            (* let array_dimensions= (get_array_dimensions t [n]) in
            Environment.combine  [
                Generator(generate_variable_type t);
                Verbatim(" ");
                Generator(generate_id d.name);
                (* Get the array dimensions *)
                Verbatim("[");
                Verbatim(String.concat "][" (List.map string_of_int array_dimensions));
                Verbatim("]")
            ] *)
        | Primitive(p) ->
            let param_string = (generate_data_type p) ^ " " ^ (generate_id id) in
            sprintf "%s" param_string
(*          | _ -> raise Exceptions.Unknown_variable_type
    | _ -> raise Exceptions.Unknown_type_of_param *)

 (* Generate expressions - including higher order function calls - and constants *)
let rec generate_expression expression  =
  let expr = match expression with
    | Function_Call(id, expr_list) ->
        (generate_id id) ^ "(" ^ generate_list generate_expression "," expr_list ^ ")"
    | Higher_Order_Function_Call(fcall) -> generate_higher_order_function_call fcall
    | Kernel_Function_Call(kfcall) ->
↪  generate_kernel_function_call kfcall
    | String_Literal(s) ->
        "\"" ^ s ^ "\""
    | Integer_Literal(i) ->
        string_of_int i
    | Boolean_Literal(b) ->
        string_of_bool b
    | Floating_Point_Literal(f) ->
        string_of_float f
    | Array_Literal(s) ->
        "vlcarray fill"
        (* Fill in with VLC_Array*)
        (* sprintf "{" ^ (generate_expression_list s) ^ "}"
↪  *)
    | Identifier_Literal(id) ->
        (generate_id id)
    | Cast(vtype,e) ->
        "(" ^ (generate_variable_type vtype) ^ ")" ^
↪  (generate_expression e)
    | Binop(e1, o, e2) ->
        (generate_expression e1) ^ " " ^
↪  (generate_binary_operator o) ^ " " ^
↪  (generate_expression e2)
    | Unop(e,o) ->
        (match o with
        | Not | Negate  -> (generate_unary_operator o) ^
↪  (generate_expression e)
        | Plus_Plus | Minus_Minus -> (generate_expression
↪  e) ^ (generate_unary_operator o))
    | Array_Accessor(e,e_list) -> (generate_expression e) ^
↪  "[" ^ (generate_list generate_expression "][" e_list) ^
↪  "]"
    | Ternary(e1,e2,e3) -> "(" ^ (generate_expression e2) ^
↪  ") ? " ^ (generate_expression e1) ^ ":" ^
↪  (generate_expression e3)
  in sprintf "%s" expr
(* Generates CUDA statements that copy constants from host
↪  to gpu *)
and generate_constant_on_gpu const  =
  let mem_alloc_constant_string = match const.variable_type
↪  with
    | Primitive(vtype) ->
          generate_device_ptr
↪  (Utils.idtos(const.kernel_name)) ^
          generate_mem_alloc_statement_host_to_device const
↪  1 ^
          generate_mem_cpy_statement_host_to_device const 1
    | Array(vtype,length) ->
        "vlcarray fill"
(*     | _ -> raise Exceptions.Unknown_variable_type *)
  in
  sprintf "%s" mem_alloc_constant_string
```

```
and generate_kernel_function_call kfcall = sprintf "hi" (* Why do we need semicolon??????*)
  (* Fill in with VLC_Array *)
(* Generates statements for higher order map or reduce calls *)
and generate_higher_order_function_call fcall =
  let higher_order_function_call_string =
   match Utils.idtos(fcall.higher_order_function_type) with
   | "map" ->
  (* Fill in with VLC_Array *)
   "{0};\n" ^
  (* Initializes CUDA driver and loads needed function *)
   "checkCudaErrors(cuCtxCreate(&context, 0, device));\n" ^
   "std::ifstream t(\"" ^ Utils.idtos fcall.applied_kernel_function ^ ".ptx\");\n" ^
   "if (!t.is_open()) {\n" ^
     " std::cerr << \"" ^ Utils.idtos fcall.applied_kernel_function ^ ".ptx not found\n\";\n" ^
     "return 1;\n" ^
   "}\n" ^
   "std::string " ^ Utils.idtos fcall.applied_kernel_function ^ "_str" ^ "((std::istreambuf_iterator<char>(t)), std::istreambuf_iterator<char>());\n" ^
   "checkCudaErrors(cuModuleLoadDataEx(&cudaModule," ^ (Utils.idtos fcall.applied_kernel_function) ^ "_str" ^ ", 0, 0, 0));\n" ^
   "checkCudaErrors(cuModuleGetFunction(&function, cudaModule, \"" ^ (Utils.idtos fcall.applied_kernel_function) ^ "_str" ^ "\"));\n" ^
  (* Copies over constants *)
   generate_list generate_constant_on_gpu "\n" fcall.constants ^ "\n" ^
  (* Allocates GPU pointers for input and result array *)
  let rec get_kernel_names a_info_list name_list =
   match a_info_list with
    | [] -> name_list
    | hd::tl -> get_kernel_names tl (Utils.idtos(hd.kernel_name)::name_list)
  in
  let kernel_names = (get_kernel_names fcall.input_arrays_info []) in
   generate_list generate_device_ptr "\n" kernel_names ^ "\n" ^
   generate_device_ptr (Utils.idtos((fcall.return_array_info).kernel_name)) ^ "\n" ^
  (* Allocations memory and copies input arrays over to GPU memory *)
   generate_mem_alloc_host_to_device fcall ^ "\n" ^
   generate_mem_cpy_host_to_device fcall ^
```

```

    (* Sets Kernel params and other information needed to call cuLaunchKernel *)
      generate_kernel_params fcall.input_arrays_info ^ "\n" ^
      "unsigned int blockSizeX = 16;\n" ^
      "unsigned int blockSizeY = 1;\n" ^
      "unsigned int blockSizeZ = 1;\n" ^
      "unsigned int gridSizeX = 1;\n" ^
      "unsigned int gridSizeY = 1;\n" ^
      "unsigned int gridSizeZ = 1;\n" ^
    (* Launches kernel *)
      "checkCudaErrors(cuLaunchKernel(function, gridSizeX, gridSizeY, gridSizeZ, blockSizeX, blockSizeY, blockSizeZ,0, NULL, KernelParams, NULL));\n" ^
    (* Copies result array back to host *)
      "checkCudaErrors(cuMemcpyDtoH(c," ^ Utils.idtos((fcall.return_array_info).host_name) ^ ", sizeof(" ^ generate_variable_type ((fcall.return_array_info).variable_type) ^ ")*" ^ string_of_int fcall.array_length ^ "));\n" ^
    (* Cleanup *)
    generate_list generate_mem_cleanup "\n" fcall.input_arrays_info ^ "\n" ^
    generate_mem_cleanup fcall.return_array_info ^ "\n" ^
    generate_list generate_mem_cleanup "\n" fcall.constants ^ "\n" ^
    "checkCudaErrors(cuModuleUnload(cudaModule));\n" ^
    "checkCudaErrors(cuCtxDestroy(context));\n"
    | "reduce" ->
    (* Fill in with VLC_Array *)
      "{0};\n" ^
    (* Initializes CUDA driver and loads needed function *)
      "checkCudaErrors(cuCtxCreate(&context, 0, device));\n" ^
      "std::ifstream t(\"" ^ Utils.idtos fcall.applied_kernel_function ^ ".ptx\");\n" ^
      "if (!t.is_open()) {\n" ^
          " std::cerr << \"" ^ Utils.idtos fcall.applied_kernel_function ^ ".ptx not found\n\";\n" ^
          "return 1;\n" ^
      "}\n" ^
      "std::string " ^ Utils.idtos fcall.applied_kernel_function ^ "_str" ^ "((std::istreambuf_iterator<char>(t)), std::istreambuf_iterator<char>());\n" ^
      "checkCudaErrors(cuModuleLoadDataEx(&cudaModule," ^ (Utils.idtos fcall.applied_kernel_function) ^ "_str" ^ ", 0, 0, 0));\n" ^
      "checkCudaErrors(cuModuleGetFunction(&function, cudaModule, \"" ^ (Utils.idtos fcall.applied_kernel_function) ^ "_str" ^ "\"));\n" ^
    (* Copies over constants *)
```

