Yo: PLT Final Report

Mengqing Wang, Munan Cheng, Tiezheng Li, Yufei Ou

{mw3061,mc4081,tl2693,yo2265}@columbia.edu

Contents

1 Introduction

Yo is a user-friendly programming language for movie production. We offer a convenient way to do non-linear video editing. Users can produce videos from varieties of sources such as images or existing video clips and apply system- or user-defined functions to perform seamless video editing such as clip construction, duration adjustment, subtitle burning. In this light, *Yo*'s objective is to facilitate editing on videos and less human effort needs to be involved.

1.1 Background

To reduce the learning curve for new users, *Yo* scripts borrows much grammar from Python and C++. The code is then compiled into C++ code and executed utilizing a collection of C++ libraries such as `libopenshot`[1].

1.2 Features

Yo is a statically and strongly typed programming language, which means the type for each variable, expression or function is determined at compile time and remain unchanged throughout the program. *Yo* does not perform type conversion any kind. since in *Yo*'s most important type, *clip*, integers

[1] https://launchpad.net/libopenshot

usually represent the number of frames, floating point numbers usually represent the length of time. It is better to distinguish them as totally different types.

Yo also has the following language features:

- Object-oriented programming: Object-oriented paradigm has great advantages including code reuse and encapsulation.
- Automatic garbage collection and easy interpolation with existing C++ code and libraries.
- Maximum code cleanliness: indent blocking, newline instead of colon between statements.

2 Tutorial

2.1 Installation

The installation guide below is fully tested on Ubuntu-Gnome 14.04 LTS.
Since *Yo* could provide various manipulation on videos, audio and images, the following libraries are needed to be installed. Most packages can be installed easily with a package manager (apt-get) on Debian based Linux.

```
buildessential cmake libavformat-dev libavdevice-dev libswscale-dev
libavresample-dev qt5-default qtbase5-dev qt5-qmake qtmultimedia5-dev
libmagick++-dev libuinittest++-dev libxinerama-dev libxcursor-dev
python3-dev libasound2-dev libconfig++ swig doxygen
```

The library `libopenshot` and `libopenshot-audio` is not available in a Debian package. please follow the installation guide [2] to get them installed.

2.2 Compiling and Running a *Yo* Program

A valid *Yo* program ended with extension `.yo`. To complie the program, go to the `bin/` directory and run

```
$ ./yo.sh yourprogram.yo
```

A config file name `config.ini` can be put in the same directory with `yo.sh` to define the output details of the videos including the width, height, fps(frame per second) and bit rate.

2.3 *Yo* examples

Here we take a quick look at the *Yo* programs. The following program creates a time-elapse video with thousands of photographs in directory `photo/` with only 5 lines of code:

```
# read all pics in directory photos/ and create a clip
# for each of the pic
photos = Clip[]("photo/")
# create mymovie
mymovie = Clip()
# set the playing time of every pic as 1 frame (2 - 1 = 1)
```

[2] http://openshot.org/files/libopenshot/InstallationGuide.pdf

```
# and concatenate it to the end of the main clip
for p in photos:
    mymovie = mymovie & p[1:2]
# save the file
mymovie.save("timelapse.webm")
```

The following program adds a bunch of white flashes on a video.

```
# calculate the fibonacci numbers
fib = Int[]()
fib.add(1)
fib.add(1)
for i = 2 to 12:
    fib.add((fib[i - 1]) + (fib[i - 2]))
# read a clip and select the first 420 frames
a = Clip("Muppets.mp4")[0:420]
for i = 1 to 12:
    # create a clip with pure white color for 10 frames
    white = Clip("white.png")[1:11]
    # set the alpha value to 0.0 at 0th frame
    white.alpha @ 0 = 0.0
    # set the alpha value to 1.0 at 5th frame
    # Yo will do interpolation
    white.alpha @ 5 = 1.0
    # put the white screen on top of the original clip
    # with an offset of fibonacci numbers
    a = a ^ white @ (fib[i])
a.save("flashwithfib.webm")
```

Yo is also capable for video content analyzing. User may get a frame of a clip at a specific time, a pixel of a frame at a specific coordinate, and the RGB values of a specific pixel. Various analyzing can be developed based on this. The following program analyzes an input video and cut off the black screen at the end of the video.

```
# a function definition, reads Clip and Int, returns with Bool
func isblack(a: Clip, f: Int) -> Bool:
    for i = 300 to 350:
        for j = 200 to 250:
            # get pixel at frame f at coordinate (i,j)
            p = a<i!j>@f
            # get RGB values of the pixel
            if (p.R == 0) && (p.G == 0) && (p.B == 0):
                return true
    return false

a = Clip("abcde.webm")
cuttime = 0
for time = 1 to 180:
    # if a black screen detected
    if isblack(a,time):
        cuttime = time
        # log it to standard output
        log("black screen detected, cut at:")
```

```
        log(cuttime)
        break

b = a[1:cuttime]
b.save("cutabcde.webm")
```

3 Language Reference Manual

3.1 Syntax Notations

In this section, we define types or identifiers in regular expression. The following notations are used to show lexical and syntactic rules.

- Dash – is a shorthand for writing continuous elements.
- Brackets `[ ]` enclose optional items and select exact one of them. If there is a caret `^` in the `[^]`, it selects exact one of character that not belongs to the following list, for example `[^a-z]` means any character other than `a-z`.
- Parenthesis `( )` enclose alternate item choices, which are separated from each other by vertical bars `|`.
- Asterisks $\star$ indicate items to be repeated zero or more times.
- Question mark `?` is a sign of option.
- Double colon with an equal sign `::=` is used for definition.
- Braces {n} matches when the preceding character, or character range, occurs n times exactly.
- `{n,m}` matches when the preceding character occurs at least n times but not more than m times, for example, `ba{2,3}b` will find `baab` and `baaab` but not `bab` or `baaaab`. Values are enclosed in braces.

Below we will write *Yo*'s formal syntax definition in gray-background box. The terminals are marked in bold while non-terminals are in regular font.

```
c ::= a | b
```

We will also give examples in the white box.

```
This is an example
```

3.2 Lexical Conventions

This chapter presents the lexical conventions of *Yo*. This section describes which tokens are valid, including the naming convention of identifiers, reserved keywords, operators, separators and whitespaces.

3.2.1 Comments

Single line comment is made with a leading `#` in the line:

```
# This is a single line comment
```

Multi-line comment starts with `#(` and ended with `#)`

```
#( This is a multiline
comment #)
```

Nested comments are not allowed in *Yo*.

3.2.2 Identifiers

An identifier of *Yo* is a case-sensitive string different from any rescerved word (see next subsection). It starts with a letter or an underscore, optionally followed by a series of characters (letter, underscorce, number). The length varies from 1 to 256.

Formally, an identifier can be any non-reserved word expressed in regular expression as

```
Identifier ::= [a-zA-Z_][a-zA-Z0-9_]{0,255}
```

Legal examples:

```
_number _number1 number2 number_3 Number
```

Illegal examples:

```
2num *num func $2 Int Double Bool
```

Note that `Int, Double, Bool` are illegal because they are keywords. A list of keyword can be found in next subsection.

3.2.3 Reserved words

This is a list of reserved words in *Yo*. Since they are used by the language, these words are not available for naming variable or functions. The reserved words are consistent of keywords, built-in-type words and special constants.

break	continue	else	eval	for	func	global	if	in	return	struct	while

Table 1: List of keywords in *Yo*

Bool	Int	Double	log	true	false

Table 2: List of built-in-type words, special constants in *Yo*

3.2.4 Operators

An operator is a special token that performs an operation, such as addition or subtraction, on either one, two, or three operands. A full coverage of operators can be found in a later chapter, See chapter **Expression and Operators**.

3.2.5 Separators

A separator separates tokens. White space (see next section) is a separator, but it is not a token. The other separators are all single-character tokens themselves:
`( ) [ ] ,`

3.2.6 New Line

A physical line ends with an explicit `\n` input from the user while a logical line contains a complete statement. A logical line can be consist of multiple physical lines, all except the last one ending with an explicit `\`.

```
line 1 \
  line 1 continued \
 line 1 last line
```

3.2.7 Whitespace

Whitespace characters such as tab and space are generally used to separate tokens. But *Yo* is not a free-format language, which means in some cases, the position and number of whitespaces matters to the code interpretation. Leading tab whitespace is used to denote code blocks and to compute the code hierarchy (similar to curly brackets in C-family languages). Briefly, an extra leading tab lowers the level of this line in the code hierarchy.

In contrast to *Python*, *Yo* only accepts `\t` for leading indent, and space is not allowed. In other words, space should not appear at the beginning of any line (except for a continuing physical line where all the leading whitespaces are ignored).

```
im_a_parent
    im_a_child
        im_a_grandchild
    im_another_child
        im_a_grandchild
```

Usually, `for`,`while`,`if`,`else if`,`else` and function definition may start a new code block. The code block ends with an un-indent. In the above example `im_a_child` and `im_another_child` are at the same code indention level.

3.3 Types

3.3.1 Built-in Types

Below we list the built-in types in *Yo*. As they are used as the building blocks for the program, *Yo* provides *literals* to initialize them conveniently in users' source code. The operators on this types are covered in Section 3.4.

- `Int` 32-bit signed integral number, ranging from $-(2^{31})$ to $2^{31}-1$. The literal has to be represented in decimal:

  ```
  IntLiteral ::= [0-9]+
  ```

  For example:

```
086 - 320 +300
```

Yo does not support the leading positive/negative sign (and in most cases, negative number would not be used). But user can still create negative numbers by subtracting from zero

```
0 - 5 # i.e., -5
```

- `Double` 64-bit double-precision floating number. The literal is represented as follows:

```
DoubleLiteral ::= [0-9]*.[0-9]+
```

Note that the dot and the fractional number is compulsory (otherwise it can be identified as `Int`). Examples:

```
32.45 - .5
```

- `Bool` Binary value of either `true` or `false`.

```
BoolLiteral ::= true | false
```

- `String` A contiguous set of characters. The literal has zero or more characters enclosed in double quotes. A character can be a regular character or an escape sequence. Escape sequences are listed in Table 3.

```
StringLiteral  ::= "StringCharacter"

StringCharacter ::= [^"\'] StringCharacter
                  | [^"\']
                  | EscapeSequence StringCharacter
                  | EscapeSequence
EscapeSequence ::= \b | \t | \n | \r | \" | \' | \\
```

EscapeSequence	Meaning
\b	Backspace
\t	Horizontal tab
\n	New line
\r	Carriage return
\"	Double quote
\'	Single quote
\\	Backslash

Table 3: Escape Characters

Examples:

```
"abc" "9j32 f0kca0" "Hello\nYo!"
```

MediaType	Extension Format
Picture	jpg, jpeg, png, bmp, gif
Video	webm, f4v, mov, rmvb, mp4, rm, wmv, avi, flv, 3gp, mkv

Table 4: Suppported formats

3.3.2 Clip type

Since *Yo* features video editing, it has a `Clip` type built-in, which would be the focus of user's code. Clip can be constructed from an image or video (format see table 4) from an existing source (i.e. a file on the disk) or transformed from an existing *Clip*.

Operations on `Clip` can be found in section 3.4.7.

3.3.3 User Defined Type

A type can be defined at the most top level of the program or be nested in another type. The definition starts with the keyword `type` and the type identifier followed by a colon. Conventionally, we use capitalized identifiers for type names.

```
type_decl ::= type type_name : NEWLINE INDENT type_element_list DEDENT
type_name ::= identifier
```

Then follows the declaration of zero or more members variables or member function types.

```
type_element_list ::=
    | empty
    | mem_var_decl  type_element_list
    | mem_func_decl type_element_list
    | mem_type_decl type_element_list
```

A member variable is declared by writing the member identifier and its type name or type definition.

```
value_element_decl ::= identifier : type_name
```

For example,

```
zindex: Double
```

Yo does not support variable initalization in type definition. Nevertheless, user can choose to initialize them in the constructor (see 3.3.4).

```
type A:
    var1: Double
    var2: Int
    var3: A

type B:
    type C:
        var1: String
    var1: A
    var2: Int
    var3: B.C
```

In the example above, `A` has a member `var3` of its own type `A`. In type `B`, member `var1` is of `A` type (defined on the same level as B) and the type of `var3` is defined inside `B` (`C` is called an *inner type* of `B`). To reference a nested type, users have to use the member operator (see Section 3.4.8), e.g. `B.C`

The member function defines the executable code that can be invoked about an instance of this type. Details about function definition is covered in Section 3.6.

3.3.4 Type Constructor and Object Instantiation

The work of creating an object (or instance) of the type is done in the `eval` function, which is required for every type. Although the details about function will be elaborated in Section 3.6, we emphasize here that the `eval` function for a value type has to take an object of its own type as the first parameter and to return an object of this type.

```
type ClipColorMode:
    mode: String
    degree: Int
    func eval (self: ClipColorMode, m: String, c: Int) -> ClipColorMode:
        mode = m
        degree = c
        return self
```

We can instantiate an object of a type in an expression of type name followed by a list of parameters in parenthesis. Formally,

```
TypeInstantiate ::= type_name ( ParamList )
Paramlist       ::=  parameter (: parameter)* [,]
parameter       ::=  (identifier , type ) | ( paramlist )
```

Here is an example of object instantiation:

```
ccMode = ClipColorMode("RGB", 4)
```

The order of type and function definition does not matter in the *Yo* script (the compiler will scan the code several times to index all user-defined types) so it is possible to reference a type defined after it.

3.3.5 Array

An `Array` holds a sequence of (zero or more) elements of the same type. It is considered as a *meta-type* and has to be specified by the type of the objects it contains. To represent the type of an `Array` of `T` type, we use a pair of square brackets immediate after the type name.

```
anArrayForTypeT: T[]
```

The `Array` literal is represented using a pair of square brackets. Elements in the array are separated by comma.

```
array_literal ::= [ arg_expr_list ]
arg_expr_list ::=
    | expr
    | arg_expr_list COMMA expr
```

where `expr` is defined in section 3.4.1. For example,

```
a = [1,2,3,4,5,6] # an Int array
```

3.4 Expressions and Operators

This section describes the expression in *Yo*.

3.4.1 Expressions

An expression consists of at least one operand and zero or more operators. Operands are typed objects such as literals, variables, and function calls that return values.

```
expr ::=
    Literal
    | ID
    | ( expr )
    | expr op expr
    | expr . ID                 // Dot Expression
    | expr [ expr ]            // Array Index
    | expr [ expr : expr ] // Array  Slicing
    | expr ^ expr @ expr                 // Clip Cascading
    | expr coord AT expr          // Clip Pixel Access
    | expr AMPERSAND expr         // Clip Concatenation
    | expr . ID ( arg_expr_opt )         // Object function call
    | ID ( arg_expr_opt )                // Simple function call
    | array_constructor ( arg_expr_opt ) // Array Constructor
array_constructor ::=
    | expr ()                          // Simple array constructor
    | array_constructor ()             // Composite array constructor

coord:
    | < expr ! expr > // Coordination

op = + | - | * | / | % | = | == | != | < | <= | > | >= | && | || |
```

For details of the definition of function calls, see Section 3.6.2.

3.4.2 Arithmetic operators

Yo provides operators for standard arithmetic operations: addition, subtraction, multiplication, and division, along with modular division. Here are some examples:

The expression type for these operators can be found in table

```
4 + 2              # addition
x - 5              # substraction
x * y              # multiplication
4 + 2.2            # error! 4 is Int but 2.2 is Double
```

3.4.3 Array Access operators

The `Array` is indexed and can be accessed using integral subscript starting from zero.

```
a = [1,2,3,4,5]
b = a[0] # 1
a[1+2*2] # out of range exception
```

Operator	Type 1	Type 2	Return type
+	Int	Int	Int
	Double	Double	Double
	String	String	String
-, *, /	Int	Int	Int
	Double	Double	Double
%	Int	Int	Int
==	Bool	Bool	Bool
	Int	Int	Bool
	Double	Double	Bool
	String	String	Bool
<, <=, >=, >	Int	Int	Bool
	Double	Double	Bool
&&	Bool	Bool	Bool
||	Bool	Bool	Bool

Table 5: Basic operators

3.4.4 Comparison operators

Comparison operators are used to determine how the value of two operands relate to each other: are they equal to each other, is one larger than the other, is one smaller than the other, and so on. The comparison result is either true or false, respectively. The result of such an expression is either true or false.

```
a > b   # a is greater than b
a >= b  # a is greater than or equal to b
a == b  # a is equal to b
a < b   # a is less than b
a <= b  # a is less than or equal to b
a != b  # a is not equal to b
```

3.4.5 Logical operators

Logical operators test the truth value of a pair of operands. Only boolean value (such as `a==b`, `true`) could be the legal operand of logical operators, integers are illegal logical operands.

```
a && b  # and
a || b  # or
```

3.4.6 Assignment operators

Assignment operators store values in variables. We only allow one assignment at a time. After the two statements below are executed, both `a` and `b` are equal to 42.

```
a = 42
b = a
```

3.4.7 Clip operations

Yo allows two clips to be concatenated easily.

```
# concatenate a clip2 to the end of clip1 and form a new Clip
clip3 = clip1 & clip2
```

Yo also subscripts the Clip so that users can extract a range of clip. The two parameters have to be both `Int` type or both `Double` type. `Int` type is used to access the clip by frame index while `Double` is used to access clip by time. The user has make sure that the second parameter is greater than the first one numerically (an exception will be thrown otherwise).

```
# get 2.4s to 8.0s of the clip (returns a clip)
clip1 = clip[2.4:8.0]

# get the 2nd to 95th frame in the clip (returns a clip)
clip2 = clip[2:96]

# get the 2nd to 95th frame in the clip (returns a clip containing 1 frame)
clip3 = clip[2:3]
```

The clip can be layered on the top of another clip to form a new clip. We use a ternary operator, cascade operator, `a ^ b@c` to denote putting b on top of a, by an offset of `c` seconds if it is an `Double` or of `c` frames if it is a `Int`.

```
# put clip2 on top of clip1, with time offset of 2.4s
clip1 ^ clip2 @ 2.4
```

Keyframe can be added to any attribute of a clip to form animation. A key frame in animation and filmmaking is a drawing that defines the starting and ending points of any smooth transition. The drawings are called "frames" because their position in time is measured in frames on a strip of film. *Yo* will do interpolation between keyframes. User has to set the value at a certain time by the pattern `clip.attribute @ time = value`. Once again, time can be `Int` or `Double` to access by frame index or time.

```
# a fade-in effect
clip1.alpha @ 0.0 = 0.0
clip1.alpha @ 2.4 = 1.0
# the clip flies from right to left
clip1.location_x @ 0,0 = -1.0
clip1.location_x @ 2.4 = 1.0
# zoom in effect
clip1.scale_x @ 0.0 = 1.0
clip1.scale_y @ 0.0 = 1.0
clip1.scale_x @ 2.4 = 2.0
clip1.scale_y @ 2.4 = 2.0
# a 360 degree rotation of video
clip1.rotation @ 0.0 = 0.0
clip1.rotation @ 2.4 = 360.0
```

A pixel of a specific coordinate at a specific time of a clip can be got by frame operator. The type of pixel contains three `Int` values which represent R,G and B.

```
# p is a pixel from clip1 at time 2.4s, coordinate(20,30)
```

```
p = clip1 <20!30> @ 2.4
```

3.4.8 Member Access operators

The member access operator `.` is used to access the members of an object: object name followed by the member name

```
# Get a member variable in object myObject
a = myObject.myVariable
# Call a member function defined in object myObject
b = myObject.myFunction(0)
```

3.4.9 Operator Precedence and Associative Property

When an expression contains multiple operators, such as `a + b * f()`, the operators are grouped based on rules of precedence. For instance, the meaning of that expression is to call the function f with no arguments, multiply the result by b, then add that result to a. The following is a list of types of expressions, presented in order of highest precedence first. Sometimes two or more operators have equal precedence; all those operators are applied from left to right unless stated otherwise.

1. Function calls and membership access operator expressions.
2. Multiplication, division, and modular division expressions.
3. Addition and subtraction expressions.
4. Greater-than, less-than, greater-than-or-equal-to, and less-than-or-equal-to expressions.
5. Equal-to and not-equal-to expressions.
6. Logical AND expressions.
7. Logical OR expressions.
8. Clip concatenation.
9. Clip layering.
10. All assignment expressions.

3.5 Statements

3.5.1 Overview

A statement could be either a simple statement (using stmt in short) or a compound statement (using compound_stmt in short), formally speaking, they could be defined as:

```
stmt ::= expr NEWLINE
       | log_stmt
       | return_stmt
       | continue_stmt
       | break_stmt
compound_stmt ::= if_stmt
```

```
          | while_stmt
          | for_stmt
 suite ::= NEWLINE INDENT statement+ DEDENT
 statement ::= stmt NEWLINE | compound_stmt
```

3.5.2 log

```
 log_stmt ::= log (expr)
```

`log` evaluates the `expr`, acceptable types are `Int`, `Double`, `String`, and writes the resulting object to standard output as a string.

```
log ("A string")
# Output: A string

a = 1
log (a)
# Output: 1

b = 1.01
log (b)
# Output: 1.01
```

3.5.3 if

The `if` statement is used for conditional execution:

```
 if_stmt ::=  if expr : suite
              (elif expr : suite)*
              [else : suite]
```

It selects exactly one of the `suite` by evaluating the `expr` one by one (start from the `expr` next to the `if`, and `expr` next to the `elif` sequentially) until one is found to be true; then the `suite` corresponding to this `expr` is executed, and no other part of the `if` statement is executed or evaluated. If all exprs are false, the `suite` of the `else` clause, if present, is executed.

```
a = 1
if a==1:
    log (a)
    log (" is true")
# Output: : 1 is true

a = False
b = 1
if a:
    log (b)
    log (" is true")
elif !a:
    log (1-b)
    log (" is not true")
else:
    log ("Else clause is executed")
# Output: 0 is not true
```

3.5.4 while

The `while` statement is used for repeated execution as long as the `expr` is true:

```
while_stmt ::=  while expr : suite
```

This repeatedly tests the `expr` and, if it is true, executes the `suite`; if the `expr` is false (which may be the first time it is tested) the loop terminates.

```
a = 0
while a < 3:
    a = a + 1
    log (a)
    log (" ")
# Output: 1 2 3
```

3.5.5 for

The `for` statement is used to iterate over the elements of an `array` or continuous integers:

```
for_stmt ::=  for expr (= expr (to|downto) expr | in array) : suite
```

An iterator is created according to "`= expr (to|downto) expr | in array`":

If the expression is "`= expr to expr`", the iterator is initiated as the first expr, ended as the second expr; if the expression is "`= expr downto expr`", the iterator is initiated as the second expr, ended as the first expr. In these two conditions, both `expr` must be integers.

If the expression is "`in array`", the iterator is initiated as the first element of the array, ended as the last element of the array.

The `suite` is then executed once for each item provided by the iterator, in the order of ascending indices. Each item in turn is assigned to the target list using the standard rules for assignments, and then the `suite` is executed. When the items are exhausted (which is immediately when the sequence is empty), the loop terminates.

Caution: any operate aim at iterator in the for loop is an undefined behavior; if you use `to`, the first `expr` must be less than the second, if you use `downto`, the second `expr` must be less than the first, otherwise the behavior is undefined.

```
for i = 1 to 3:
    log (i)
    log (" ")
# Output: 1 2 3

for i = 7 downto 3:
    log (i)
    log (" ")
# Output: 7 6 5 4 3

arr = [1,2,4,6]
for i in arr:
    log (i)
    log (" ")
# Output: 1 2 4 6
```

```
arr2 = ["a","b","c","d"]
for i in arr2:
    log (i)
    log (" ")
# Output: a b c d
```

3.5.6 continue

```
continue_stmt ::= continue NEWLINE
```

`continue` may only occur syntactically nested in a `for` loop or `while` loop, but not nested in a function or class definition within that loop. It continues with the next cycle of the nearest enclosing loop.

```
arr2 = ["a","b","c","d"]
for i in arr2:
    if i=="b":
        continue
    log (" ")
    log (i)
# Output: a c d
```

3.5.7 break

```
break_stmt ::=  break NEWLINE
```

`break` may only occur syntactically nested in a `for` or `while` loop, but not nested in a function or class definition within that loop. It terminates the nearest enclosing loop.

If a `for` loop is terminated by `break`, since the loop iterator is a local variable, its current value can not be used after the break.

```
arr2 = ["a","b","c","d"]
for i in arr2:
    log (" ")
    log (i)
    if i=="b":
        break
# Output: a b
```

3.5.8 return

```
return_stmt ::=  return expr | return NEWLINE
```

`return` may only occur syntactically nested in a function definition, `return` leaves the current function call, if the optional value `expr` exists, with `expr` as return value.

```
func foo(a: Int, b: Int):
    return a+b
```

3.6 Functions

This section discusses how to declare and define functions, specify parameters and return types and call functions.

3.6.1 Function definition

A function definition will create a user-defined function object and provide the information below

- the types and values of parameters
- the types and values returned by the function
- the logic composed of a collection of statements that are executed when the function is called

The syntax for a function definition is shown below:

```
func_decl       ::=  func func_name ( parameter ) -> type :
                     NEWLINE INDENT suite DEDENT
func_name       ::=  identifier
paramlist       ::=  parameter | (paramlist ,  parameter)
parameter       ::=  identifier : type
```

An example which shows how the function is defined is

```
func FUNCNAME (param1: paramtype1, param2: paramtype2) -> returntype:
    # Function logic goes here
```

In the example, *func* is a keyword which indicates that a function type is defined. Parameters are statically typed with two components: parameter name as an identifier and parameter type as a type.

The function definition is an executable statement which binds the function name of the local namespaces (defined in Section 3.7.2) to the function objects. The function objects create a method `eval`. The shorthand for `eval` is to name the instance and follow it with parentheses containing the arguments to the call.

The function definition does not execute the function body until the function object is called, where we define the function call in Section 3.6.2.

3.6.2 Function calls

A function call is defined as:

```
function call         ::=  eval ( argument_list [,] )
argument_list         ::=  expr | (argument_list , expr)
```

Expressions can be passed as arguments to function calls. Arguments are values passed to a function object when calling the function, which is defined in the following section. All argument expressions are evaluated before the call is attempted.

An example of the function call is

```
a = 5
b = 3
c = factorial(a+b)
```

3.6.3 Function Return Types

A function returns an object as the execution result. Return types need to be specified during function definition.

3.6.4 Recursion

Recursion is a property that a function can be called by themselves. The following example shows recursion is useful in the calculating an integer's factorial:

```
func factorial (n: Int) -> Int:
    if (n==1):
        return 1
    else:
        return factorial(n-1)*n
```

3.6.5 Built-in functions

There are built-in log functions defined in the following Table 6.

Function	Argument
log	Int, Double, String, Bool, Clip

Table 6: Built-in functions

3.7 Program Structure and Scope

3.7.1 Program Structure

Yo program must exist entirely in a single source file, with a ".yo" extension. It consists of a number of function/type declarations or statements.

```
program ::= decls

decls ::= empty
        | type_decl
        | func_decl
        | stmt
```

The position of function/type declarations in the source code does not matter. This means a function can call another defined later and that the member of a user-defined type can use another type defined on the bottom of the source code.

3.7.2 Namespaces

A namespace *Yo* is a mapping from names to objects. Namespaces include: the set of built-in types (containing functions such as log()); the global types including built-in types and user-defined types; and the local types in a function invocation. In a sense the set of attributes of an object also form a namespace.

3.7.3 Scope

A scope is a textual region of a program where a namespace is directly accessible. "Directly accessible" here means that an unqualified reference to a name attempts to find the name in the namespace. There are two hierarchies of namespaces in *Yo*:

- Local: the innermost scope, which is searched first, contains the local names

- Global: the next-to-last scope contains the current program's global names

Yo follows the rule of Local → Global, where the right arrow denotes the namespace-hierarchy search order.

An example shown here gives how the two different scopes of variable are accessed:

```
p1 = "global variable"

func a_func(p1: String) -> Void:
    log(p1)

a_func("local variable")
#p1: local variable
log(p1)
#p1: global variable
```

4 Project Plan

4.1 Processes

4.1.1 Planning

We had a two to four hour regular meeting each Wednesday. In the early stage of these meetings, we discussed project milestones, responsibilities to each member. After that, we exchanged experience of updating versions and modifying functions of *Yo*.

When the topic is settled down, we also scheduled a short but meaningful weekly meeting with our benevolent T.A., Richard. He instructed us modifying the manual, language features and answered our concerns.

4.1.2 Specification

We discussed the main contents of proposal, LRM at the meeting, and pushed the outline to a real-time collaborative writing platform Overleaf [3]. We wrote our own part during the week and discussed the problem at the next meeting. Each group member had a portion to write and another portion to proofread. Once we started coding, any updates that needed to be made were in charged by the person coding that portion of the language, if he could not do it, he might ask other group members for suggestion.