```
   generate_list generate_constant_on_gpu "\n" fcall.constants ^ "\n" ^
 (* Allocates GPU pointers for input and result array *)
 let rec get_kernel_names a_info_list name_list =
   match a_info_list with
     | [] -> name_list
     | hd::tl -> get_kernel_names tl (Utils.idtos(hd.kernel_name)::name_list)
 in
 let kernel_names = (get_kernel_names fcall.input_arrays_info []) in
   generate_list generate_device_ptr "\n" kernel_names ^ "\n" ^
   generate_device_ptr (Utils.idtos((fcall.return_array_info).kernel_name))  ^ "\n" ^
 (* Allocations memory and copies input arrays over to GPU memory *)
   generate_mem_alloc_host_to_device fcall ^ "\n" ^
   generate_mem_cpy_host_to_device fcall ^

 (* Sets Kernel params and other information needed to call cuLaunchKernel *)
   generate_kernel_params fcall.input_arrays_info ^ "\n" ^
   "unsigned int blockSizeX = 16;\n" ^
   "unsigned int blockSizeY = 1;\n" ^
   "unsigned int blockSizeZ = 1;\n" ^
   "unsigned int gridSizeX = 1;\n" ^
   "unsigned int gridSizeY = 1;\n" ^
   "unsigned int gridSizeZ = 1;\n" ^
 (* Launches kernel *)
   "checkCudaErrors(cuLaunchKernel(function, gridSizeX, gridSizeY, gridSizeZ, blockSizeX, blockSizeY, blockSizeZ,0, NULL, KernelParams, NULL));\n" ^
 (* Copies result array back to host *)
   "checkCudaErrors(cuMemcpyDtoH(c," ^ Utils.idtos((fcall.return_array_info).host_name) ^ ", sizeof(" ^ generate_variable_type ((fcall.return_array_info).variable_type) ^ ")*" ^ string_of_int fcall.array_length ^ "));\n" ^
 (* Cleanup *)
 generate_list generate_mem_cleanup "\n" fcall.input_arrays_info ^ "\n" ^
 generate_mem_cleanup fcall.return_array_info ^ "\n" ^
 generate_list generate_mem_cleanup "\n" fcall.constants ^ "\n" ^
 "checkCudaErrors(cuModuleUnload(cudaModule));\n" ^
 "checkCudaErrors(cuCtxDestroy(context));\n"
| _ -> raise Exceptions.Unknown_higher_order_function_call
in sprintf "%s" higher_order_function_call_string


```

```

let generate_variable_statement vstatement =
  let vstatement_string = match vstatement with
    | Declaration(d)   ->
        (generate_vdecl d) ^ ";\n"
    | Assignment (id, e) ->
        (generate_id id) ^ "=" ^ (generate_expression e) ^ ";\n"
    | Initialization(d,e) ->
        (generate_vdecl d) ^ "=" ^ (generate_expression e) ^ ";\n"
(*      | _ -> raise Exceptions.Unknown_variable_statement *)
  in sprintf "%s" vstatement_string

(* Generates statements *)
let rec generate_statement statement  =
  let statement_string = match statement with
    | Variable_Statement(vsmtm) ->
        generate_variable_statement vsmtm
    | Expression(e) ->
        (generate_expression e) ^ ";\n"
    | Block(stmt_list) -> generate_list generate_statement "" stmt_list
    | If(e,stmt1,stmt2) ->
        (match stmt2 with
        | Block([]) -> "if(" ^ (generate_expression e) ^ "){\n" ^ (generate_statement stmt1) ^ "}\n"
        | _ -> "if(" ^ (generate_expression e) ^ "){\n" ^ (generate_statement stmt1) ^ "}\n" ^ "else{\n" ^ (generate_statement stmt2) ^ "}\n")
    | While(e,stmt) -> "while(" ^ (generate_expression e) ^ "){\n" ^ (generate_statement stmt) ^ "}\n"
    | For(stmt1,e,stmt2,stmt3) -> "for(" ^ (generate_statement stmt1) ^ (generate_expression e) ^ ";" ^ (generate_statement stmt2) ^ "){\n" ^ (generate_statement stmt3) ^ "}\n"
    | Return(e) ->
        "return" ^ (generate_expression e) ^ ";\n"
    | Return_Void ->
        "return;\n"
    | Continue ->
        "continue;\n"
    | Break ->
        "break;\n"
(*      | _ -> raise Exceptions.Unknown_type_of_statement *)
  in sprintf "%s" statement_string


(* Generates function declarations *)
let generate_fdecl f  =
  let fdecl_string =
    (generate_variable_type f.c_fdecl_return_type) ^ " " ^
    (generate_id f.c_fdecl_name) ^ "(" ^
    (generate_list generate_param "," f.c_fdecl_params) ^ "){\n" ^
    (generate_list generate_statement "\n" f.c_fdecl_body) ^ "}\n"
  in
  sprintf "%s" fdecl_string

(* Writing out to CUDA file *)
let write_cuda filename cuda_program_string =
  let file = open_out (filename ^ ".cu") in
  fprintf file "%s" cuda_program_string

(* Generates the full CUDA file *)
let generate_cuda_file filename program =
  let cuda_program_body =
    (generate_list generate_variable_statement "" (Utils.triple_fst(program))) ^
    (generate_list generate_fdecl "" (Utils.triple_trd(program)))
  in
  let cuda_program_string = sprintf "\n\
  #include <stdio.h>\n\
  #include <stdlib.h>\n\
  #include \"cuda.h\"\n\
  #include <iostream>\n\
  #include <vlc.hpp>\n\
  CUdevice    device;\n\
  CUmodule    cudaModule;\n\
  CUcontext   context;\n\
  CUfunction  function;\n\
  %s" cuda_program_body in
  write_cuda filename cuda_program_string

(* Generate program *)
let generate_program cuda_filename program =
  generate_cuda_file cuda_filename program;
  Codegen_ptx.generate_ptx_function_files program
```

codegen_ptx.ml

```
open Sast
(* open Exceptions *)
(* For sprintf *)
open Printf
(*------------------------------------------------------------ KERNEL CODE GENERATION ------------------------------------------------------------*)
(*
let generate_kernel_fdecl kernel_f  =
  Environment.combine  [
    Generator(generate_variable_type kernel_f.kernel_r_type);
    Verbatim(" ");
    Generator(generate_id kernel_f.kernel_name);
    Verbatim("(");
    Generator(generate_parameter_list kernel_f.kernel_params);
    Verbatim("){\n");
    Generator(generate_statement_list kernel_f.kernel_body);
    Verbatim("}\n");
  ]
let rec generate_nonempty_kernel_fdecl_list kernel_fdecl_list  =
  match kernel_fdecl_list with
    | kernel_fdecl :: [] -> Environment.combine [Generator(generate_kernel_fdecl kernel_fdecl)]
    | kernel_fdecl :: tail ->
      Environment.combine  [
        Generator(generate_kernel_fdecl kernel_fdecl);
        Verbatim("\n\n");
        Generator(generate_nonempty_kernel_fdecl_list tail)
      ]
    | [] -> raise (Empty_kernel_fdecl_list)
and generate_kernel_fdecl_list kernel_fdecl_list  =
  match kernel_fdecl_list with
    | [] -> Environment.combine  [Verbatim("")]
    | decl :: tail -> Environment.combine [Generator(generate_nonempty_kernel_fdecl_list kernel_fdecl_list)]
 *)



(*-------------------------------------Duplicated in codegen_-------------------------------------*)

(* Generate id *)
let generate_id id  =
  sprintf "%s" (Utils.idtos(id))
```

```
(* 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 *)
let generate_list generate_func concat mylist =
  let list_string = String.concat concat (List.map generate_func mylist) in
  sprintf "%s" list_string

(*-------------------------------------------------------------------------

let generate_ptx_binary_operator operator =
  let op = match operator with
    | Ptx_Add -> "add"
    | Ptx_Subtract -> "sub"
    | Ptx_Multiply -> "mul"
    | Ptx_Divide -> "div"
    | Ptx_Modulo -> "rem"
  in
  sprintf "%s" op

let generate_ptx_data_type data_type =
  let t = match data_type with
    | U16 -> ".u16"
    | U32 -> ".u32"
    | U64 -> ".u64"
    | S16 -> ".s16"
    | S32 -> ".s32"
    | S64 -> ".s64"
  in
  sprintf "%s" t

let generate_ptx_variable_type vtype =
  let v = ""
  (* TODO *)
  in
  sprintf "%s" v

let generate_ptx_vdecl dtype vtype id =
  let v =
    ".param " ^ generate_ptx_data_type dtype ^ " " ^ generate_ptx_variable_type vtype
    ^ " " ^ generate_id id
  in
  sprintf "%s" v

let generate_ptx_register_decl declaration =
  let r = match declaration with
  | Register_Declaration(dtype, name, size ) -> ".reg " ^ generate_ptx_data_type dtype
      ^ "   %" ^ name ^ "<" ^ string_of_int size ^ ">;\n"
  in
  sprintf "%s" r
```