4.1.3 Development

First of all, we together implemented preprocessor, scanner, parser and ast as soon as possible, then our language guru designed the interface of the remain works, they were typeReader, semanticAnalysis, semantic abstract syntax tree and codeGeneration.

Second, each member of our group was given a slice of our language to implement. Mengqing implemented typeReader, Munan implemented semanticAnalysis and semantic abstract syntax tree, Yufei implemented codeGeneretion. Tiezheng did almost all the other things such as wrote Makefile, test suites, main function ,and dealed with `libopenshot` rebase the code and write wrappers. We used GitHub to track our code, LRM as part of the reference on how to implement our section (since there were so many dependency on `libopenshot`, we might choose to compromise with its interface

[3] https://www.overleaf.com/

Sept 17th	Milestone: first group meeting, brain storming
Sept 24th	Second group meeting, decided to do movie editing language
Sept 25th	Proposal started
Sept 29th	Proposal finished
Sept 30th	**Project proposal due**
Oct 14th	Scanner finished
Oct 25th	LRM finished
Oct 26th	**LRM due**
Nov 5th	Parser, ast, bash makefile finished
Nov 15th-17th	Hackthon, HelloWorld finished
Nov 16th	**Hello world demo due**
Nov 17th-Dec 3rd	Installed libopenshot, wrote new test cases and implemented clip and frame
Dec 14th-Dec 20th	Semantic, sast, code generation finished
Dec 20th-Dec 21st	Finished final project, ppt and other works
Dec 22nd	**Final project report**

Table 7: Project timeline

and change our features). Changes were proposed by anyone in the group, the structurally level change must be approved by the language guru, and the feature addition or deletion is decided by the manager.

4.1.4 Testing

Normally, each group member wrote unit tests to ensure their slice of the code did not have syntax error, at the end of each stage of development, architect organized us testing from the start to the end to ensure slice of the code worked as anticipated. This integration testing took the form of "Hello World" programs. Any failed tests were addressed as soon as the failure was discovered.

4.2 Project Timeline

The milestones of the project timeline are presented in table 7

4.3 Team Responsibility

The contribution or responsibility of the project could be seen in table 8.

4.4 Software Develop Environment

We use the following tools and languages:

- Compiler implementation: OCaml
- *Yo* code preprocessing: Python
- Object code: c++11
- Video interface: `libopenshot`
- Testing environment: shell script
- Version control system: Github

Mengqing	Super fire fighter, test cases designer Design function call Implement type_reader Document works (such as Proposal, LRM, final project and PPT)
Munan	Powerful language guru, designer, OCaml expert Implement parser, semantic_analysis, ast, sast Design "type" Unification: uniform all function, type, built-in type as "type"
Tiezheng	Versatile architectural designer, test designer, Github administrator, commitment calculator Implement *Yo* preprocessing Write bash files and many other automatic test-run suits utils Implement the library interface of `libopenshot` Handle all the other tough problems
Yufei	Lucky group timer: organize weekly meeting, make sure every milestone delivered in time Decide which feature should be included in the language Implement scanner, code_generation Laugh at them all the time

Table 8: Team responsibility

5 Architectural Design

5.1 Preprocessor

The preprocessor takes the whitespace delimited *.yo source code and produces an output that can be easily tokenized by the scanner. The preprocessor goes line by line and calculates where new blocks of code are created based on the indentation level, as well as replacing newlines with semi colons to mark the end of statements, except for continue line separator '. Lines ended with 'áre concatenated together into one line. In-line comments and block comments are also removed in the preprocessor this is required to correctly calculate correct indentation levels. In addition to recognize scope correctly from whitespaces, the preprocessor also checks for certain invalid characters (brackets and tabs, for they will change the structure of the code and break scanner) and throws an error if they are encountered. The preprocessor is implemented in Python. It is called by the execution script, and produces an intermediary output file, which would be deleted if the program is correctly complied in the end.

5.2 Scanner

The scanner, written in OCamlLex, takes the intermediate output of the preprocessor (symbol stream) as input and tokenizes it to produce a token stream. The tokenization process provides basic syntax checking, rejecting programs that contain illegal symbols and illegal combinations of symbols.

5.3 Parser

The parser takes the token stream produced by the scanner as input and parses it to produce an abstract syntax tree (AST), which describes the overall structure of the program. ast.ml provides parser.mly with the acceptable structure of the AST, as well as the print function of AST nodes for debugging.

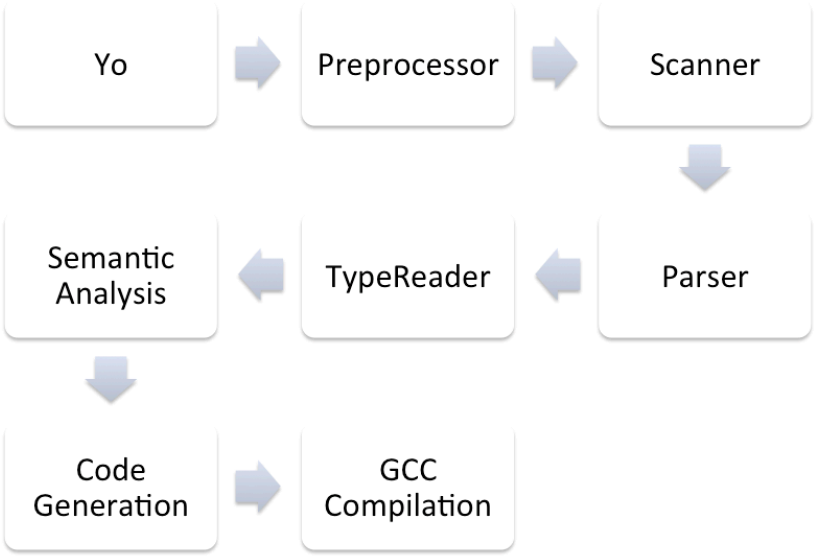


Figure 1: Architectural Design

5.4 TypeReader

Built-in types, user-defined types and functions are all resolved to types in TypeReader. TypeReader will perform two passes of scanning. The first pass is to resolve types and functions from definitions. Nested types and functions are hierarchically flattened. The second pass is to resolve members variables or argument lists and return types to existing types definition, which depends on the type is resolved from a type or a function. Each type will have a default constructor named eval. TypeReader can also resolve function overloading by defining one-to-one mapped eval functions from function definitions. Additionally, the typereader adds built-in information (i.e. built-in variables and functions) to the sast.

5.5 SAST

The analyzer takes the ast produced by the parser and analyzes it to produce a semantically analyzed abstract syntax tree (SAST). Like the AST, the SAST describes the overall structure of the program, but it also includes type information that was attached during the analysis process. sast.ml provides generate.ml with the acceptable structure of the SAST. The analysis process provides rigorous semantic checking, rejecting programs that violate type requirements (e.g. assigning a complex number to a variable declared as an integer), declaration requirements (e.g. using a variable that was not declared or attempting to declare a variable more than once), scope requirements (e.g. using a variable declared in another function), order requirements (e.g. calling a function before it is declared), and other language-specific requirements (e.g. not declaring a compute function).

5.6 Code generation

The generator takes the sast produced by semantic.ml and generates c++ code from it. Since we have invented many convenient operators for clip operation and file manipulation. The system architect reconstructed the video editing library, wrote many utility functions and repackaged it to new classes so

it could be easy for code generator to generate codes by simply calling a function and passing in the arguments. (e.g. a statement with the layering operator `clipbottom ^ cliptop @ shiftime` will be translated into `shared_ptr<_Clip> layering(Clip shared_ptr<_Clip> clipbottom, shared_ptr<_Clip> cliptop,double shifttime))`
In code generation a lot of new features in C++ 11 is used, such as `auto` , `shared_ptr` and so on which helps us resolve issues like garbage collection and type consistency.

6 Test Plan

During the development of Yo, we slice the work so that every team member can have an assignment to work on.

6.1 Testing Suite

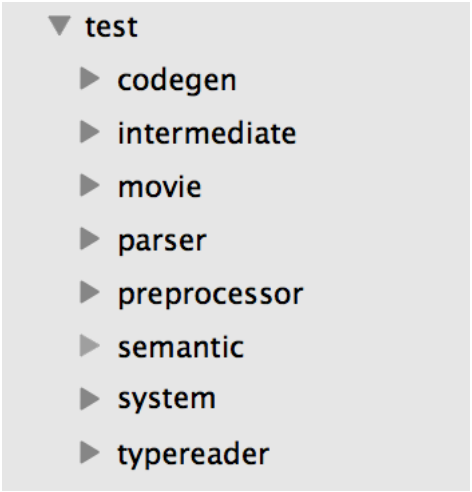
6.1.1 Unit Testing

We have a separate executable to test each functionality including parser test, typereader test, semantic test, generate test.

6.1.2 System Testing

We had full end-to-end testing from scanning a Yo program to generating a video. The most challenging part is how to verify a video. Since we have the built-in function log and it supports the pass the Clip as an argument, values in the member attributes of a clip can be output to a JSON file. We can therefore verify the correctness of a clip by comparing the output JSON file with the expected information using script.

Here is the hierarchical structure of Yo's testing suites:



6.2 Examples and Results

Here we give an example of system testing:

```
func isblack(a: Clip, time: Int) -> Bool:
    for i = 300 to 350:
        for j = 200 to 250:
            p = a<i!j>@time
            if (p.R == 0) && (p.G == 0) && (p.B == 0):
                return true
    return false

a = Clip("abcde.webm")
cuttime = 0
for time = 1 to 180:
    if isblack(a,time):
        cuttime = time
        log("black screen detected, cut at:")
        log(cuttime)
        break

b = a[1:cuttime]
b.save("cutabcde.webm")
```

More test cases can be found under the test folder in the source code. Here is the sample result from our movie test:

```
Running 01-basicio... Finished. Compare... OK. \\
Running 02-cutandlink... Finished. Compare... OK.\\
Running 03-keyframe... Finished. Compare... OK.\\
Running 04-readfolers... Finished. Compare... OK.\\
Running 05-arrpr.yo... Finished. Compare... OK.\\
Running 06-timeelapse... Finished. Compare... OK.\\
Running 07-analyze... Finished. Compare... OK.\\
Running 08-analyzewithfunc... Finished. Compare... OK.\\
Running 09-flash...  Finished. Compare... OK.\\
Running 10-analyzeanddelete...  Finished. Compare... OK.\\
Running 11-flashwithfib...  Finished. Compare... OK.\\
11/11 Testcases Passed.\\
Test Finished.\\
```

7 Lessons Learned

7.1 Mengqing Wang

The semester-long project was painstaking at the initial stage but I did enjoy it at the end. We had a hard time in coming up with a project idea at the beginning. There were a lot of domains that we felt interested in but when problems come when diving into the detail. Designing a programming language is not as easy as finding an area that we feel interested in. Feasibility matters much more. It turned out that our drastic discussion about feasibility saved us a lot of time in drafting LRM because we have already born very structural design in mind. Therefore the most important lesson that I learned is that a good design can save tons of work and faults during implementation.

Learning OCaml is a huge challenge for most of our team members. Functional programming has a completely different paradigm compared with imperative programming languages. After I worked out some practices on OCaml website, things turned more lovely as I could write small OCaml programs. It triggered me with more interest and passion in learning OCaml when professor talked a lot about

comparison of different programming languages. After learning OCaml systematically I have to say OCaml is the best fit for this project, for example, recursion and pattern matching can be solved in an efficient way, which appear in compiler implementation frequently.

Beyond these technical learning, I must say I learned so much in the passionate team collaboration throughout the term. There are always more tasks than what we have expected. I was always inspired by teammates and each of our teammates always take as more tasks as they can. It gives me so much positive encouragement to take active part in the project. However since we are all first-year master student, we have to take four courses at the same time. Time allocation becomes very challenging for us but we managed to learn how to set progressive milestone in this project.

7.2 Munan Cheng

Frankly, this is one of the best project experience so far. Every team member is intelligent, hard-working and particularly, encouraging. This is far different from many other team projects I have worked on in which some unlucky guy has to be the single weight carrier.

From my perspective, there were a number milestones that should be remembered. Finding an interesting topic to work on took us almost a week! But it was rewarding as it turned out working on something visually appealing could greatly alleviate our panic in our final sprint. It did not took long to figure out the working of scanner and parser, but designing a new language was a trial-and-error process. Back then, I was deeply impacted by the conciseness of python. I tried to get rid of all things unnecessary in the C++, such as explicit variable type declaration in the statement and function signature. But as I worked on the semantic analysis, I found out how important role these declarations have played in type checking in functions. This is one of the example that how this project has helped me to gain an understanding of the elements in existing languages.

Figuring out the whole pipeline helped our work division. To allow video editing features to be added incrementally without modifying the core compiler code, I found out having a type reading phase could make our compiler much more flexible. In this way, we can write much intensive work in C++ instead of hard-coding it in OCaml. But admittedly, OCaml offers much better guarantee of the program correctness than C++ thanks to its rigorous type checking system.

Finally, prioritization is extremely important in project management, especially when everyone has a number of other ongoing projects. Fortunately, we had great experts on the team to identify the next big-goal.

7.3 Tiezheng Li

Test-driven development would be a good idea for developing the project. Produce minimum features of the code to pass the smallest unit test and finally refactor the new code to acceptable standards. Untested code is meaningless.

The labor should not be simply divided by roles. If one member takes full charge of one component, say code generation. Then there is risk that the part could become bottleneck if any obstacles encountered, and other member are hard to help since they have to understand the code before any correction.

The architect should code quickly at the very beginning and build up the work-flow of the compiler. Just like the hello-world check for this semester. Then forget about the role, everyone is a developer. Do vertical slicing of the assignment and finish one module at a time.

Be very careful, when using third-party libraries. Better to choose one with detailed document and examples. Take a deep investigation into the libraries to see if there are known bugs or open issues. Like the video editing library we used in Yo, it is indeed strong in video related functions. But the dependency is complex and there is no guide nor examples for compile and run. There are

some issues like memory leak and type cast needed to be fixed manually. I had a really hard time understanding the whole bunch of codes and reconstruct the classes, redefined function interfaces and tests.

There are various ways of traveling to Rome. So when faced with an impossible or hard problem during implementation, identify the real constraints first. Just ask yourself: *Does this have to be done in this way?* or *Does it have to be done at all?* Before thinking outside the box it is necessary to find the box first.

Special thanks, to my excellent teammates, for the happy coding and playing hours we spent together. You make amazing happens.

7.4 Yufei Ou

First, I agree Tiezheng's opinion that it is better to do vertical slicing of the whole project. But the thing is that none of us is familiar with OCaml, therefore we had to learn from the start and finished scanner, parser, semantic analysis and code generation one by one. The shorthand of this development process is that, since any of them is dependent on the rest, we have to wait until the very end to test if they are correct.