```
let generate_ptx_register register =
  let r = match register with
    | Register(s, i) -> "%" ^ s ^ string_of_int i
  in
  sprintf "%s" r

let generate_ptx_parameter parameter =
  let p = match parameter with
    | Parameter_register(r) -> generate_ptx_register(r)
    | Parameter_constant(c) -> string_of_int c
    | Parameter_variable(v) -> "[" ^ generate_id v ^ "]"
  in
  sprintf "%s" p

let generate_ptx_expression expression =
  let e = match expression with
  | Ptx_reg_declaration(r) -> generate_ptx_register_decl(r)
  | Ptx_Binop(o, t, p1, p2, p3) ->
 ↪  generate_ptx_binary_operator(o) ^
 ↪  generate_ptx_data_type(t)
      ^ "     " ^ generate_ptx_parameter(p1) ^ ", " ^
 ↪  generate_ptx_parameter(p2) ^ ", "
      ^ generate_ptx_parameter(p3) ^ ";\n"
  | Ptx_Return -> "ret;\n"
  in
  sprintf "%s" e

let generate_ptx_subroutine subroutine =
  let s =
  generate_id subroutine.routine_name ^ ": \n" ^
  generate_list generate_ptx_expression ""
 ↪  subroutine.routine_expressions
  in
  sprintf "%s" s

let generate_ptx_statement statement =
  let s = match statement with
  | Ptx_expression(e) -> generate_ptx_expression(e)
  | Ptx_subroutine(s) -> generate_ptx_subroutine(s)
  in
  sprintf "%s" s


(* Generates the ptx function string *)
(* Fill in once you have the generation for other ptx types
 ↪  in the sast *)
(*
  should look like this
  .entry <function name>(
    <param list>
  ){
    <statement list>
  }
*)


(* Writing out to PTX file *)
let write_ptx filename ptx_string =
  let file = open_out (filename ^ ".ptx") in
  fprintf file "%s" ptx_string


(* Before each program include
// Generated by Vlc
.version 3.1
.target sm_20
.address_size 64
*)
(* Generates the ptx function string *)
let generate_ptx_function f =
  let ptx_function_body =
    ".visible .entry " ^ f.ptx_fdecl_name ^ "(" ^ (generate_list generate_ptx_vdecl "," f.ptx_fdecl_params) ^ ")\n" ^
    "{" ^
    (generate_list generate_ptx_register_decl "\n" f.register_declarations ) ^ "\n" ^
    (generate_list generate_ptx_statement "" f.ptx_fdecl_body) ^
    "}"
  in
  let ptx_function_string = sprintf "
  .version 3.1
  .target sm_20
  .address_size 64
  %s" ptx_function_body
  in
  sprintf "%s" ptx_function_body

(* Main function for generating all ptx files*)
let generate_ptx_function_files program =
  let ptx_function_list = Utils.triple_snd(program) in
  let rec generate_ptx_files ptx_func_list =
    match ptx_func_list with
      | [] -> ()
      | hd::tl ->
        write_ptx (Utils.idtos(hd.ptx_fdecl_name)) (generate_ptx_function hd);
        generate_ptx_files tl
  in generate_ptx_files ptx_function_list
```

vlc.ml

```
type action = Tokens | Ast | Compile | Sast | Run

let _ =
  if Array.length Sys.argv < 2 then
  print_string (
      "Usage: vlc [mode] <VLC program file>\n" ^
      "\t-t: prints tokens read in by scanner\n" ^
      "\t-a: prints ast as a program\n" ^
      "\t-s: prints sast as a program\n" ^
      "\t-c: compiles VLC program to CUDA C file and PTX files\n")
  else
    let action = List.assoc Sys.argv.(1) [ ("-t", Tokens);
                                           ("-a", Ast);
                                           ("-s", Sast);
                                           ("-c", Compile);
                                           ("-r", Run)] and
filename = Sys.argv.(2) in
print_endline filename;
(* let base_filename = List.hd (Str.split (Str.regexp ".vlc") (List.hd (List.rev (Str.split (Str.regexp "/") filename)))) in
*)let file_in = open_in filename in
      let lexbuf = Lexing.from_channel file_in in
      let token_list = Processor.get_token_list lexbuf in
      let program = Processor.parser token_list in
      let sast = Semant.analyze program in
      match action with
        | Tokens ->
            print_string (Utils.token_list_to_string token_list)
        | Ast ->
            print_string (Utils.program_to_string program)
        | Sast ->
            print_string (Utils.sast_to_string sast)
        | Compile ->
            Codegen_c.generate_program filename sast
        | Run ->
            Codegen_c.generate_program filename sast
(*             Sys.command ("nvcc -" ^ filename ^ " " ^ filename ^ ".cu");
            Sys.command ("./" ^ filename); *)
```

Makefile

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


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

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

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

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

depend:
  echo "Not needed."
```

Interfaces

ast.ml

```
type binary_operator =
    | Add | Subtract | Multiply | Divide | Modulo
(*      | Plus_Equal | Subtract_Equal | Multiply_Equal | Divide_Equal   *)
(*      | Exp | Dot | Matrix_Multiplication *)
    | And | Or | Xor
    | Equal | Not_Equal | Greater_Than | Less_Than | Greater_Than_Equal | Less_Than_Equal
    | Bitshift_Right | Bitshift_Left
type unary_operator =
    | Not   | Negate
    | Plus_Plus | Minus_Minus

type identifier =
    Identifier of string

type data_type =
    | String
    | Byte
    | Unsigned_Byte
    | Integer
    | Unsigned_Integer
    | Long
    | Unsigned_Long
    | Float
    | Double
    | Boolean
    | Void

type variable_type =
    | Primitive of data_type
    | Array of variable_type * int (* variable type, size *)
(*  | Struct of variable_type list * expression list * int *)

type vdecl =
    Variable_Declaration of variable_type * identifier

type expression =
    | Function_Call of identifier * expression list
    | Higher_Order_Function_Call of higher_order_function_call
    | String_Literal of string
    | Integer_Literal of int
    | Boolean_Literal of bool
    | Floating_Point_Literal of float
    | Array_Literal of expression list
    | Identifier_Literal of identifier
    | Cast of variable_type * expression
    | Binop of expression * binary_operator * expression
    | Unop of expression * unary_operator
    | Array_Accessor of expression * expression list (*
  ↪ Array, indexes *)
    | Ternary of expression * expression * expression
and constant =
    | Constant of identifier * expression
and higher_order_function_call = {
    higher_order_function_type                       :
  ↪ identifier; (* Map or reduce *)
    kernel_function_name                             :
  ↪ identifier;
    constants                                        :
  ↪ constant list;
    input_arrays                                     :
  ↪ expression list; (* Check in semantic analyzer that
  ↪ type is array*)
}

type variable_statement =
    | Declaration of vdecl
    | Initialization of vdecl * expression
    | Assignment of expression * expression

type statement =
    | Variable_Statement of variable_statement
    | Expression of expression
    | Block of statement list (* Used for if, else, for,
  ↪ while blocks *)
    | If of expression * statement * statement (*
  ↪ expression-condition, statement-if block,
  ↪ statement-optional else block *)
    | While of expression * statement
    | For of statement * expression * statement * statement
    | Return of expression
    | Return_Void
    | Continue
    | Break

type fdecl = {
    is_kernel_function                               : bool;
  ↪ (* Host or Kernel *)
    return_type                                      :
  ↪ variable_type;
    name                                             :
  ↪ identifier;
    params                                           : vdecl
  ↪ list;
```

```
    body                                        : statement list;
}

(* Program Definition *)
type program = variable_statement list * fdecl list
```