It has been a really hard time to debug the whole program from the start (scanner) to the end (code generation) before deliver HelloWorld demo, I think the main problem is we focus on the integrity of the language and try to implement all of them before we have a whole program to test. Bad idea. Thanks to our magical architect, he had years of debugging experience and helped us get over this big problem and deliver HelloWorld successfully. But next time we should have a smart strategy and avoid implementing tons of code without testing, because we may not be that lucky again.

Second, as for organizing the group, the most important trick is to choose a good teammate, from this aspect, I think I am very successful in this course. All of my teammates have strong capability and know how to cooperate with each other. So it is not a challenge to schedule meeting or assign work. As a CS master student, everyone is busy finding job and preparing interviews, so it is important to know their personal schedule and make our timeline accordingly and flexible. Generally speaking, I plan to assign harder jobs to the stronger or less busy one, utilize the guilty of less contributive one and assign him some boring jobs. However, I did not have any chance to play these tricks, everyone was so passionate and dedicative, I just need to laugh at them.

I can't think a better way to learn a compiler than Prof. Stephen's way, and I admit OCaml is a very smart language is implementing compiler. But I still feel it very hard to understand everything in this course, the feeling of doing this project is quite strange, but it was a good experience to study and implement at the same time. And I do appreciate Richard's help.

8 Appendix

Detail	Author	Description
b340046	LeeTZ	Initial commit
113b3c6	LeeTZ	add proposal
898bbbe	LeeTZ	add lrm
1e4829c	LeeTZ	framework of lrm
a751796	LeeTZ	add to framework of LRM
fa4a375	Yufei Ou	add something for test
ff66193	Edward Cheng	add gitignore to doc

02f7029	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
bfda07c	mengqwang	add gitignore
60128f7	Edward Cheng	add microc compiler for start up skeleton
5796419	Edward Cheng	update .gitignore
0fa938c	LeeTZ	renew lrm
fdf9905	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
3d6ce3a	Yufei Ou	add scanner
65e0e7f	Yufei Ou	modified scanner double
cc3bd2e	LeeTZ	renew lrm
09763bb	LeeTZ	clean folders
5335297	Edward Cheng	update lrm
755eac9	Yufei Ou	change scanner token lexbuf
745186f	Yufei Ou	add brackets
a5a7a3d	Yufei Ou	modify the constant part
9730c03	Yufei Ou	finish statement
089dd04	Edward Cheng	parser expr
10f21b5	Yufei Ou	add scanner
ed463c7	LeeTZ	add preprocessor
c68ce6c	LeeTZ	add preprocessor
6dce834	Yufei Ou	Merge branch 'master' of https://github.com/LeeTZ/Yo into yo
02cfb04	LeeTZ	finish preprocessor and tested
01a8e7b	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
a6c7d6c	LeeTZ	renew proposal
1689621	Edward Cheng	parser add statement
d267ac2	Edward Cheng	Merge branch 'mc'
4b96dfb	Edward Cheng	update gitignore
963b608	Edward Cheng	Merge branch 'mc'
fcd8183	Yufei Ou	add ast
3ed7c1a	Edward Cheng	revise if statement
1fdfcb2	Yufei Ou	update stmt
5b19817	Yufei Ou	Merge remote-tracking branch 'origin/master' into yo
5e63772	Yufei Ou	merged
d1cd973	mengqwang	add type def
7f127c1	mengqwang	typedef
b51793a	Yufei Ou	add ast if
90b2fee	Yufei Ou	merged func def
a75e1b2	Edward Cheng	bug fix in parser
b4a4e82	Edward Cheng	bug fix in parser
1a2cd09	Yufei Ou	delete continue lexbuf
8d37360	mengqwang	func
6c7e183	mengqwang	func
c94d8ee	LeeTZ	modify the rules of comments
36449d8	mengqwang	bug fix for type
f7f9a86	mengqwang	Merge branch 'master' of https://github.com/LeeTZ/Yo
a0153b5	mengqwang	bug fix for type
b47cb37	mengqwang	bug fix for primary expr

9a7aca5	mengqwang	bug fix for statement
c9152c7	Edward Cheng	start semantic and test
af512fd	Edward Cheng	merged with master
c66c330	Edward Cheng	refactor the type resolution; add new_var
11cbf33	mengqwang	string_of_expr
8367138	mengqwang	string_of_expr
4e1a09c	Edward Cheng	ready for merge
0081bd9	Edward Cheng	merged with update in ast
5684ff1	Edward Cheng	removed primary_expr
8400a39	Edward Cheng	revised parser and sat
e60529a	Yufei Ou	add output.ml
bdfc715	mengqwang	read types in first pass
237076e	mengqwang	read types in the first pass
6c0c2aa	Edward Cheng	create sast
04a8223	Edward Cheng	add stmt_semantic
9916c1b	Yufei Ou	add output again
a333ed1	mengqwang	type reader
b5bd02c	mengqwang	type read
f57d4d7	Edward Cheng	fix bugs in semantic
6638630	Edward Cheng	merged
a8a2776	Edward Cheng	update test semantic
fb61362	LeeTZ	add some parser_test
afb636a	Yufei Ou	add generate.ml
67250df	Yufei Ou	add generate.ml
495dc15	Yufei Ou	Merge branch 'yo'
f3aa5c6	mengqwang	2nd pass
1f3addb	mengqwang	2nd pass
a8edf00	mengqwang	2nd pass type
973f1e1	mengqwang	readme
9fa3934	mengqwang	readme
209ec5e	mengqwang	readme
a9fb3b5	mengqwang	readme
dc65172	mengqwang	readme
444a937	LeeTZ	continue some parser test
8aadbac	LeeTZ	remove output for Bro Fei
ad7b2fe	LeeTZ	modified parser and ast print func
115b4af	LeeTZ	lots of debug on ast and parser
14168a7	LeeTZ	modify sast
6cd909e	Yufei Ou	update generate.ml
7e6eed8	LeeTZ	finish ast and parser syntax debug
a121478	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
97b4afd	LeeTZ	let s change it to semi
e626854	mengqwang	typereader exception
3667ad0	LeeTZ	finish testing of parser!
0b67d95	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
263d0aa	mengqwang	bug plz help

0fe8d40	Yufei Ou	tested generate.ml
035dafc	Yufei Ou	tested generate.ml
8c6ed6f	Edward Cheng	update semantic
2a2e2de	mengqwang	wth and?
b576846	LeeTZ	finish parser TEST sh, lets make more examples
101d173	LeeTZ	update sast and semantic from the south
2d097a0	Edward Cheng	fix return type checking
95aaec0	Edward Cheng	fix semantic
c70cec4	Edward Cheng	before merge with maste
3365731	LeeTZ	modify ast string and add more tests
27e9124	LeeTZ	modify gitignore
922a5fe	LeeTZ	add testcases
1535376	LeeTZ	finish ast
9fc10b2	LeeTZ	renew all testcases
1a197f8	mengqwang	type read
a794748	Yufei Ou	Merge branch 'master' of https://github.com/LeeTZ/Yo into yo
5d91c61	mengqwang	type read
720ee20	mengqwang	exist type
03af9d9	Edward Cheng	update sast
d209a3d	LeeTZ	add sast from mc
ba64e05	LeeTZ	finish sast stringlize part
71e4144	mengqwang	type read
c372523	mengqwang	type read
d3fdde4	LeeTZ	fix typo
408bb98	LeeTZ	t pushMerge branch 'master' of github.com:LeeTZ/Yo
e3a53c1	LeeTZ	add mengqing_test
995055e	mengqwang	test
4c61879	Yufei Ou	Merge branch 'master' of https://github.com/LeeTZ/Yo into yo
4b55a97	Edward Cheng	merge with master
131f123	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
ee7a104	Yufei Ou	Merge branch 'master' of https://github.com/LeeTZ/Yo into yo
5392220	LeeTZ	finish lots of debugs
943e772	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
6f78a5e	Yufei Ou	generate main
178ce88	Yufei Ou	Merge branch 'yo'
17d05eb	LeeTZ	we made a great progess
d598168	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
402d7ad	LeeTZ	fix makefile
bbb6bd6	Yufei Ou	change something
9b7785d	LeeTZ	make a generate test but NOTHING HAPPEND we donot know why
a99a030	Yufei Ou	change something
0ea45af	Yufei Ou	Merge branch 'master' of https://github.com/LeeTZ/Yo
36b9a13	LeeTZ	feige is so lihai
97ada76	Yufei Ou	add header thing
608c6b5	Yufei Ou	fix string output
dbcdcf7	Edward Cheng	fix ifstmt

0495aa1	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
a4e283c	Edward Cheng	add more types to log
29a4410	Yufei Ou	change op
8c4c16d	Yufei Ou	Merge branch 'master' of https://github.com/LeeTZ/Yo
c842a40	Yufei Ou	change while
2b71a1c	LeeTZ	we successfully build helloworld
303c790	Yufei Ou	add yolib.h
9ce0969	Yufei Ou	Merge branch 'master' of https://github.com/LeeTZ/Yo
e37d2d3	Yufei Ou	add yolib.h
8e6c105	LeeTZ	add fullstach test!
a654c83	LeeTZ	modify gitignore and clean it
d1d97da	LeeTZ	add test
ab0f777	LeeTZ	add preprocess test and parser test
20957f9	LeeTZ	renew test shell
330c84e	LeeTZ	clean
42eca40	mengqwang	bug fix
c8b06b1	mengqwang	typereader bug fix
1661d76	mengqwang	bug fix typereader
7c9a08a	mengqwang	makefile bug
651596b	LeeTZ	add util
36f7e3d	Edward Cheng	fix makefile dependency
4921ffd	Yufei Ou	add HAT and AT
bcf1564	Edward Cheng	add new array
c06056e	Edward Cheng	add array literal type checking
de7293b	Edward Cheng	eval_entry ret not None; op rename; Binop semantic analysis
f68818d	Edward Cheng	add ForIn semantic
7017316	Edward Cheng	modify ForEq to ForRange
04c8670	Edward Cheng	add return type to function signature
8bb142c	Edward Cheng	add ArrayIndex semantic analysis; remove Noexpr
9f93c42	LeeTZ	add openshot example
8e7196c	Edward Cheng	modify array type in ast, sast, built_in_type
3534d0e	mengqwang	Merge branch 'typename' of https://github.com/LeeTZ/Yo
6100e6c	LeeTZ	add definition of clip
048573f	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
eb97e81	Edward Cheng	make array type recursive
8e4527c	Edward Cheng	separate array_constructor
6500865	Edward Cheng	update all type related semantic analysis
a541409	Edward Cheng	Merge with branch typename
94c390c	LeeTZ	clean
135b51a	Yufei Ou	forange
c420360	Mengqing Wang	array resolve
f293271	Mengqing Wang	delete mengqingtest
fa57a0f	Mengqing Wang	type test
e25c91d	Yufei Ou	move tabs to sast
f46e3ac	Yufei Ou	Merge branch 'master' of https://github.com/LeeTZ/Yo
07be179	Yufei Ou	add test executable to gitignore

87a5d42	Yufei Ou	work from another computer
f0a020d	Edward Cheng	reverse entry move
42dba33	LeeTZ	add clip, frame libs
6403ac4	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
c83cc26	Mengqing Wang	type test
7cbf0d7	Edward Cheng	generate almost finished
1a309b5	Mengqing Wang	array resolve
97bf2e7	Mengqing Wang	print type
3f1a817	Mengqing Wang	print type
3ebe52e	Mengqing Wang	my new typerheader
3b4fa59	Edward Cheng	codegen finished
ddfb362	Edward Cheng	merged
cb1b115	LeeTZ	fix type_entry comparsion
48458f1	LeeTZ	merged
b52c64f	LeeTZ	big move
fd97386	Edward Cheng	add array operation to generate
15ee52e	LeeTZ	finish dream function of clipindex and cliprange
e36260e	LeeTZ	add testplan
7b8a5cf	LeeTZ	clean scanner test
eedbe69	LeeTZ	clean it
67e0954	LeeTZ	renew fullstack testsuits
8f51587	LeeTZ	renew absolutely
76f8d95	LeeTZ	add logclip for testing
9ddd66c	LeeTZ	remove main function in libclip, it is part of lib now
924e11c	Edward Cheng	add set attribute
5a89b35	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
e002085	Mengqing Wang	type redefine exception
0b49a83	Mengqing Wang	type redefinition exception
592dc28	mengqwang2	typereader test
8b56156	mengqwang2	test sh
e646b37	Edward Cheng	add static to function generation
ade4f3d	Edward Cheng	merged
f7a0f78	mengqwang2	typereader bug fix
0708e34	mengqwang2	bug fix tr
d0129b3	mengqwang2	bug fix tr
577d6cc	LeeTZ	fix lib and generate
29bbe90	Edward Cheng	modify youlib
0b470eb	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
a64c21d	LeeTZ	munan test suits
917a503	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
86da28d	Edward Cheng	memory bug fix
c5f108a	LeeTZ	compiled with libopenshot!
185fe69	mengqwang2	typereader eval
1da7975	Edward Cheng	add forward; declaration; fix naming convention
fbb7c03	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
451a257	mengqwang2	t_actual