sast.ml

```ocaml
open Ast
(* Contains sast type definitions for conversions during semantic analysis *)
(* ----------------------------------------PTX types -----------------------------------------*)
type ptx_data_movement =
  | Ptx_Move | Ptx_Load | Ptx_Store

type ptx_binary_operator =
    | Ptx_Add | Ptx_Subtract | Ptx_Multiply | Ptx_Divide | Ptx_Modulo

type ptx_data_type =
  | U16 | U32 | U64 | S16 | S32 | S64

(* should use this as our information about global/param etc.*)
type ptx_variable_type =
  | Ptx_Primitive of ptx_data_type
  | Ptx_Array of ptx_variable_type * int           (* 'int' refers to the length of the array *)
  | Ptx_Pointer of ptx_variable_type * int         (* 'int' refers to size of memory pointed by the pointer *)

type ptx_register_decl =
  | Register_Declaration of ptx_data_type * string * int   (* type, name, number of registers *)

type ptx_register =
  | Register of string * int                 (* register name,  register number *)
(* Not sure what this is  | Typed_Register of ptx_data_type * string * int     (* type, register name, register number *)  *)
(* Implement later  | Special_Register of string              (* register name *)  *)

type ptx_parameter =
  | Parameter_register of ptx_register
  | Parameter_constant of int
  | Parameter_variable of Ast.identifier


type ptx_expression =
  | Ptx_reg_declaration of ptx_register_decl
  | Ptx_movement of ptx_data_movement * ptx_data_type * ptx_variable_type * ptx_parameter * ptx_parameter
  | Ptx_Binop of ptx_binary_operator * ptx_data_type * ptx_parameter * ptx_parameter * ptx_parameter
  | Ptx_Return
```

```
(*      | Ptx_Array_Literal of ptx_expression list
  | Ptx_Function_Call of Ast.identifier * ptx_expression list
  | Ptx_Identifier_Expression of Ast.identifier
 *)

type ptx_subroutine = {
  routine_name                : Ast.identifier;
  routine_expressions              : ptx_expression list;
}

type ptx_statement =
(*      | Ptx_Initialization of ptx_vdecl * ptx_expression *)
(*      | Ptx_Assignment of Ast.identifier * ptx_expression *)
    | Ptx_expression of ptx_expression
    | Ptx_subroutine of ptx_subroutine

type ptx_function_type =
  | Global
  | Device

type ptx_constant =
{
  ptx_constant_name                : Ast.identifier;
  ptx_constant_variable_type           : ptx_variable_type;
}

type ptx_variable_space =
  | Global
  | Local
  | Shared

type ptx_vdecl =
    | Ptx_Vdecl of ptx_data_type * ptx_variable_space(* need something about global/ptrs here*) ptx_variable_type * Ast.identifier


(* ptx fdecl is the entire file
  it seems it really only needs to be composed of a few parts - a name, a variable declaration list
  and a statement list
  register_decl list should go inside body generated from semantic analyzer
*)
type ptx_fdecl = {
  (* Global or Device *)
  ptx_fdecl_type               : ptx_function_type;  (* probably not needed *)

  (* Name of the function *)
  ptx_fdecl_name                   : Ast.identifier;

   (* Expected parameters of the function *)
  ptx_fdecl_params                 : ptx_vdecl list;

   (* List of constants that function needs to know - aka variables that aren't in scope of function when it goes through semantic analyzer
      If this constant list doesn't match the constant list of the higher order function, throw error in semant.ml *)
  ptx_consts                     : ptx_constant list;
   (* Declares the virtual registers that are needed for the function *)
  register_decls                   : ptx_register_decl list;
   (* Statements within the function *)
  ptx_fdecl_body                   : ptx_statement list;
}






(* -----------------------------------------C types -----------------------------------------*)

(*----------------------------------Unnecessary?????????----------------------------------*)
type c_binary_operator =
    | Add | Subtract | Multiply | Divide | Modulo
(*      | Plus_Equal | Subtract_Equal | Multiply_Equal | Divide_Equal   *)
(*      | Exp | Dot | Matrix_Multiplication *)
    | And | Or | Xor
    | Equal | Not_Equal | Greater_Than | Less_Than | Greater_Than_Equal | Less_Than_Equal
    | Bitshift_Right | Bitshift_Left
type c_unary_operator =
    | Not | Negate
    | Plus_Plus | Minus_Minus

type c_data_type =
  | String
    | Byte
    | Unsigned_Byte
    | Integer
    | Unsigned_Integer
    | Long
    | Unsigned_Long
    | Float
    | Double
    | Boolean
    | Void

type c_variable_type =
  | Primitive of c_data_type
  | Array of c_variable_type * int
(*   | Struct of variable_type list * expression list * int *)

type c_vdecl =
    Variable_Declaration of c_variable_type * Ast.identifier

(* ----------------------------------Necessary----------------------------- *)

type c_kernel_variable_info = {
  variable_type       : c_variable_type;
  host_name         : Ast.identifier;
  kernel_name       : Ast.identifier;
}

type c_higher_order_function_call = {
  (* Map or reduce *)
  higher_order_function_type         : Ast.identifier;
  (* Name of kernel function that is called from host (would be kernel function corresponding to map/reduce) *)
    applied_kernel_function            : Ast.identifier;
  (* List of constants passed into map and reduce *)
  constants                  : c_kernel_variable_info list;
  (* Size of input and return arrays *)
  array_length                : int;
  (* Input array information
    --If an array has no name (just simply passed in as something like {1,2,3}) then it is given a temporary generated name *)
  input_arrays_info             : c_kernel_variable_info list; (* type, host name, kernel name *)
    (* Return array information *)
    return_array_info                    : c_kernel_variable_info; (* type, host name, kernel name*)
}

(* Type for calling defg functions directly from host *)
type c_kernel_function_call = {
  (* Name of the function that is called from the host *)
  kernel_function             : Ast.identifier;
  (* Input array information
    --If an array has no name (just simply passed in as something like {1,2,3}) then it is given a temporary generated name *)
  input_args_info              : c_kernel_variable_info list; (* type, host name, kernel name *)
    (* Return array information *)
    return_arg_info                    : c_kernel_variable_info; (* type, host name, kernel name*)
}

type c_expression =
    | Function_Call of Ast.identifier * c_expression list
    | Higher_Order_Function_Call of c_higher_order_function_call
    | Kernel_Function_Call of c_kernel_function_call
    | String_Literal of string
    | Integer_Literal of int
    | Boolean_Literal of bool
    | Floating_Point_Literal of float
    | Array_Literal of c_expression list
    | Identifier_Literal of Ast.identifier
    | Cast of c_variable_type * c_expression
    | Binop of c_expression * c_binary_operator * c_expression
    | Unop of c_expression * c_unary_operator
    | Array_Accessor of c_expression * c_expression list (* Array, indexes *)
    | Ternary of c_expression * c_expression * c_expression (* expression if true, condition, expression if false *)

type c_variable_statement =
    | Declaration of c_vdecl
    | Initialization of c_vdecl * c_expression
    | Assignment of Ast.identifier * c_expression

type c_statement =
    | Variable_Statement of c_variable_statement
    | Expression of c_expression
    | Block of c_statement list (* Used for if, else, for, while blocks *)
    | If of c_expression * c_statement * c_statement (* expression-condition, statement-if block, statement-optional else block *)
    | While of c_expression * c_statement
    | For of c_statement * c_expression * c_statement * c_statement
    | Return of c_expression
    | Return_Void
    | Continue
    | Break

type c_fdecl = {
    c_fdecl_return_type     : c_variable_type;
    c_fdecl_name          : Ast.identifier;
    c_fdecl_params        : c_vdecl list;
    c_fdecl_body          : c_statement list;
}

(* Overall Program *)
type program = c_variable_statement list * ptx_fdecl list *
 ↪  c_fdecl list
```

exceptions.ml

```ocaml
(* Collection of exceptions for different parts of the compiler *)

(*-------------------------------------Scanner-----------------------------
exception Bad_dedent
(*-------------------------------------Parser------------------------------
exception Array_parsing_error
exception Invalid_data_type of string

exception Lexing_error of string  (* Unused atm *)
exception Parsing_error of string (* Unused atm *)
(*-------------------------------------Processor---------------------------
exception Missing_eof
(*-------------------------------------Utils-------------------------------
(*-------------------------------------Semantic Analyzer-------------------------------------*)
exception Cannot_infer_expression_type
exception Exception of string
exception Already_declared
exception Name_not_found of string
exception Invalid_environment
exception Variable_not_found_in_scope
exception Function_not_defined
exception Cannot_pop_empty_variable_scope_stack
exception Variable_already_declared
exception Not_an_array_expression
exception Type_mismatch of string
exception Empty_array_expression_list
exception Variable_not_declared
(*-------------------------------------Codegen C-------------------------------------*)
exception Unknown_variable_type
exception Unknown_operator
exception Unknown_data_type
exception Unknown_type_of_param
exception Unknown_higher_order_function_call
exception Unknown_type_of_vdecl
exception Unknown_type_of_expression
exception Unknown_variable_statement
exception Unknown_type_of_statement
(*-------------------------------------Codegen PTX-------------------------------------*)
```

utils.ml

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

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

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

(* Print data types *)

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

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

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

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

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

(* Print expressions *)