61dcc37	Edward Cheng	fix type name convention
f5ff013	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
867109d	Edward Cheng	fix name convention in generate
8b23fc3	Edward Cheng	Clip creation works
41452aa	mengqwang2	parser test
d4ef220	mengqwang2	tr test
26aad5e	LeeTZ	modify test.sh to log error in file
5a36285	Edward Cheng	use base_type instead of string for function name in SCall
4b75484	Edward Cheng	merged test.sh
2af16a3	Edward Cheng	add clip read/write
4a7ecdd	Edward Cheng	use auto
6a203bb	LeeTZ	add a new bash to run prog
027ef88	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
1173392	Edward Cheng	fix video attribute editing
8bd217b	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
b5dde72	LeeTZ	add new bin
48d8b4d	LeeTZ	add a new disto
10b714a	Edward Cheng	add wrapper to clip
3159f37	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
50d65ae	mengqwang2	test
5199b46	LeeTZ	debug yolib and finish rename!
c7e977b	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
0535068	LeeTZ	finish a fadeout effect
e6f34e0	LeeTZ	add logclip
75c4e73	Edward Cheng	add log to clip
bfd82e0	LeeTZ	debug cliprange
e173f7b	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
cdc3fc5	Edward Cheng	add void to basic type in codegen; add clip func test
2f86ee2	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
ad86ccf	LeeTZ	debug for alpha value
55f6c14	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
8dd138e	Edward Cheng	fix another compare_type bug in semantic
6015904	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
f927810	LeeTZ	debugging layerclip
811ff8d	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
3da332a	LeeTZ	clean output
2fe8cc1	Edward Cheng	rename ClipConcat to ClipCascade; add ClipConcat
1f8e52f	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
6370556	LeeTZ	add move in x array
c6d907e	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
3c2ab6c	Edward Cheng	add video concat test
2a0646e	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
b05cfbf	Edward Cheng	change Void type name
e3c4807	LeeTZ	a very guiyi de error
ad7e227	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
51d8543	mengqwang2	seman test

f88f83b	Edward Cheng	add eval init
f94c0b1	Edward Cheng	fix pointer
db1d54d	LeeTZ	hack for pixel
f5b3c27	LeeTZ	finish all properties to set
26a7f36	Edward Cheng	add pixel
c64e9af	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
ddbdfad	LeeTZ	modify test cases
3061643	Edward Cheng	change return type to pixel
7a89483	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
2c02fa6	Edward Cheng	fix log name in yolib.h
9554d48	LeeTZ	hack
d3d6cc6	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo
c7d0a03	LeeTZ	debugging pixels
859b68c	LeeTZ	set yo.sh again, delete intermediate fileso
1d3eeee	LeeTZ	add new test cases
bf266f7	LeeTZ	clean
a451cdc	LeeTZ	add keyframe tests
57dc58f	Edward Cheng	add array ops
840ab27	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
5caf6e0	Edward Cheng	fix array add return type in cpp
2b24f17	LeeTZ	fix typename
d167cb3	Edward Cheng	add Clip[](DIR_STRING)
591a1ea	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
0065d48	Edward Cheng	fix function eval
e53a852	LeeTZ	add mengqing test
e2c5260	LeeTZ	fix array add
9fd6236	Edward Cheng	fix function eval signature
06db98d	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
57d7dc8	LeeTZ	gcd test
c4ae6b5	LeeTZ	what
b76d780	Edward Cheng	fix array indexing
4fd7a5f	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
72f9e9f	Edward Cheng	add webm to gitignore
117733c	Edward Cheng	exclude array length in forward decl
df1b187	LeeTZ	a
85aeeef	LeeTZ	add test cases
fca20e3	Edward Cheng	add array ops
882a5d5	Edward Cheng	Merge branch 'master' of https://github.com/LeeTZ/Yo
c6df624	Edward Cheng	correct array func name
3402a0b	LeeTZ	help munan debugging with array
e78878e	LeeTZ	array finish
173a4ac	Edward Cheng	add log bool
424c747	Edward Cheng	merge
1874208	Edward Cheng	add array literal back
d9885e2	LeeTZ	finish all tests
515ce71	LeeTZ	Merge branch 'master' of github.com:LeeTZ/Yo

dfc7ec0	LeeTZ	Merge branch 'master' into demo
f90845c	LeeTZ	debug and add demo
95f7a22	LeeTZ	add more demo
3dad3d2	LeeTZ	add logclip
fc59950	LeeTZ	clean bin/
f6c6e89	LeeTZ	clean src
c4acbfd	LeeTZ	finish clean except test

8.1 preprocessor.py

```
#! /usr/bin/python
import os
import re
import sys
try:
  from cStringIO import StringIO
except:
  from StringIO import StringIO

def process(input_file):
  invalidchar = ('\t','{','}')
  blockcomment = ['#(',')#']

  stack = [0]
  output = StringIO()
  newindent = False
  commented = False
  linejoin  = False

  for i, line in enumerate(input_file):
    lineout = remove_inline(line)

    if lineout:
      for x in invalidchar:
        if x in lineout:
          error("SyntaxError: Invalid character {} found on line {}".format(x,i))

      # Check if first statement is a block comment
      lstripline = lineout.lstrip()

      if len(lstripline) > 1 and blockcomment[0] == lstripline[:2]:
        commented = True

      # Checks if line gets uncommented
      if commented:
          if len(lineout) > 1 and blockcomment[1] == lineout[-2:]:
            commented = False
      else:

        if not linejoin:
          wcount  = len(lineout) - len(lineout.lstrip(' '))

          # If the previous line began an indentation,
```

```
          # add the new indentation level to the block
          # (so long as the new indentation
          # level is greater than the previous one)
          if newindent == True:
            if wcount > stack[-1]:
              stack.append(wcount)
              newindent = False
            else:
              error("IndentationError on line {}".format(i))

          # If the indentation level is greater than expected,
          # throw an error
          if wcount > stack[-1]:
            error("IndentationError on line {}".format(i))

          else:

            # If the indentation level is less than the current level,
            # return to a previous indentation block.
            #Throw an error if you return to an indentation
            # level that doesn't exist
            while(wcount < stack[-1]):
              lineout = "}\n" + lineout
              stack.pop()

            if wcount != stack[-1]:
              error("IndentationError on line {}".format(i))

        # Given that the indentation level is correct,
        # check for the start of a new code block
        # (where a line ends with a ':')
        # and insert a '{'. At the end of a line, add a semicolon ';'
        # unless if there is a linejoin character '\'.
        if lineout[-1] == ':':
          lineout = lineout + '{\n'
          newindent = True

        elif lineout[-1] == '\\':
          linejoin = True
          lineout = lineout[:-1]

        else:
          lineout = lineout + ';\n'
          linejoin = False

        output.write(lineout)

  while 0 < stack[-1]:
    output.write("}\n")
    stack.pop()

  return output

def error(msg):
```

```
  sys.stderr.write(msg+"\n")
  sys.exit(2)

def remove_inline(line):
  if "##" in line:
    regex = re.compile("^(.*?)#.*|.*")
    m = regex.match(line)
    comments_removed = m.group(1)
  else:
    comments_removed = line
  return comments_removed.rstrip()

def usage():
  print"""
  python preprocessor.py [input.yo]
  """

if __name__ == "__main__":

  if len(sys.argv) != 2:
    usage()
    sys.exit(2)

  try:
    f_in = open(sys.argv[1],"r")
  except IOError:
    error("IOError: Cannot read input file %s.\n" % sys.argv[1])

  name_ext = os.path.basename(f_in.name)
  dir_ext = os.path.dirname(f_in.name)+"/"

  if name_ext.lower().endswith(".yo"):
    fname = os.path.splitext(name_ext)[0]
  else:
    error('NameError: Input must have yo file extension')

  out_str = process(f_in)

  f_out = open(dir_ext+ "intermediate/" + fname+".yo", 'w')
  f_out.write(out_str.getvalue())
```

8.2 scanner.mll

```
{ open Parser }

let Integer_cons = ['0'-'9']+
let Double_cons = ['0'-'9']+ '.' ['0'-'9']+
let Id_cons = ['a'-'z' 'A'-'Z' '_']['a'-'z' 'A'-'Z' '0'-'9' '_']*
let String_cons = [^ '"' ]*

rule token = parse
  [' ' '\r' '\t' '\n'] { token lexbuf } (* Whitespace *)
| "#("     { comment lexbuf }           (* Comments *)
```

```
| '#'       { oneLineComment lexbuf}
| '('       { LPAREN }
| ')'       { RPAREN }
| '['       { LBRACKET }
| ']'       { RBRACKET }
| '{'       { LBRACE }
| '}'       { RBRACE }
| ';'       { SEMI }
| ','       { COMMA }
| '.'       { DOT }
| '~'       { TILDE }
| '"'       { QUOTATION }
| ':'       { COLON }
| ';'       { SEMI }
| '+'       { PLUS }
| '-'       { MINUS }
| '*'       { TIMES }
| '/'       { DIVIDE }
| '='       { ASSIGN }
| "=="      { EQ }
| "!="      { NEQ }
| '<'       { LT }
| "<="      { LEQ }
| '>'       { GT }
| ">="      { GEQ }
| '%'       { MOD }
| "&&"      { AND }
| '&'       { AMPERSAND }
| "||"      { OR }
| '!'       { EXCLAMATION }
| "->"      { RIGHTARROW }
| "^"       { HAT }
| "@"        { AT }

| "if"      { IF }
| "else"    { ELSE }
| "elif"    { ELIF }
| "in"      { IN }
| "for"     { FOR }
| "while"   { WHILE }
| "return"  { RETURN }
| "continue" { CONTINUE }
| "break"   { BREAK }
| "to"      {TO}
| "downto"   {DOWNTO}
| "func"    { FUNCTION }
| "type"    { TYPE }
| "true"     as lxm  { BoolLITERAL(bool_of_string lxm) }
| "false"    as lxm  { BoolLITERAL(bool_of_string lxm) }

| Integer_cons as lxm { IntLITERAL(int_of_string lxm) }
| Double_cons as lxm { DoubleLITERAL(float_of_string lxm) }
| '"' String_cons '"' as lxm { StringLITERAL(lxm) }
| Id_cons as lxm { ID(lxm) }
```

```
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment = parse
  "#)" { token lexbuf }
| _    { comment lexbuf }

and oneLineComment = parse
  '\n' { token lexbuf }
| _    { oneLineComment lexbuf }

(*and continue = parse
  [' ' '\t' '\r' '\n'] {continue lexbuf}
| [^ ' ' '\t' '\r' '\n'] {token lexbuf}*)
```

8.3 parser.mly

```
%{
    open Ast
%}

%token SEMI LPAREN RPAREN LBRACKET RBRACKET LBRACE RBRACE
%token COMMA DOT TILDE QUOTATION COLON
%token PLUS MINUS TIMES DIVIDE ASSIGN MOD AND OR AMPERSAND EXCLAMATION
%token EQ NEQ LT LEQ GT GEQ
%token RETURN IF ELSE ELIF FOR WHILE IN TO DOWNTO CONTINUE BREAK
%token FUNCTION TYPE /*EVAL*/
%token RIGHTARROW /*LEFTARROW*/ HAT AT
%token TRUE FALSE
%token <int> IntLITERAL
%token <float> DoubleLITERAL
%token <string> StringLITERAL
%token <bool> BoolLITERAL
%token <string> ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%nonassoc COMMA
%left RPAREN
%right LPAREN
%left OR
%left AND
%left LT GT LEQ GEQ
%left EQ NEQ
%left HAT AT
%left AMPERSAND
%left PLUS MINUS
%left TIMES DIVIDE MOD
%left DOT
%left  RBRACKET
%right  LBRACKET
%nonassoc EXCLAMATION
```

```
%start global
%type <Ast.program> global

%%
literal:
    | IntLITERAL                                  { IntConst $1 }
    | DoubleLITERAL                               { DoubleConst $1 }
    | StringLITERAL                               { StrConst $1 }
    | BoolLITERAL                                 { BoolConst $1 }
    | array_literal                               { $1  }

arg_expr_opt:
    /* nothing */                                  { [] }
    | arg_expr_list                                { List.rev $1 }

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

array_literal:
    LBRACKET arg_expr_list RBRACKET             { ArrayConst (List.rev $2) }

type_base:
    | ID                                          { SimpleType $1 }
    | type_base DOT ID                          { NestedType($1, $3) }

type_name:
    type_base                                   { $1 }
    | type_name LBRACKET RBRACKET               { ArrayType $1 }

expr:
    | ID                                        { Var $1}
    | literal                                   {$1}
    | LPAREN expr RPAREN                        {$2}
    | expr PLUS   expr                          { Binop($1, Add,   $3) }
    | expr MINUS  expr                          { Binop($1, Sub,   $3) }
    | expr TIMES  expr                          { Binop($1, Mult,  $3) }
    | expr DIVIDE expr                          { Binop($1, Div,   $3) }
    | expr MOD      expr                        { Binop($1, Mod,   $3) }
    | expr EQ     expr                          { Binop($1, Eq,    $3) }
    | expr NEQ    expr                          { Binop($1, Neq,   $3) }
    | expr LT     expr                          { Binop($1, Less,  $3) }
    | expr LEQ    expr                          { Binop($1, Leq,   $3) }
    | expr GT     expr                          { Binop($1, Gt,    $3) }
    | expr GEQ    expr                          { Binop($1, Geq,   $3) }
    | expr AND      expr                        { Binop($1, And,   $3) }
    | expr OR           expr                    { Binop($1, Or,   $3) }
    | dot_expr                                { $1 }
    | expr LBRACKET expr RBRACKET                 { ArrayIndex($1, $3) }
    | expr LBRACKET expr COLON expr RBRACKET { ArrayRange($1, $3, $5) }
    | expr HAT expr AT expr                       { ClipCascade($1, $3, $5) }
    | expr coord AT expr                      { ClipPixel($1, $2, $4) }
    | expr AMPERSAND expr                         { ClipConcat($1, $3)}
```

```
    | expr DOT ID LPAREN arg_expr_opt RPAREN { Call(Some($1), $3, $5) }
    | ID LPAREN arg_expr_opt RPAREN         { Call(None, $1, $3) }
    | array_constructor LPAREN arg_expr_opt RPAREN { BuildArray($1, $3) }

dot_expr:
    | expr DOT ID  { DotExpr($1, $3) }

coord:
    | LT expr EXCLAMATION expr GT { Coord($2, $4)}

array_constructor:
    | expr LBRACKET RBRACKET { SimpleArrayConstructor $1 }
    | array_constructor LBRACKET RBRACKET { CompositeArrayConstructor $1}

expr_opt:
    /* nothing */ { None }
    | expr          { Some($1) }

statement:
    | expr SEMI            { Assign(None, $1) }
    | expr ASSIGN expr SEMI              { Assign(Some($1), $3) }
    | dot_expr AT expr ASSIGN expr SEMI      { SetAttribute($1, $3, $5) }
    | IF expr COLON LBRACE statement_opt
    RBRACE elif_statement_list else_statement
        { IfStmt(List.rev ($8 @ $7 @ [ CondExec(Some($2), $5) ])) }
    | WHILE expr COLON LBRACE statement_opt RBRACE  { WhileStmt($2, $5) }
    | FOR ID IN for_in_expr COLON LBRACE  statement_opt RBRACE
        { ForIn($2, $4, $7) }
    | FOR ID ASSIGN expr TO expr COLON LBRACE statement_opt RBRACE
        { ForRange($2, $4, $6, $9, Inc) }
    | FOR ID ASSIGN expr DOWNTO expr COLON LBRACE statement_opt RBRACE
        { ForRange($2, $4, $6, $9, Dec) }
    | CONTINUE SEMI                { Continue }
    | BREAK SEMI                   { Break }
    | RETURN expr_opt SEMI         { Return $2 }

for_in_expr:
    ID            {Var $1}
    | array_literal {$1}

statement_opt:
    /* nothing */ { [] }
    | statement_list { List.rev $1 }

statement_list:
    | statement { [ $1 ] }
    | statement_list statement { $2 :: $1 }

elif_statement_list:
    /* nothing */ { [] }
    | ELIF expr COLON LBRACE statement_opt RBRACE elif_statement_list
        { $7 @ [ CondExec(Some($2), $5) ] }

else_statement:
```

```
    /* nothing */ { [] }
    | ELSE COLON LBRACE statement_opt RBRACE { [CondExec(None, $4)] }

var_decl:
    ID COLON type_name  { VarDecl($1, $3) }

mem_var_decl:
    var_decl SEMI  { MemVarDecl($1) }

func_decl:
    FUNCTION ID LPAREN func_arg_opt RPAREN RIGHTARROW type_name
    COLON LBRACE statement_opt RBRACE
    {FuncDecl($2, $4, $7, $10)}

mem_func_decl:
    func_decl { MemFuncDecl($1) }

global_func_decl:
    func_decl { GlobalFunc($1) }

func_arg_opt:
    /* nothing */        { [] }
    | func_arg_list         { List.rev $1 }

func_arg_list:
    | var_decl                                              { [$1] }
    | func_arg_list COMMA var_decl    { $3 :: $1 }

type_decl:
    TYPE ID COLON LBRACE type_element_list RBRACE {TypeDecl($2, $5)}

mem_type_decl:
    type_decl  {MemTypeDecl($1)}

global_type_decl:
    type_decl { GlobalType($1) }

global_statement:
    statement  { GlobalStmt($1) }

type_element_list:
    /*  nothing */  { [] }
    | mem_var_decl  type_element_list { $1 :: $2 }
    | mem_func_decl type_element_list { $1 :: $2 }
    | mem_type_decl type_element_list { $1 :: $2 }

global_ele:
    | global_func_decl  { $1 }
    | global_type_decl  { $1 }
    | global_statement  { $1 }

global_ele_list:
    | global_ele                        { [$1] }
    | global_ele_list global_ele    { $2 :: $1 }
```

```
global:
    global_ele_list { List.rev $1 }
```