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

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

```
;;

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

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

(* Print statements *)

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

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(* Print Function *)

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

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

let string_of_formal_name = function
Formal(_, s) -> s
|   _ -> ""

let string_of_func_decl fdecl =
"" ^ (string_of_scope fdecl.scope) ^ " " ^
 ↪  (string_of_datatype fdecl.returnType) ^ " " ^
 ↪  (string_of_fname fdecl.fname) ^ " " ^
(* Formals *)
```

```
"(" ^ String.concat "," (List.map string_of_formal fdecl.formals) ^ ") {\n" ^
 (* body *)
 String.concat "" (List.map (string_of_stmt 2) fdecl.body) ^
 "\t}\n\n"

(* Class Printing *)

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

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

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

(* Include Printing *)

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

(* Print whole program *)

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

(* Print AST tree representation *)

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

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

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

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

let rec map_stmt_to_json = function
```

```
Block(stmts)       -> `Assoc [("block", `List (List.map (map_stmt_to_json) stmts))]
|   Expr(expr)         -> `Assoc [("expr", map_expr_to_json expr)]
|   Return(expr)       -> `Assoc [("return", map_expr_to_json expr)]
|   If(e, s1, s2)       -> `Assoc [("if", `Assoc [("cond", map_expr_to_json e); ("ifbody", map_stmt_to_json s1)]); ("else", map_stmt_to_json s2)]
|   For(e1, e2, e3, s)     -> `Assoc [("for", `Assoc [("init", map_expr_to_json e1); ("cond", map_expr_to_json e2); ("inc", map_expr_to_json e3); ("body", map_stmt_to_json s)])]
|   While(e, s)       -> `Assoc [("while", `Assoc [("cond", map_expr_to_json e); ("body", map_stmt_to_json s)])]
|    Break          -> `String "break"
|    Continue        -> `String "continue"
|   Local(d, s, e)       -> `Assoc [("local", `Assoc [("datatype", `String (string_of_datatype d)); ("name", `String s); ("val", map_expr_to_json e)])]

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


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

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

(* Print SAST tree representation *)

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

let rec map_sstmt_to_json =
let datatype d = [("datatype", `String (string_of_datatype d))] in
function
SBlock sl                 -> `Assoc [("sblock", `List (List.map (map_sstmt_to_json) sl))]
|   SExpr(e, d)               -> `Assoc [("sexpr", `Assoc ([("expr", map_sexpr_to_json e)] @ (datatype d)))]
|   SReturn(e, d)           -> `Assoc [("sreturn", `Assoc ([("return", map_sexpr_to_json e)] @ (datatype d)))]
|   SIf (e, s1, s2)             -> `Assoc [("sif", `Assoc [("cond", map_sexpr_to_json e); ("ifbody", map_sstmt_to_json s1)]); ("selse", map_sstmt_to_json s2)]
|   SFor (e1, e2, e3, s)        -> `Assoc [("sfor", `Assoc [("init", map_sexpr_to_json e1); ("cond", map_sexpr_to_json e2); ("inc", map_sexpr_to_json e3); ("body", map_sstmt_to_json s)])]
|   SWhile (e, s)           -> `Assoc [("swhile", `Assoc [("cond", map_sexpr_to_json e); ("body", map_sstmt_to_json s)])]
|   SBreak                  -> `String "sbreak"
|   SContinue               -> `String "scontinue"
|   SLocal(d, s, e)         -> `Assoc [("slocal", `Assoc [("datatype", `String (string_of_datatype d)); ("name", `String s); ("val", map_sexpr_to_json e)])]

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

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

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

```

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

let map_sprogram_to_json sprogram =
`Assoc [("sprogram", `Assoc [
("classes", map_scdecls_to_json sprogram.classes);
("functions", map_sfdecls_to_json sprogram.functions);
("main", map_sfdecl_to_json sprogram.main);
("reserved", map_sfdecls_to_json sprogram.reserved);
])]

(* Print tokens *)

let string_of_token = function
LPAREN        -> "LPAREN"
|   RPAREN        -> "RPAREN"
|   LBRACE        -> "LBRACE"
|   RBRACE        -> "RBRACE"
|   SEMI        -> "SEMI"
|   COMMA        -> "COMMA"
|   PLUS        -> "PLUS"
|   MINUS        -> "MINUS"
|   TIMES        -> "TIMES"
|   DIVIDE        -> "DIVIDE"
|   ASSIGN        -> "ASSIGN"
|   EQ          -> "EQ"
|   NEQ          -> "NEQ"
|   LT          -> "LT"
|   LEQ          -> "LEQ"
|   GT          -> "GT"
|   GEQ          -> "GEQ"
|   AND          -> "AND"
|   OR          -> "OR"
|   NOT          -> "NOT"
|   DOT          -> "DOT"
|   LBRACKET      -> "LBRACKET"
|   RBRACKET      -> "RBRACKET"
|   BAR          -> "BAR"
|   IF          -> "IF"
|   ELSE        -> "ELSE"
|   FOR          -> "FOR"
|   WHILE        -> "WHILE"
|   RETURN        -> "RETURN"
|   INT          -> "INT"
|   FLOAT        -> "FLOAT"
|   BOOL        -> "BOOL"
|   CHAR        -> "CHAR"
|   VOID        -> "VOID"
|   NULL        -> "NULL"
|   TRUE        -> "TRUE"
|   FALSE        -> "FALSE"
|   CLASS        -> "CLASS"
|   CONSTRUCTOR      -> "CONSTRUCTOR"
|   PUBLIC        -> "PUBLIC"
|   PRIVATE        -> "PRIVATE"
|   EXTENDS        -> "EXTENDS"
|   INCLUDE        -> "INCLUDE"
|   THIS        -> "THIS"
|   BREAK        -> "BREAK"
|   CONTINUE      -> "CONTINUE"
|   NEW         -> "NEW"
|   INT_LITERAL(i)    -> "INT_LITERAL(" ^ string_of_int i ^ ")"
|   FLOAT_LITERAL(f)  -> "FLOAT_LITERAL(" ^ string_of_float f ^ ")"
|   CHAR_LITERAL(c)    -> "CHAR_LITERAL(" ^ Char.escaped c ^ ")"
|   STRING_LITERAL(s)  -> "STRING_LITERAL(" ^ s ^ ")"
|   ID(s)        -> "ID(" ^ s ^ ")"
|   DELETE         -> "DELETE"
|   MODULO         -> "MODULO"
|    EOF          -> "EOF"

let string_of_token_no_id = function
LPAREN        -> "LPAREN"
|   RPAREN        -> "RPAREN"
|   LBRACE        -> "LBRACE"
|   RBRACE        -> "RBRACE"
|   SEMI        -> "SEMI"
|   COMMA        -> "COMMA"
|   PLUS        -> "PLUS"
|   MINUS        -> "MINUS"
|   TIMES        -> "TIMES"
|   DIVIDE        -> "DIVIDE"
|   ASSIGN        -> "ASSIGN"
|   EQ          -> "EQ"
|   NEQ          -> "NEQ"
|   LT          -> "LT"
|   LEQ          -> "LEQ"
|   GT          -> "GT"
|   GEQ          -> "GEQ"
|   AND          -> "AND"
|   OR          -> "OR"
|   NOT          -> "NOT"
|   DOT          -> "DOT"
|   LBRACKET      -> "LBRACKET"
|   RBRACKET      -> "RBRACKET"
|   BAR          -> "BAR"
|   IF          -> "IF"
|   ELSE        -> "ELSE"
|   FOR          -> "FOR"
|   WHILE        -> "WHILE"
|   RETURN        -> "RETURN"
|   INT          -> "INT"
|   FLOAT        -> "FLOAT"
|   BOOL        -> "BOOL"
|   CHAR        -> "CHAR"
|   VOID        -> "VOID"
|   NULL        -> "NULL"
|   TRUE        -> "TRUE"
|   FALSE        -> "FALSE"
|   CLASS        -> "CLASS"
|   CONSTRUCTOR      -> "CONSTRUCTOR"
|   PUBLIC        -> "PUBLIC"
|   PRIVATE        -> "PRIVATE"
|   EXTENDS        -> "EXTENDS"
|   INCLUDE        -> "INCLUDE"
|   THIS        -> "THIS"
|   BREAK        -> "BREAK"
|   CONTINUE      -> "CONTINUE"
|   NEW         -> "NEW"
|   INT_LITERAL(i)    -> "INT_LITERAL"
|   FLOAT_LITERAL(f)  -> "FLOAT_LITERAL"
|   CHAR_LITERAL(c)    -> "CHAR_LITERAL"
|   STRING_LITERAL(s)  -> "STRING_LITERAL"
|   ID(s)        -> "ID"
|   DELETE         -> "DELETE"
|   MODULO         -> "MODULO"
|    EOF          -> "EOF"