8.4 ast.ml

```
type op = Add | Sub | Mult | Div | Mod | Eq
    | Neq | Less | Leq | Gt | Geq | And | Or

type types = Int | Double | Bool | String  of types

type type_name =
  | SimpleType of string
  | NestedType of type_name * string
  | ArrayType of type_name

type expr =                                          (* Expressions*)
    IntConst of int                                  (* 35 *)
  | DoubleConst of float                             (* 21.4 *)
  | BoolConst of bool                                (* True *)
  | StrConst of string                               (* "ocaml" *)
  | ArrayConst of expr list                          (* [12,23,34,56] *)
  | ArrayIndex of expr * expr                        (* A[B[3]]*)
  | Var of string                                    (* foo *)
  | DotExpr of expr * string                         (* A.B *)
  | Call of expr option * string * expr list         (* foo(a, b) *)
  | Binop of expr * op * expr
  | ArrayRange of expr * expr * expr
  | ClipCascade of expr * expr * expr
  | ClipPixel of expr * coord * expr
  | ClipConcat of expr * expr
  | BuildArray of array_constructor * expr list
and array_constructor =
  | SimpleArrayConstructor of expr
  | CompositeArrayConstructor of array_constructor
and coord =
  | Coord of expr * expr

type stmt =
  | Assign of expr option * expr
  | SetAttribute of expr * expr * expr
  | IfStmt of cond_exec list
  | ForIn of string * expr * stmt list
  | ForRange of string * expr * expr * stmt list * for_range_dir
  | WhileStmt of expr * stmt list
  | Continue
  | Break
  | Return of expr option

and for_range_dir = Inc | Dec

and cond_exec =
   CondExec of expr option * stmt list
```

```
and var_decl =
  | VarDecl of string * type_name

and func_decl =
  | FuncDecl of string *
  var_decl list * type_name * stmt list

and type_decl =
  | TypeDecl of string * mem_type_decl list

and mem_type_decl =
  | MemVarDecl of var_decl
  | MemFuncDecl of func_decl
  | MemTypeDecl of type_decl

and global_ele_decl =
  | GlobalStmt of stmt
  | GlobalFunc of func_decl
  | GlobalType of type_decl

type program = global_ele_decl list

let rec string_of_type_name = function
  | SimpleType t -> t
  | NestedType (p, t) ->
  (string_of_type_name p) ^ "." ^ t
  | ArrayType t -> (string_of_type_name t) ^ "[]"

let string_of_op = function
  | Add -> "+" | Sub -> "-" | Mult -> "*" | Div -> "/" | Mod -> "%"
  | Eq -> "==" | Neq -> "!=" | Less -> "<" | Leq -> "<=" | Gt -> ">"
  | Geq -> ">=" | And -> "&&" | Or -> "||"



let rec string_of_expr = function
  | IntConst l -> string_of_int l
  | DoubleConst d -> string_of_float d
  | BoolConst b -> string_of_bool b
  | StrConst s -> s
  | Var v -> v
  | ArrayIndex(a, b) -> (string_of_expr a) ^
  "[" ^ (string_of_expr b) ^ "]"
  | ArrayConst(e) -> let s =
  (List.fold_left (fun a b -> a ^ ", " ^ b)
  "" (List.map string_of_expr e)) in
  "[" ^ (String.sub s 2 ((String.length s) - 2))  ^ "]"
  | DotExpr(a, b) -> (string_of_expr a) ^ "." ^ b
  | Binop(e1, o, e2) -> (string_of_expr e1) ^ " "
  ^ (string_of_op o) ^ " " ^ (string_of_expr e2)
  | Call(obj, f, el) -> (match obj with
          | None -> "" | Some s -> (string_of_expr s) ^ "." )^
          f ^ "(" ^ (String.concat ", "
```

```
          (List.map string_of_expr el)) ^ ")"
  | ArrayRange (cl, st, ed) -> (string_of_expr cl) ^
  "[" ^ (string_of_expr st) ^ ":" ^ (string_of_expr ed) ^ "]"
  | ClipCascade (cl1, cl2, tm) -> (string_of_expr cl1) ^
  "^" ^ (string_of_expr cl2) ^ "@" ^ (string_of_expr tm)
  | ClipConcat (cl1, cl2) ->
  (string_of_expr cl1) ^ "&" ^ (string_of_expr cl2)
  | ClipPixel (cl, c, tm) ->
  (string_of_expr cl) ^ (string_of_coord c) ^ (string_of_expr tm)
  | BuildArray (t, el) -> (string_of_array_constructor t) ^
  "(" ^ (String.concat ", " (List.map string_of_expr el)) ^ ")"

and string_of_coord = function
  | Coord (x, y) ->
  "<" ^ (string_of_expr x) ^ ", " ^ (string_of_expr y) ^ ">"

and string_of_array_constructor = function
  | SimpleArrayConstructor e ->
  (string_of_expr e) ^ "[]"
  | CompositeArrayConstructor e ->
  (string_of_array_constructor e) ^ "[]"

and string_of_stmt = function
  | Assign(None,rvalue) -> string_of_expr rvalue
  | Assign(Some(lvalue), rvalue) ->
  (string_of_expr lvalue) ^ " = " ^ (string_of_expr rvalue)
  | SetAttribute(main, time, value) ->
  (string_of_expr main) ^ "@"
  ^ (string_of_expr time) ^ "=" ^ (string_of_expr value)
  | IfStmt(conds) ->
  string_of_first_cond_exec (List.hd conds) ^ "\n" ^
  (String.concat "\n"
  (List.map string_of_cond_exec (List.tl conds)))
  | ForIn(var, expr, stmts) ->
  "for " ^ var ^ " in " ^ (string_of_expr expr)
    ^ ":\n" ^ (String.concat "\n" (List.map string_of_stmt stmts))
  | ForRange(var, exprst, expred, stmts, dir) ->
  "for " ^ var ^ " = " ^ (string_of_expr exprst)
    ^ (match dir with | Inc -> " to " | Dec -> " downto ")
    ^ (string_of_expr expred) ^ ":\n"
    ^(String.concat "\n" (List.map string_of_stmt stmts))
  | WhileStmt(expr, stmts) ->
  "while " ^ (string_of_expr expr) ^ ":\n"
    ^ (String.concat "\n" (List.map string_of_stmt stmts))
  | Continue -> "continue"
  | Break -> "break"
  | Return(None) -> "return"
  | Return(Some(expr)) -> "return " ^ (string_of_expr expr)

and string_of_first_cond_exec = function
  | CondExec(None, stmts) -> "else:" ^
  (String.concat "\n" (List.map string_of_stmt stmts))
  | CondExec(Some(expr), stmts) ->
  "if " ^ (string_of_expr expr) ^ ":\n" ^
```

```
  (String.concat "\n" (List.map string_of_stmt stmts))

and string_of_cond_exec = function
  | CondExec(None, stmts) -> "else:" ^
  (String.concat "\n" (List.map string_of_stmt stmts))
  | CondExec(Some(expr), stmts) -> "elif "
  ^ (string_of_expr expr) ^ ":\n"
  ^ (String.concat "\n" (List.map string_of_stmt stmts))

and string_of_var_decl = function
  | VarDecl(id, ty) -> (string_of_type_name ty) ^ " " ^ id

and string_of_func_decl = function
  | FuncDecl(name, args, retype, stmts) ->
  "func " ^ name ^ " (" ^
  (String.concat ", " (List.map string_of_var_decl args))
    ^ ")->" ^ (string_of_type_name retype)
    ^ "\n" ^ (String.concat "\n" (List.map string_of_stmt stmts))

and string_of_type_decl = function
  | TypeDecl(name, args) -> "type " ^ name ^
  "\n" ^ (String.concat "\n"
  (List.map string_of_type_mem_decl args))

and string_of_type_mem_decl = function
  | MemVarDecl o -> string_of_var_decl o
  | MemFuncDecl o -> string_of_func_decl o
  | MemTypeDecl o -> string_of_type_decl o

and string_of_global_ele_decl = function
  | GlobalStmt o -> string_of_stmt o
  | GlobalFunc o -> string_of_func_decl o
  | GlobalType o -> string_of_type_decl o

and string_of_program program =
  String.concat "\n" (List.map string_of_global_ele_decl program)

  (*List.iter
  fun global_ele -> match global_ele with
  | FuncDecl
  | TypeDecl
  | Stmt
  program*)


exception VariableNotDefined of string
exception TypeNotDefined of string
exception SemanticError of string
exception ProcessingError of string
exception TypeExist of string
exception GenerationError of string
exception TypeRedefined of string

module NameMap = Map.Make(String)
```

```
type eval_entry = {
    mutable args: var_entry list;
    mutable ret: type_entry
    }
and base_type =  {
  t_name: string;
  (* type name used in yo *)
  t_actual: string;
  (* actual name used in target language *)
  mutable evals: eval_entry list;
  (* a list of eval functions *)
  mutable members: type_entry NameMap.t
  (* map of member_name => type_entry *)
  }
and type_entry =  BaseTypeEntry of base_type
| ArrayTypeEntry of type_entry
and var_entry = {
  v_name: string;
  (* type name used in yo *)
  v_actual: string;
  (* actual name used in target language *)
  v_type: type_entry
  (* type definition *)
  }

let rec compare_type t1 t2 = match t1, t2 with
  | BaseTypeEntry _, ArrayTypeEntry _
  | ArrayTypeEntry _,  BaseTypeEntry _
  -> false
  | ArrayTypeEntry a, ArrayTypeEntry b ->
  compare_type a b
  | BaseTypeEntry a, BaseTypeEntry b ->
  a.t_name = b.t_name && a.t_actual = b.t_actual

(* compile environment:
variable symbol table * type environment table *)
type compile_context = {
  mutable vsymtab: var_entry NameMap.t list;
  (* a stack of variable symbol maps of varname => var_entry *)
  mutable typetab: base_type NameMap.t
  (* type environment table: a map of base type name => base_type *)
}
(*
let base_type ctx type_name =
BaseTypeEntry(look_up_type type_name ctx.typetab)
*)
let binop_type_tab = function
  | Add   -> "__Add"
  | Sub   -> "__Sub"
  | Mult  -> "__Mult"
  | Div   -> "__Div"
  | Mod   -> "__Mod"
```

```
  | Eq    -> "__Equal"
  | Neq   -> "__Neq"
  | Less  -> "__Less"
  | Leq   -> "__Leq"
  | Gt    -> "__Gt"
  | Geq   -> "__Geq"
  | And   -> "__And"
  | Or    -> "__Or"
```

8.5 type_reader.ml

```
open Ast

let walk_dec program context =

    let generate_scope parent_scope id =
    (if parent_scope="" then "" else (parent_scope ^ "_")) ^ id
    in

    let generate_var_type id tail =
    if tail="" then id else (id^"_")^ tail
    in

    let rec wrapArray curtype arrcnt =
        if arrcnt = 0 then curtype
        else (wrapArray (ArrayTypeEntry(curtype)) (arrcnt-1))
    in

    let duplicate_types typetab id =
        try
            let _ = NameMap.find id typetab in
                raise (TypeRedefined id)
        with
            Not_found -> id
    in

    let exists_types_id typetab basename arrcnt =
            try
                let be = (NameMap.find basename typetab) in
                let curtype =
                wrapArray (BaseTypeEntry(be)) arrcnt in
                curtype
            with Not_found -> raise (TypeNotDefined basename)
    in

    let rec exists_types typetab tail arrcnt = function
        | SimpleType(st) ->
                let basename = generate_var_type st tail in
                    exists_types_id typetab basename arrcnt
        | NestedType(nt,id) ->
                let basename = generate_var_type id tail in
                    exists_types typetab basename arrcnt nt
        | ArrayType(at) ->
```

```
            exists_types typetab tail (arrcnt+1) at
in

let rec type_nested_walk_1 typetab oid = function
    | Ast.TypeDecl(id, type_element) ->
            let newid = generate_scope oid id in
            let _ = duplicate_types typetab newid in
            let typeEntry = {t_name=newid; t_actual=
            "_"^newid; evals=[]; members=NameMap.empty;} in
            let tt = NameMap.add newid typeEntry typetab in
                List.fold_left
                (fun tt e ->
                mem_nested_1 tt newid e) tt type_element

and func_nested_walk_1 typetab oid = function
    | Ast.FuncDecl(id, arglist, retype, stmtlist) ->
            if id<>"eval" then (
            let newid = generate_scope oid id in
            let typeEntry = {t_name=newid; t_actual=
            "_"^newid; evals=[]; members=NameMap.empty;} in
            let tt = NameMap.add newid typeEntry typetab in
            tt) else typetab


and mem_nested_1 typetab id = function
    | Ast.MemTypeDecl(typedecl) ->
       type_nested_walk_1 typetab id typedecl
    | Ast.MemFuncDecl(funcdecl) ->
       func_nested_walk_1 typetab id funcdecl
    | _ -> typetab
in

let funcwalk_1 typetab parent_scope = function
    | Ast.FuncDecl(id, arglist, retype, stmtlist) ->
            let newid = generate_scope parent_scope id in
            let entry = {t_name=newid; t_actual="_"^
            newid; evals=[]; members=NameMap.empty} in
            let tt = NameMap.add newid entry typetab in
            tt

and typewalk_1 typetab parent_scope= function
     | Ast.TypeDecl(id, type_element) ->
            let newid = generate_scope parent_scope id in
            let _ = duplicate_types typetab newid in
            let entry = {t_name=newid; t_actual="_"^newid;
            members=NameMap.empty; evals=[]} in
            let tt = NameMap.add newid entry typetab in
            List.fold_left (fun tt e ->
            mem_nested_1 tt id e) tt type_element
    in

let walk_decl_1 typetab = function
      | Ast.GlobalType(type_decl) ->
      let tt = typewalk_1 typetab "" type_decl in tt
```

```
      | Ast.GlobalFunc(func_decl) ->
      let tt = funcwalk_1 typetab "" func_decl in tt
      | _ -> typetab
in


let varwalk_2 typetab parent = function
    | Ast.VarDecl(name, typename) ->
        let memVarEntry =
        exists_types typetab "" 0 typename in
        let parent_base = NameMap.find parent typetab in
        parent_base.members <-
        NameMap.add name memVarEntry parent_base.members
in

let funcwalk_2 typetab parent_scope = function
    | Ast.FuncDecl(id, arglist, retype, stmtlist) ->
        let scope = if id <> "eval" then
        (generate_scope parent_scope id)
        else parent_scope in
        let f_type = NameMap.find scope typetab in
        f_type.evals <-
        {args=(List.map (fun x -> match x with
            VarDecl(n,t) -> {v_name=n; v_actual="_"^n;
            v_type=(exists_types typetab "" 0 t)}) arglist);
            ret=(exists_types typetab "" 0 retype)}
            :: f_type.evals
in

let rec typewalk_2 typetab parent_scope = function
    | Ast.TypeDecl(id, ele_list) ->
        let newid = generate_scope parent_scope id in
        List.iter (fun e -> match e with
            | Ast.MemFuncDecl(mf) ->
            funcwalk_2 typetab newid mf; ()
            | Ast.MemVarDecl(mv) ->
            varwalk_2 typetab newid mv; ()
            | Ast.MemTypeDecl(mt) ->
            typewalk_2 typetab newid mt
        ) ele_list;
in

let rec walk_decl_2 typetab = function
    | Ast.GlobalType(type_decl) ->
    typewalk_2 typetab "" type_decl; typetab
    | Ast.GlobalFunc(func_decl) ->
    funcwalk_2 typetab "" func_decl; typetab
    | _ -> typetab
in

let add_typeeval k bt =
    if (List.length bt.evals)=0 then bt.evals <-
        {args=[]; ret=BaseTypeEntry(bt)} :: bt.evals
in
```

```
    let first_pass tt program =
        let listpass = List.fold_left (fun tt e ->
        walk_decl_1 tt e) tt program
            in listpass
    and second_pass tt program =
        let listpass = List.fold_left (fun tt e ->
        walk_decl_2 tt e) tt program
            in listpass
    and third_pass tt =
        let listpass =
        (NameMap.iter
        (fun k v -> add_typeeval k v ) tt;tt) in listpass
in let t = context.typetab in let t =
first_pass t program in let t =
second_pass t program in let t = third_pass t in
{vsymtab=[NameMap.empty]; typetab=t}
```