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

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

processor.ml

```
open Parser
(* open Exceptions *)

let last_token = ref EOF

(* Gets the original raw tokens from the scanner *)
let get_tokens lexbuf =
  let rec next lexbuf token_list =
  match Scanner.token lexbuf with
    | DEDENT_EOF(c) as eof-> eof :: token_list
    | _ as token -> token :: (next lexbuf token_list)
  in next lexbuf []

(* Replaces DEDENT_COUNT with DEDENTS *)
let rec get_tokens_with_dedents original_token_list
 ↪  new_token_list=
  let rec fill_dedent count mylist =
    if count <= 0 then mylist
    else
      fill_dedent (count-1)
 ↪  (List.rev(DEDENT::List.rev(mylist)))
  in
  if (List.length(original_token_list)) != 0 then
    match (List.hd original_token_list) with
      | DEDENT_COUNT(c) ->
        let temp1 = (List.rev (TERMINATOR::(List.rev
 ↪  new_token_list))) in
        let temp = fill_dedent c temp1 in
        get_tokens_with_dedents (List.tl
 ↪  original_token_list) temp
      | DEDENT_EOF(c) ->
        let temp1 = (List.rev (TERMINATOR::(List.rev
 ↪  new_token_list))) in
        let temp = fill_dedent c temp1 in
        List.rev(EOF::(List.rev temp));
      | _ as token -> get_tokens_with_dedents (List.tl
 ↪  original_token_list) (List.rev (token :: (List.rev
 ↪  new_token_list)))
  else
    new_token_list

(* Removes opening TERMINATOR if it is there *)
let filter_opening_whitespace token_list =
    match token_list with
    | [] -> []
    | hd::tail -> if (hd = TERMINATOR) then tail else
 ↪  token_list

(* Function that uses above three functions to get the
 ↪  final list of tokens *)
```

```
let get_token_list lexbuf =
  let original_token_list = get_tokens lexbuf in
  let new_token_list = get_tokens_with_dedents
 ↪  original_token_list [] in
  let filtered_token_list = filter_opening_whitespace
 ↪  new_token_list
in filtered_token_list

(* Parse function *)
let parser token_list =
  let token_list = ref(token_list) in
  let tokenizer _ =
    match !token_list with
        | head :: tail ->
            last_token := head;
            token_list := tail;
            head
        | [] -> raise (Exceptions.Missing_eof) in
  let program = Parser.program tokenizer
 ↪  (Lexing.from_string "") in
  program
```

Library files

vlc.hpp

```
#ifndef VLC_H
#define VLC_H


// Defines the default block and grid size
#ifndef BLOCK_SIZE
#define BLOCK_SIZE 1024
#endif

#ifndef GRID_SIZE
#define GRID_SIZE 32
#endif
// Include statements
#include <stdlib.h>
#include <iostream>
#include <stdarg.h>

// Useful Macros for CUDA
// #define min(a, b)  (((a) < (b)) ? (a) : (b))
// #define max(a, b)  (((a) > (b)) ? (a) : (b))

// CUDA Error checking function
// void checkCudaErrors(CUresult err) {
//   assert(err == CUDA_SUCCESS);
// }

/* Why this class exists:
  - For ensuring that we don't have any arrays allocated on the stack and all are allocated on the heap
  ( can get messy with memory otherwise )
  - To bypass C/C++ not being able to do things like the following assignment
      size_t a[5];
      size_t b[5] = {1,2,3,4,5};
      a=b;

      !!size_t[5] not assignable error!!
*/

// VLC Array class
template <class T>
class VLC_Array {
  private:
    size_t num_values; //Tells us how many values are in the array in total. Ex. would be 4 if [2][2] array
    T*  values; // Posize_ter to values in array

    size_t num_dimensions; // Integer that tells us how many dimensions the array contains
    size_t *dimensions; // Integer array of the dimensions of the VLC_Array
  public:
    // Constructors and Destructors
    VLC_Array();
    VLC_Array(size_t num_values, size_t num_dimensions,...);
    VLC_Array(size_t num_values, size_t num_dimensions,size_t total_args...);
    VLC_Array(size_t num_values, T*values, size_t num_dimensions, size_t *dimensions);                                    // For declarations
      // For declarations and initializations like size_t a[5] = {1,2,3,4,5}
    VLC_Array(const VLC_Array<T> &vlcarray);   // For assignments like size_t a[1] = {5}, size_t b[1]={7}, a=b
    ~VLC_Array();

    /* Class Accessors and Getters */
    T*      get_values() const ; // Returns the posize_ter to VLC's size_ternal array
    size_t* get_dimensions() const; // Returns the posize_ter to VLC's dimensions
    size_t  get_num_dimensions () const; // Returns number of dimensions
    size_t  size()const; // Returns length of first dimension
    size_t  total_elements() const; // Returns total elements in the array

    /* Element Accessors and Getters */
    T get_element_value(size_t number_accessing_dims,...) const;
    VLC_Array<T> get_array_value_host(size_t number_accessing_dims,...) const;
    T* get_array_value_kernel(size_t number_accessing_dims,...) const;
    void set_element_value(T new_value,size_t number_accessing_dims, ...);
    void set_array_value(const VLC_Array<T> &array,size_t number_accessing_dims, ...);
    VLC_Array<T> operator=(const VLC_Array<T> &vlcarray);

};

/*---------------------------------- Regular constructors ----------------------------------*/
template <class T>
VLC_Array<T>::VLC_Array(){
  this->num_values = 0;
  this->values = NULL;
  this->num_dimensions = 0;
  this->dimensions = NULL;
}


template <class T>
VLC_Array<T>::VLC_Array(size_t num_values, T*values, size_t num_dimensions, size_t *dimensions){
  this->num_values = num_values;
  this->num_dimensions = num_dimensions;

  T *values_copy = (T*)calloc(num_values,sizeof(T));
  for(size_t i = 0; i < num_values; i++){
    values_copy[i] = values[i];
  }

  size_t *dims_copy = (size_t*)calloc(num_dimensions,sizeof(size_t));
  for(size_t j = 0; j < num_dimensions; j++){
    dims_copy[j] = dimensions[j];
  }

  this->values = values_copy;
  this->dimensions = dims_copy;
}

//Declarations
template <class T>
VLC_Array<T>::VLC_Array(size_t num_values, size_t num_dimensions,...){
  /* Assign the dimensions and values */
  this->num_dimensions = num_dimensions;
  this->num_values = num_values;

  this->dimensions = (size_t *)calloc( num_dimensions,sizeof(size_t));
  this->values = (T*)calloc( num_values,sizeof(T));

  /* Now access the values that are passed in */
  std::cout<<num_dimensions<<std::endl;
  va_list args;
  va_start(args,num_dimensions);
  for(size_t i = 0; i < num_dimensions;   i++)  {    this->dimensions[i] = va_arg(args,size_t);    }
  va_end(args);
}

// Declarations, Assignments by value
template <class T>
VLC_Array<T>::VLC_Array(size_t num_values, size_t num_dimensions,size_t total_args...){
  /* Assign the dimensions and values */
  this->num_dimensions = num_dimensions;
  this->num_values = num_values;

  this->dimensions = (size_t *)calloc(num_dimensions,sizeof(size_t));
  this->values = (T*)calloc( num_values,sizeof(T));

  /* Now access the values that are passed in */
  va_list args;
  va_start(args,total_args);
  for(size_t i = 0; i < num_dimensions;   i++)  {    this->dimensions[i] = va_arg(args,size_t);    }
  for(size_t j = 0; j < num_values;     j++)  {  this->values[j] = va_arg(args,T);      }
  va_end(args);
}

// Assignments to other arrays
template <class T>
VLC_Array<T>::VLC_Array(const VLC_Array<T> &vlcarray){
  /* For now, make a deep copy every time. Can optimize later */
  this->num_values = vlcarray.total_elements();
  this->num_dimensions = vlcarray.get_num_dimensions();

  this->values = (T *)calloc(this->num_values,sizeof(T));
  this->dimensions = (size_t *)calloc( this->num_dimensions,sizeof(size_t));

  /* Now access the values that are passed in */
  for(size_t j = 0; j < this->num_values;    j++)   {
 ↪      this->values[j]     = vlcarray.get_values()[j];
 ↪        }
  for(size_t i = 0; i < this->num_dimensions;i++)   {
 ↪      this->dimensions[i]    =
 ↪  vlcarray.get_dimensions()[i];      }
}

// Destructor
template <class T>
VLC_Array<T>::~VLC_Array(){
  free(this->values);
  free(this->dimensions);
}

/*---------------------------------- Accessing Functions
 ↪  ----------------------------------*/
// Get Element Value
// Accesses element of the array - must check num_accessing
 ↪  = num_dims in semant
template <class T>
T VLC_Array<T>::get_element_value(size_t
 ↪  number_accessing_dims,...) const{
  size_t index = 1;
  size_t corr_dim;
  printf("num_access_dim%zu\n",number_accessing_dims );
  va_list dims;
  va_start(dims,number_accessing_dims);
  for(size_t i=0; i < number_accessing_dims;i ++){
    index = va_arg(dims,size_t) * index;
    printf("index right now is %zu\n",index);
    corr_dim = this-> dimensions[i];
    printf("dim right now is%zu\n",corr_dim);
    index = i * corr_dim + index;
  }
  printf("%zu\n",index);
  va_end(dims);
  return this->values[index];
}

// Get Array Value In Host
// Accesses an array of the array - must check
 ↪  num_accessing < num_dims in semant
template <class T>
VLC_Array<T> VLC_Array<T>::get_array_value_host(size_t
 ↪  number_accessing_dims,...) const{
  // Get where new array starts
  size_t index = 1;
  size_t corr_dim;
  va_list dims;
  va_start(dims,number_accessing_dims);
  for(size_t i=0; i < number_accessing_dims; i++){
    index = va_arg(dims,size_t) * index;
    corr_dim = this-> dimensions[i];
    index = i * corr_dim + index;
  }
  va_end(dims);

  // Get all the elements in this new array
  size_t num_elements = 1;
  for(size_t i = this->num_dimensions -
 ↪  number_accessing_dims; i < this->num_dimensions;i++){
    num_elements = num_elements * this->dimensions[i];
  }

  // Set values
  size_t num_dimensions = this->num_dimensions -
 ↪  number_accessing_dims;
  size_t *new_dimensions =
 ↪  &(this->dimensions[this->num_dimensions -
 ↪  number_accessing_dims]);
  size_t num_values = num_elements;
  size_t *new_values = this->values[index];

  // Return a VLC_Array
  return
 ↪  VLC_Array(num_values,new_values,num_dimensions,new_dimensions);

}

// Get Array Value In Kernel
// Accesses an array of the array - must check
 ↪  num_accessing < num_dims in semant
template <class T>
T* VLC_Array<T>::get_array_value_kernel(size_t
 ↪  number_accessing_dims,...) const{
  // Get where new array starts
  size_t index = 1;
  size_t corr_dim;
  va_list dims;
  va_start(dims,number_accessing_dims);
  for(size_t i=0; i < number_accessing_dims; i++){
    index = va_arg(dims,size_t) * index;
    corr_dim = this-> dimensions[i];
    index = i * corr_dim + index;
  }
  va_end(dims);

  // Get all the elements in this new array
  size_t num_elements = 1;
  for(size_t i = this->num_dimensions -
 ↪  number_accessing_dims; i < this->num_dimensions;i++){
    num_elements = num_elements * this->dimensions[i];
  }

  // Set values
  size_t num_dimensions = this->num_dimensions -
 ↪  number_accessing_dims;
```

```
  size_t *new_dimensions = &(this->dimensions[this->num_dimensions - number_accessing_dims]);
  size_t num_values = num_elements;
  size_t *new_values = this->values[index];

  // Return a VLC_Array
  return VLC_Array(num_values,new_values,num_dimensions,new_dimensions);

}

// Set Element Value
// Sets value for element of an array
template <class T>
void VLC_Array<T>::set_element_value(T new_value,size_t number_accessing_dims,...){
  // Get where new array starts
  size_t index = 1;
  size_t corr_dim;
  va_list dims;
  va_start(dims,number_accessing_dims);
  for(size_t i=0; i < number_accessing_dims; i++){
    index = va_arg(dims,size_t) * index;
    corr_dim = this-> dimensions[i];
    index = i * corr_dim + index;
  }
  va_end(dims);
  printf("new value is %d\n", new_value);
  this->get_values()[index] = new_value;
}

// Set Array Value
// Sets value for an array of an array
template <class T>
void VLC_Array<T>::set_array_value(const VLC_Array<T> &array,size_t number_accessing_dims,...){
  // Get where new array starts
  size_t index = 1;
  size_t corr_dim;
  va_list dims;
  va_start(dims,number_accessing_dims);
  for(size_t i=0; i < number_accessing_dims; i++){
    index = va_arg(dims,size_t) * index;
    corr_dim = this-> dimensions[i];
    index = i * corr_dim + index;
  }
  va_end(dims);

  //Get number of elements to replace
  size_t num_elements = 1;
  for(size_t i = this->num_dimensions - number_accessing_dims; i < this->num_dimensions;i++){
    num_elements = num_elements * this->dimensions[i];
  }
  // Copy values
  for(size_t i =0; i < num_elements; i++){
    this->values[int(index + i)] =
 ↪  array.get_element_value(1,i);
  }
}
//Operator =
template <class T>
VLC_Array<T> VLC_Array<T>::operator=(const VLC_Array<T>
 ↪  &vlcarray){
  if(this == &vlcarray){
    return *this;
  }
  /* For now, make a deep copy every time. Can optimize
 ↪  later */
  num_values = vlcarray.total_elements();
  num_dimensions = vlcarray.get_num_dimensions();

  values = (T*)calloc(sizeof(T) *
 ↪  vlcarray.total_elements());
  dimensions = (size_t *)calloc(sizeof(size_t) *
 ↪  vlcarray.get_num_dimensions());

  /* Now access the values that are passed in */
  for(size_t j = 0; j < this->num_values;    j++)   {
 ↪      this->values[int(j)]        =
 ↪  vlcarray.get_values()[int(j)];        }
  for(size_t i = 0; i < this->num_dimensions; i++)   {
 ↪        this->dimensions[int(i)]       =
 ↪  vlcarray.get_dimensions()[int(i)];      }
  return *this;
}

template <class T>
T* VLC_Array<T>::get_values() const{ return this->values;}

template <class T>
size_t* VLC_Array<T>::get_dimensions() const{ return
 ↪  this->dimensions;}

template <class T>
size_t VLC_Array<T>::get_num_dimensions() const{ return
 ↪  this->num_dimensions; }

template <class T>
size_t VLC_Array<T>::size()const{ return
 ↪  this->dimensions[0]; }

template <class T>
size_t VLC_Array<T>::total_elements() const{ return
 ↪  this->num_values; }

#endif
```

Tests

test.sh

```
#!/bin/bash
(set -o igncr) 2>/dev/null && set -o igncr; # this comment is required

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

NVCC="nvcc"

VLC="sudo vlc -c"

globallog=./tests/test.log
rm -f $globallog
error=0
globalerror=0
NC='\033[0m'
GREEN='\033[0;32m'
CYAN='\033[0;36m'
keep=0
pass=0
fail=0
Usage() {
    echo "Usage: test.sh [options] [.mc files]"
    echo "-k    Keep intermediate files"
    echo "-h    Print this help"
    exit 1
}

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

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

```
  SignalError "$1 differs"
  echo "FAILED $1 differs from $2" 1>&2
    }
}

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

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

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

    echo -n "$basename..."

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

    generatedfiles=""

    generatedfiles="$generatedfiles ./tests/${basename}.cu ./tests/${basename}.out ./tests/${basename}" &&
    Run "$VLC" $1 "> null" &&
    Run "$NVCC" "./tests/${basename}.cu -o ./tests/${basename} && ./tests/${basename}" ">" "./tests/${basename}.out" &&
    Compare ./tests/${basename}.out ./${reffile}.out ./tests/${basename}.diff

    if [ $error -eq 0 ] ; then
  if [ $keep -eq 0 ] ; then
      rm -f $generatedfiles
  fi
  echo "OK"
  echo "###### SUCCESS" 1>&2
  ((pass++))
    else
  echo "###### FAILED" 1>&2
  globalerror=$error
  ((fail++))
    fi
    echo -n "$basename 2"

}

CheckFail() {
     error=0
     basename=`echo $1 | sed 's/.*\\///
                              s/.vlc//'`

     echo -n "$basename..."

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

     generatedfiles=""

     generatedfiles="$generatedfiles ./${basename}.out ./${basename}.diff" &&
     RunFail "$VLC" $1 "2>" "${basename}.out" ">>" $globallog &&
     Compare "tests/$basename.vlc.err" "./${basename}.out" "./${basename}.diff"

     # Report the status and clean up the generated files

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

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

shift `expr $OPTIND - 1`

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

for file in $files
do
    case $file in
  *test-*)
      Check $file 2>> $globallog
      ;;
  *fail-*)
            CheckFail $file 2>> $globallog
      ;;
  *)
      echo "unknown file type $file"
      globalerror=1
      ;;
    esac
done
echo ""
echo -e "Tests Passed: $pass"
echo -e "Tests Failed: $fail"
exit $globalerror
```

test-arithmetic_ops.vlc

```
int def vlc():
  int a = 2
  int b = 4

  int c = a + b
  int d = b - a
  int e = a * b
  int f = b / a
  int g = b % a

  print("success")
  return 0
```

test-arithmetic_ops.cu

```
#include <stdio.h>
#include <stdlib.h>
#include "cuda.h"
#include <iostream>
#include "vlc.hpp"
#include <stdarg.h>
CUdevice    device;
CUmodule    cudaModule;
CUcontext   context;
CUfunction  function;
int vlc(){
int a=2;
int b=4;
int c=a + b;
int d=b - a;
int e=a * b;
int f=b / a;
int g=b % a;
printf("success");
return 0;
}


int main(void) { return vlc(); }
```

test-print_hello_world.vlc

```
string helloworld

int def vlc():
  helloworld = "Hello world!"
  print(helloworld)
  return 0
```

test-print_hello_world.cu

```
#include <stdio.h>
#include <stdlib.h>
#include "cuda.h"
#include <iostream>
#include "vlc.hpp"
#include <stdarg.h>
CUdevice    device;
CUmodule    cudaModule;
CUcontext   context;
CUfunction  function;
char * helloworld;
int vlc(){
helloworld="Hello world!";
printf(helloworld);
return 0;
}


int main(void) { return vlc(); }
```

test-statements.vlc

```
int defg add(int x, int y):
    return scale * (x + y)

int defg vector_add(int a, int b):
    int index = a
    if(index == 1):
        index = 5

    for (int i = 0, i < 2, i = i + 1):
        print(i)

    while(i < 3):
        print(i)

    index = 5 if (i == 2) else 2
    a[4]



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

    int[5] c = ~map(add, consts(scale = 4),a,b)
    int[5] d = vector_add(a,b)

    print(d)

    return 0
```

test-statements.cu

```

```

fail-Already_declared.vlc

```
int a = 5
int a = 6

int def main():
  return 1
```

fail-Already_declared.vlc.err

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

fail-Array_elements_not_all_same_type.vlc

```
int[5] a = [1.0, 1]

int def main():
  return 1
```

fail-Array_elements_not_all_same_type.vlc.err

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

fail-bad_array_initialization.vlc

```
/*Number of elements in array does not match specified size*/
int[5] a = [1]

int def vlc():
    return 0
```

fail-bad_array_initialization.vlc.err

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

fail-bad_return_type.vlc

```
/*Return statement doesn’t match type*/
int defg add(int x, int y):
    float index = 1.0
    return index

int def vlc():
    return 0
```

fail-bad_return_type.vlc.err

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

fail-Boolean_condition.vlc

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

fail-Boolean_condition.vlc.err

```
Fatal error: Conditional_must_be_a_boolean
```

fail-Cannot_perform_operation_on_array.vlc

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

fail-Cannot_perform_operation_on_array.vlc.err

```
Fatal error: Cannot_perform_operation_on_array
```

fail-Cannot_perform_operation_on_string.vlc

```
int def main():
  string hi = "hello"
  if(!hi):
    print("wrong")
  return 1
```

fail-Cannot_perform_operation_on_string.vlc.err

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

fail-Constants_missing_in_defg.vlc

```
int defg test(int a):
  scale = 3
  return a

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

fail-Constants_missing_in_defg.vlc.err

```
Fatal error: Constants_missing_in_defg
```

fail-defg_reinitialize.vlc

```
/*Constant input re-initialized in defg*/
int defg add(int x, int y):
    int scale = 5
    return scale * (x + y)

int def vlc():
    int[5] a = {1,2,3,4,5}
    int[5] b = {1,2,3,4,5}
    int[5] c = ~map(add, consts(scale = 4), a, b)

    return 0
```

fail-defg_reinitialize.vlc.err

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

fail-Empty_array_access.vlc

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

fail-Empty_array_access.vlc.err

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

fail-Function_already_declared.vlc

```
int def one(int i):
  return 1

int def one(int i):
  return 1

int def main():
  return 1
```

fail-Function_already_declared.vlc.err

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

fail-Function_not_defined.vlc

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

fail-Function_not_defined.vlc.err

```
Fatal error: Function_not_defined
```

fail-Have_statements_after_break.vlc

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

fail-Have_statements_after_break.vlc.err

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

fail-Have_statements_after_return.vlc

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

fail-Have_statements_after_return.vlc.err

```
Fatal error: exception
 ↪  Exceptions.Have_statements_after_return_break_continue
```

fail-Higher_order_function_only_takes_defg.vlc

```
int def test(int b):
  return 1

int def main():
  int[3] a = [0, 0, 0]
  int[3] c = ~map(test, a)
  return 1
```

fail-Higher_order_function_only_takes_defg.vlc.err

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

fail-Invalid_accessor_value.vlc

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

fail-Invalid_accessor_value.vlc.err

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

fail-multidec.vlc

```
/*Multiple declarations of the same variable*/
int a = 5
int a = 6

int def vlc():
    return 0
```

fail-multidec.vlc.err

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

fail-Name_not_found.vlc

```
a = 5

int def main()
  return 1
```

fail-Name_not_found.vlc.err

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

fail-nomain.vlc

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

fail-nomain.vlc.err

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

fail-No_strings_allowed_in_gdecl.vlc

```
int defg test(string a):
  return a

int def main():
  return 1
```

fail-No_strings_allowed_in_gdecl.vlc.err

```
Fatal error: exception
 ↪  Exceptions.NO_STRINGS_ALLOWED_IN_GDECL
```

fail-nomain.vlc

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

fail-nomain.vlc.err

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

fail-Nonarray_passed_into_gdecl.vlc

```
int defg test(int i):
  return 1

int def main():
  int a = a
  int c = ~map(a)
  return 1
```

fail-Nonarray_passed_into_gdecl.vlc.err

```
Fatal error:
 ↪ Nonarray_argument_passed_into_higher_order_function
```

fail-nonconstdefg.vlc

```
/*Constant inputs to defg not constant*/
int defg add(int x, int y):
    scale = 5
    return scale * (x + y)

int def vlc():
    return 0
```

fail-nonconstdefg.vlc.err

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

fail-Out_of_scope.vlc

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

fail-Out_of_scope.vlc.err

```
Fatal error: exception
 ↪  Exceptions.Variable_not_found_in_scope
```

fail-predefined_defg.vlc

```
/*Print function in defg*/
int defg vector_add(int a, int b):
    print(a)
    return a

int def vlc():
    return 0
```

fail-predefined_defg.vlc.err

```
Fatal error: Invalid_function_in_defg
```

fail-Type_mismatch.vlc

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

fail-Type_mismatch.vlc.err

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

fail-undefined_defg.vlc

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

    return 0
```

fail-undefined_defg.vlc.err

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

fail-uninitialized_variable.vlc

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

fail-uninitialized_variable.vlc.err

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

fail-unmatching_args.vlc

```
/*PTX arguments do not match*/
int defg add(float x, int y):
    return (x + y)

int def vlc():
    return 0
```

fail-unmatching_args.vlc.err

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

fail-unsupported_defg_args.vlc

```
/*Unsupported PTX type as argument*/
string defg add(string x, string y):
    return x

int def vlc():
    return 0
```

fail-unsupported_defg_args.vlc.err

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

fail-Variable_not_declared.vlc

```
b = 5

int def main():
  return 1
```

fail-Variable_not_declared.vlc.err

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

fail-Variable_not_found_in_scope.vlc

```
int def main():
  int i
  for (i=0, i<5, i++):
    int j = j + i

  return j
```

fail-Variable_not_found_in_scope.vlc.err

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

fail-Void_type_in_gdecl.vlc

```
int defg test(string a):
  return a

int def main():
  return 1
```

fail-Void_type_in_gdecl.vlc.err

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

fail-wrong_array_type2.vlc

```
/*Array type not supported*/
string[2] wrong_array = ["hi", "hello"]

int def vlc():
    return 0
```

fail-wrong_array_type2.vlc.err

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

fail-wrong_array_types.vlc

```
/*Different types in array declaration*/
int[2] wrong_array = [1.0, 1]

int def vlc():
    return 0
```

fail-wrong_array_types.vlc.err

```
Fatal error: exception
 ↪  Exceptions.Array_elements_not_all_same_type
```