8.6 semantic.ml

```
open Ast
open Sast

let rec look_up_var id =
function
    | [] ->
 raise (VariableNotDefined id)
    | hd :: tail ->
 (try NameMap.find id hd with Not_found ->
 look_up_var id tail)

(* lookup a variable from local to global in vsymtab.
Usage: look_up_var id context.vsymtab *)
let rec look_up_type typeName typenv =
(try NameMap.find typeName typenv
  with Not_found ->
 raise (TypeNotDefined typeName))

(* lookup a variable from local to global in vsymtab.
Usage: look_up_var id context.vsymtab *)
let rec look_up_type2 typeName typenv =
  try
  (let rec generate_simple_type_name =
function
    | SimpleType s ->
 s
    | NestedType(m, s) ->
 (generate_simple_type_name m) ^
 "_" ^
 s
    | _ ->
 raise (SemanticError "Nested type should be SimpleType") in
  let rec generate_type =
```

```
function
    | ArrayType t ->
 ArrayTypeEntry(generate_type t)
    | x ->
 BaseTypeEntry(NameMap.find (generate_simple_type_name x) typenv) in
  generate_type typeName)
  with Not_found ->
 raise (TypeNotDefined (string_of_type_name typeName))


(* create a new variable for the given name and
type in the topmost variable map in vsymtab *)
let new_var ctx varName typeDef =
  try let _ =
NameMap.find varName (List.hd ctx.vsymtab) in
    raise (SemanticError (varName ^
 " already defined in current scope"))
  with Not_found ->
 ctx.vsymtab <- (NameMap.add varName
    {v_name=varName;
 v_actual=varName ^
 "_";
 v_type=typeDef}
    (List.hd ctx.vsymtab)) :: (List.tl ctx.vsymtab);

 SVar (varName, {actions=[NewVar];
 type_def=typeDef})

let push_var_env ctx =
{ctx with vsymtab =
NameMap.empty :: ctx.vsymtab}

let rec string_of_arg =
function | BaseTypeEntry e ->
 e.t_name | ArrayTypeEntry e ->
 "Array<" ^
 (string_of_arg e) ^
 ">"

let find_matching_eval func_type call_arg_types =
  let rec compare_args args1 args2 =
match args1, args2 with
    | [], [] ->
 true | _, [] | [], _ ->
 false |  x::xs, y::ys ->
 (compare_type x y) && (compare_args xs ys) in
  List.find (fun e ->
  compare_args (List.map (fun k ->
 k.v_type) e.args) call_arg_types) func_type.evals

type replace_item =
{src_type: type_entry;
 tgt_type: type_entry}
```

```
let find_matching_template_eval func_type call_arg_types replace_list=
  let rec compare_args args1 args2 =
match args1, args2 with
    | [], [] ->
 true | _, [] | [], _ ->
 false
    | x::xs, y::ys ->

      (compare_type (try (List.find (fun t ->
 compare_type t.src_type x) replace_list).tgt_type
       with Not_found ->
 x) y) && (compare_args xs ys) in
  List.find (fun e ->
  compare_args (List.map (fun k ->
 k.v_type) e.args) call_arg_types) func_type.evals

(* build semantic for an expression;
 extract_semantic can be usd to get the semantic*)
let rec build_expr_semantic ctx (expression:expr) : s_expr=

  let int_type =
BaseTypeEntry(look_up_type "Int" ctx.typetab)
  and double_type =
BaseTypeEntry(look_up_type "Double" ctx.typetab)
  and string_type =
BaseTypeEntry (look_up_type "String" ctx.typetab)in
  let bool_type =
BaseTypeEntry(look_up_type "Bool" ctx.typetab)
  and clip_type =
BaseTypeEntry(look_up_type "Clip" ctx.typetab)
  and pixel_type =
BaseTypeEntry(look_up_type "Pixel" ctx.typetab)
  and void_type =
BaseTypeEntry(look_up_type "Void" ctx.typetab)
  and frame_type =
BaseTypeEntry(look_up_type "Frame" ctx.typetab) in
  match expression with
  (* Int, Double, Bool, Str are consolidated into
  SLiteral since there aren't much*)
  (* difference in code generation - just
  print the string representation! *)
    | IntConst x ->
 SLiteral(string_of_int x, {actions=[];
 type_def=int_type})
  | DoubleConst x ->
 SLiteral(string_of_float x, {actions=[];
 type_def=double_type})
  | BoolConst x ->
 SLiteral(string_of_bool x, {actions=[];
 type_def=bool_type})
  | StrConst x ->
 SLiteral(x, {actions=[];
 type_def=string_type})
```

```
    | ArrayConst x ->

      let elementType =
match x with (* determine the array type by its first element *)
      | [] ->
 raise (SemanticError ("Array literal length has to be at least 1"))
      | hd::tl ->
 (extract_semantic (build_expr_semantic ctx hd)).type_def in
    let arraySem =
{actions=[NewArr];
 type_def=ArrayTypeEntry(elementType)} in
    SArrayLiteral (
      (List.map (fun e ->
 let s =
build_expr_semantic ctx e in
      if compare_type (extract_semantic s).type_def elementType
 then s
      else raise (SemanticError
      ("Element types in the array has to be uniform"))) x), arraySem)
    (* try to find the variable in the symbol table;
 may throw exception when it is not found *)
    | Var x ->
 SVar (x, {actions=[];
 type_def=(look_up_var x ctx.vsymtab).v_type})

  | ArrayIndex (main, idx) ->

    let smain =
build_expr_semantic ctx main and sidx =
build_expr_semantic ctx idx in
    (let main_type =
(extract_semantic smain).type_def in
      match main_type with
      | BaseTypeEntry t ->
 (
       if compare_type main_type clip_type
 then
          (if compare_type
          (extract_semantic sidx).type_def double_type
          then SClipTimeIndex(smain, sidx, {actions=[];
 type_def=frame_type})
          else if compare_type
          (extract_semantic sidx).type_def int_type
          then SClipFrameIndex(smain, sidx, {actions=[];
 type_def=frame_type})
          else raise (SemanticError ((string_of_expr idx) ^
 " has to be of either Double or Int type")))
       else raise (SemanticError ((string_of_expr main) ^
 " has to be of Array type")))
      | ArrayTypeEntry ele_type ->
 (
       if compare_type (extract_semantic sidx).type_def int_type
       then SArrayIndex(smain, sidx, {actions=[];
 type_def=ele_type})
```

```
        else raise (SemanticError ((string_of_expr idx) ^
 " has to be of Int type"))))

  | ArrayRange (main, st, ed) ->
    let smain =
build_expr_semantic ctx main in
    let sst =
build_expr_semantic ctx st and sed =
build_expr_semantic ctx ed in
    let type_main =
(extract_semantic smain).type_def in
    (match type_main with
      | BaseTypeEntry t ->
 (
        if compare_type type_main clip_type
 then
          (if compare_type
          (extract_semantic sst).type_def double_type
            && compare_type
            (extract_semantic sed).type_def double_type
          then SClipTimeRange(smain, sst, sed, {actions=[];
 type_def=clip_type})
          else if compare_type
          (extract_semantic sst).type_def int_type
            && compare_type (extract_semantic sed).type_def int_type
          then SClipFrameRange(smain, sst, sed, {actions=[];
 type_def=clip_type})
          else raise (SemanticError ((string_of_expr st) ^
 " and " ^
 (string_of_expr ed)
            ^
 " have to be of both Double types or Int types")))
        else raise (SemanticError ((string_of_expr main) ^
 " has to be of Array type")))
      | ArrayTypeEntry _ ->
 (
        if compare_type (extract_semantic sst).type_def int_type
          && compare_type (extract_semantic sed).type_def int_type
        then SArrayRange(smain, sst, sed, {actions=[];
 type_def=type_main})
        else raise (SemanticError ((string_of_expr st) ^
 " and " ^
 (string_of_expr ed) ^
 " have to be of Int type"))))

    | DotExpr (expr, x) ->

      let sexpr =
build_expr_semantic ctx expr in
      SDotExpr (sexpr, x,  (try {actions=[];
 type_def=NameMap.find x (
                    match (extract_semantic sexpr).type_def with
                      | BaseTypeEntry t ->
 t.members
```

```
                      | ArrayTypeEntry t ->
 (look_up_type "Array" ctx.typetab).members)}
                with Not_found ->
 raise (VariableNotDefined (x ^
 " in " ^
 (string_of_expr expr)))))

    | Binop (x, op, y) ->

      let b1 =
build_expr_semantic ctx x and b2 =
build_expr_semantic ctx y in
    let s1 =
extract_semantic b1 and s2 =
extract_semantic b2 in
    let func_eval =
try find_matching_eval
(look_up_type (binop_type_tab op) ctx.typetab) [s1.type_def;
 s2.type_def]
           with Not_found ->
 raise (SemanticError ("Operator " ^
 (string_of_op op)
              ^
 " does not take params of type " ^
 (string_of_type s1.type_def)
              ^
 " and " ^
 (string_of_type s2.type_def)))
   in SBinop (b1, op, b2, {actions=[];
 type_def=func_eval.ret})


    | Call (obj, fname, args) ->

      let s_call_args =
List.map (fun e ->
 build_expr_semantic ctx e) args in (*build semantics for args*)
      let type_obj =
match obj with
        | None ->
 void_type
        | Some x ->
 (extract_semantic (build_expr_semantic ctx x)).type_def in
      let array_type =
BaseTypeEntry(look_up_type "Array" ctx.typetab) in
    let func_name =
      if compare_type void_type type_obj
 then fname
      else (match type_obj with
        | ArrayTypeEntry t ->
 (string_of_type array_type)
        | _ ->
(string_of_type type_obj)) ^
 "_" ^
```

```
 fname in
    let func_type =
try NameMap.find func_name ctx.typetab (* get type_entry for this func *)
            with Not_found ->
 raise (TypeNotDefined ("Function " ^
 fname ^
 " is not defined")) in
    let augmented_s_args =
match obj with
      | None ->
 s_call_args
      | Some x ->
 (build_expr_semantic ctx x) :: s_call_args in
    let call_arg_types =
      List.map (fun e ->
 (extract_semantic e).type_def) augmented_s_args in
    let func_eval_ret =
      try match type_obj with
        | ArrayTypeEntry t ->
 (let element_type_t =
BaseTypeEntry(look_up_type "ArrayElementT" ctx.typetab) in
          let type_replace_list =
            [{src_type=element_type_t;
 tgt_type=t};
            {src_type=array_type;
 tgt_type=type_obj}] in
          let matching_eval =
find_matching_template_eval
func_type call_arg_types type_replace_list in
            try (List.find (fun p ->
 compare_type p.src_type matching_eval.ret)
 type_replace_list).tgt_type
            with Not_found ->
 matching_eval.ret)
        | _ ->
 (find_matching_eval func_type call_arg_types).ret
 (* get the matching eval *)
      with Not_found ->
 raise (TypeNotDefined ("Function " ^
 fname ^
 " does not take params of type ("
            ^
 (String.concat ", " (List.map string_of_expr args)) ^
 ")")) in
    (
      match type_obj with
      | ArrayTypeEntry t ->

        SArrayOperation ((match obj with
            | None ->
 raise (ProcessingError("Expecting obj in ArrayOperation"))
            | Some s ->
 build_expr_semantic ctx s),
          fname, s_call_args,
```

```
          {actions=[];
 type_def=func_eval_ret})
      | _ ->
 SCall ((match obj with
            | None ->
 None
            | Some s ->
 Some(build_expr_semantic ctx s)),
          func_type, s_call_args,
          {actions=[];
 type_def=func_eval_ret}))

  | ClipCascade (cl1, cl2, tm) ->
    let scl1 =
build_expr_semantic ctx cl1 and scl2 =
build_expr_semantic ctx cl2 and stm =
build_expr_semantic ctx tm in
    if compare_type (extract_semantic scl1).type_def clip_type
      && compare_type (extract_semantic scl2).type_def clip_type
      then (if compare_type
      (extract_semantic stm).type_def double_type
        then SClipTimeCascade(scl1, scl2, stm, {actions=[];
 type_def=clip_type})
        else if compare_type (extract_semantic stm).type_def int_type
        then SClipFrameCascade(scl1, scl2, stm, {actions=[];
 type_def=clip_type})
        else
        raise (SemanticError
"ClipCascade operation expects last argument of type Int or Double"))
    else raise (SemanticError
"First two arguments of ClipCascade operation expects type of Clip, Clip")

  | ClipConcat (cl1, cl2) ->
    let scl1 =
build_expr_semantic ctx cl1 and scl2 =
build_expr_semantic ctx cl2 in
    if compare_type (extract_semantic scl1).type_def clip_type
      && compare_type (extract_semantic scl2).type_def clip_type
    then SClipConcat(scl1, scl2, {actions=[];
 type_def=clip_type})
    else raise (SemanticError
    "ClipConcat operation expects type of Clip, Clip")

  | ClipPixel (clip, coord, tm) ->
    let scl =
build_expr_semantic ctx clip and stm =
build_expr_semantic ctx tm in
    if compare_type (extract_semantic scl).type_def clip_type
    && ((compare_type (extract_semantic stm).type_def int_type)
      || (compare_type (extract_semantic stm).type_def double_type))
    then
    (match coord with Coord (x, y) ->

      let sx =
```

```
build_expr_semantic ctx x
      and sy  =
build_expr_semantic ctx y in
      if compare_type (extract_semantic sx).type_def int_type
      && compare_type (extract_semantic sy).type_def int_type
      then SClipPixel (scl, sx, sy, stm, {actions=[];
 type_def=pixel_type})
      else raise
      (SemanticError
        "Coordinate in ClipPixel operation expects type of Int, Int"))
  else raise (SemanticError
    "ClipPixel operation expects type of Clip, <Int, Int>, Int")

  | BuildArray (main, args) ->
    let rec resolve_ele_type =
function
      | SimpleArrayConstructor expr ->
 BaseTypeEntry(look_up_type (string_of_expr expr) ctx.typetab)
      | CompositeArrayConstructor arr ->
 ArrayTypeEntry(resolve_ele_type arr) in
    let ele_type =
resolve_ele_type main in
    let clip_type =
BaseTypeEntry(look_up_type "Clip" ctx.typetab) in
    if compare_type ele_type clip_type
    then
      (if (List.length args)=1
 then
        (let sarg1 =
build_expr_semantic ctx (List.hd args) in
        if compare_type (extract_semantic sarg1).type_def string_type
 then SBuildClipArray(sarg1, {actions=[NewArr];
 type_def=ArrayTypeEntry(ele_type)})
        else raise
        (SemanticError
          "Clip array constructor expects directory name of String type"))
      else raise (SemanticError
        "Clip array constructor expects exactly one argument"))
    else SBuildArray (ele_type, [], {actions=[NewArr];
 type_def=ArrayTypeEntry(ele_type)})
let rec build_stmt_semantic ctx =
function
  | Assign (e1, e2) ->

    (let r_expr_sem =
(build_expr_semantic ctx e2) in
    match e1 with
      (* When there is no l-value, set
      the semantic of the stmt to that of r-value *)
      | None ->
 SAssign (None, r_expr_sem)
      | Some e ->

        try let l_expr_sem =
```

```
build_expr_semantic ctx e in
        (* When the l-value is defined,
        check its type against that of r-value *)
          let ltype =
(extract_semantic l_expr_sem).type_def
          and rtype =
(extract_semantic r_expr_sem).type_def in
          if  compare_type ltype rtype
            then SAssign (Some(l_expr_sem), r_expr_sem)
            else raise (SemanticError
             ("Invalid assignment: expecting " ^
 (string_of_type ltype)
              ^
 ", but having " ^
 (string_of_type rtype) ))
        (* create a new variable with the type of r-value *)
        with VariableNotDefined s ->
 match e with
        | Var x ->
 let l_expr_sem =
new_var ctx x (extract_semantic r_expr_sem).type_def in
          SAssign (Some(l_expr_sem), r_expr_sem)
        (* make sure l-value is a var:
        otherwise it must be defined (processed above) *)
        | _ ->
 raise (SemanticError ("Invalid assignment: lvalue " ^
 (string_of_expr e) ^
 " not found")))
  | SetAttribute (main, time, value) ->
    let s_main =
(match main with
      | DotExpr (expr, x) ->
 build_expr_semantic ctx main
      | _ ->
 raise (SemanticError ((string_of_expr main) ^
 "is expected to have DotExpr here")))
    and s_time =
build_expr_semantic ctx time
    and s_value =
build_expr_semantic ctx value in
    (match s_main with SDotExpr (sexpr, x, sem) ->
      let int_type =
BaseTypeEntry(look_up_type "Int" ctx.typetab) and
        double_type =
BaseTypeEntry(look_up_type "Double" ctx.typetab) and
        clip_type =
BaseTypeEntry(look_up_type "Clip" ctx.typetab) in
      if compare_type (extract_semantic sexpr).type_def clip_type
 then (
        if compare_type
        (extract_semantic s_value).type_def double_type
 then (
          if compare_type (extract_semantic s_time).type_def int_type
 then SFrameSetAttribute (sexpr, x, s_time, s_value)
```

```
          else if
          compare_type (extract_semantic s_time).type_def double_type
 then STimeSetAttribute (sexpr, x, s_time, s_value)
          else raise (SemanticError ("Attribute assignment index" ^
 (string_of_expr time) ^
 " has to be of Int of Double type"))
       ) else raise (SemanticError ((string_of_expr value) ^
 ": rvalue for attribute assignment has to be of Double type"))
     ) else raise
 (SemanticError
 ("Attribute assignment has to be performed to a Clip, but " ^
 (string_of_expr main) ^
 "'s type is mismatched"))
     | _ ->
 raise (ProcessingError
 ("SetAttribute analysis error when processing " ^
 (string_of_s_expr s_main))))
  | IfStmt cl ->

    let bool_type =
BaseTypeEntry(look_up_type "Bool" ctx.typetab) in
    let ctx2 =
push_var_env ctx in
    SIfStmt (List.map (* go through each cond_exec *)
    (fun t ->
 match t with CondExec (x,y) ->
 let s_stmt_list =
List.map (build_stmt_semantic ctx2) y in
      match x with
      | None ->
 SCondExec(None,  s_stmt_list) (* the final else has no predicate *)
      | Some cond ->

        let expr_sem =
build_expr_semantic ctx2 cond in
(* make sure the predicate has type Bool *)
        if compare_type (extract_semantic expr_sem).type_def bool_type
 then SCondExec(Some(expr_sem), s_stmt_list)
        else raise (SemanticError ("Condition expression" ^
 (string_of_expr cond)
          ^
 " in the if statement should be of Bool type")) )
    cl)

  | WhileStmt (pred, stmts) ->

    let bool_type =
BaseTypeEntry(look_up_type "Bool" ctx.typetab) in
    let ctx2 =
push_var_env ctx in
    let s_pred =
build_expr_semantic ctx2 pred in
    if compare_type (extract_semantic s_pred).type_def bool_type
 then ()
```

```
    else raise (SemanticError ("Condition expression" ^
 (string_of_expr pred)
      ^
 " in the while statement should be of Bool type"));
    SWhileStmt (s_pred, List.map (build_stmt_semantic ctx2) stmts)

  | Continue ->
 SContinue
  | Break ->
 SBreak
  | Return exp ->
 (match exp with
    | None ->
 SReturn (None)
    | Some x ->
 SReturn (Some(build_expr_semantic ctx x)))

  | ForIn (varname, expr, stmts) ->

    let s_expr =
build_expr_semantic ctx expr in
    let arr_ele_type =
match (extract_semantic s_expr).type_def with
              | ArrayTypeEntry t ->
 t
              | _ ->
 raise (SemanticError ((string_of_expr expr)
                ^
" in the for loop has to be of Array type")) in
    let nested_env =
push_var_env ctx in
    let svar =
new_var nested_env varname arr_ele_type in
    let s_stmt_list =
List.map (build_stmt_semantic nested_env) stmts in
    SForIn(varname, (extract_semantic svar), s_expr, s_stmt_list)

  | ForRange (varname, st_expr, ed_expr, stmts, dir) ->
    let int_type =
BaseTypeEntry(look_up_type "Int" ctx.typetab) in
    let s_st_expr =
build_expr_semantic ctx st_expr and s_ed_expr =
build_expr_semantic ctx ed_expr in
    let nested_env =
push_var_env ctx in
    let svar =
new_var nested_env varname
      (if compare_type (extract_semantic s_st_expr).type_def int_type
      then let ed_type=(extract_semantic s_ed_expr).type_def in
      (if compare_type ed_type int_type
 then ed_type
      else raise (SemanticError ((string_of_expr ed_expr) ^
 " is expected to be of type Int")))
      else raise (SemanticError ((string_of_expr st_expr) ^
```

```
 " is expected to be of type Int"))) in
    let s_stmt_list =
List.map (build_stmt_semantic nested_env) stmts in
    SForRange(varname,
      (extract_semantic svar), s_st_expr, s_ed_expr, s_stmt_list, dir)


let build_func_semantic ctx =
function
  FuncDecl (funcName, argList, retype, stmtList) ->
    let ctx2 =
push_var_env ctx in (* create a new variable env on top of the old *)
    let sarglist =
List.map (*  *)
        (fun x ->
 match x with VarDecl (name, typename) ->
          let svar =
new_var ctx2 name (look_up_type2 typename ctx.typetab) in
          SVarDecl (name, extract_semantic svar)) argList in
    let s_stmtlist =
List.map (build_stmt_semantic ctx2) stmtList in
    let expected_ret =
look_up_type2 retype ctx.typetab in
    let check_ret_type =
function
      | SReturn expr_option ->

        let actual_ret_type =
(match expr_option with
          | Some ep ->
 (extract_semantic ep).type_def | None ->
 BaseTypeEntry(look_up_type "Void" ctx.typetab)) in
        if compare_type actual_ret_type expected_ret
 then ()
        else raise (SemanticError ("Return expressions must have type " ^
 (string_of_type expected_ret)
          ^
 " as defined in function " ^
 funcName))
      | _ ->
 () in
    List.iter (fun x ->
 match x with
      | SReturn r ->
 check_ret_type (SReturn r)
      | SIfStmt ceList ->
 List.iter check_ret_type
              (List.flatten (List.map (fun ce ->
 match ce with SCondExec (_, stList) ->
 stList) ceList))
      | SForIn (_, _, _, stList) ->
 List.iter check_ret_type stList
      | SForRange (_, _, _, _, stList, _) ->
 List.iter check_ret_type stList
```

```
      | _ ->
 ()
    ) s_stmtlist;

    SFuncDecl (funcName, sarglist, s_stmtlist, {actions=[];
 type_def=expected_ret})

let rec build_type_mem_semantic ctx =
function
  | MemFuncDecl memfunc ->
 SMemFuncDecl (build_func_semantic ctx memfunc)
  | MemTypeDecl memtype ->
 SMemTypeDecl (build_type_semantic ctx memtype)
  | MemVarDecl  memvar  ->

    match memvar with
    | VarDecl (varname, vartype) ->
 SMemVarDecl (SVarDecl (varname,
        {actions=[];
 type_def=look_up_type2 vartype ctx.typetab}))

and build_type_semantic ctx =
function
  | TypeDecl (typename, memlist) ->
 STypeDecl (typename, List.map (build_type_mem_semantic ctx) memlist)

let build_program_semantic ctx =
function
  | GlobalStmt stmt ->
 SGlobalStmt (build_stmt_semantic ctx stmt)
  | GlobalType type_decl ->
 SGlobalType (build_type_semantic ctx type_decl)
  | GlobalFunc func_decl ->
 SGlobalFunc (build_func_semantic ctx func_decl)


let build_semantic context program =
List.map (build_program_semantic context) program
```

8.7 semantic_test.ml

```
open Ast
open Sast
let yoheader = "../src/yoheader.yo"

let _ =
    let lexbuf = Lexing.from_channel (open_in yoheader) in
    let program = Parser.global Scanner.token lexbuf in
    let builtincontext = Type_reader.walk_dec program
    {vsymtab=[NameMap.empty]; typetab=NameMap.empty} in
    let lexbuf = Lexing.from_channel stdin in
    let program = Parser.global Scanner.token lexbuf in
    let context = Type_reader.walk_dec program builtincontext in
```

```
    let seman = Semantic.build_semantic context program in
    print_string (string_of_s_program seman)
```

8.8 generate.ml

```
open Ast
open Sast

let is_primitive_type t_def = match t_def.t_name with
    | "Int" | "Double" | "String" | "Bool" | "Void" -> true
    | _ -> false

let rec generate_type_modifier = function
    | BaseTypeEntry b ->
    (match b.t_name with
        | "Int" -> "int" | "Double" -> "double"
        | "Bool" -> "bool" | "String" -> "string" | "Void" -> "void"
        | _ -> "tr1::shared_ptr<" ^ b.t_actual ^ ">")
    | ArrayTypeEntry a ->
    "tr1::shared_ptr<std::vector<" ^
    (generate_type_modifier a) ^ ">>"


let extract_array_ele_type t =
match t with ArrayTypeEntry a -> a | _ ->
raise (GenerationError "Expect an array type here")

let rec generate_expr = function
    | SLiteral (x, s) -> x
    | SArrayLiteral (x, s)-> "create_array<"
    ^ (generate_type_modifier (extract_array_ele_type s.type_def)) ^
        ">({" ^ String.concat ", " (List.map generate_expr x) ^ "})"
    | SVar (x, s) -> x
    | SArrayIndex (x, y, s) ->
    "(*" ^ generate_expr x ^ ")[" ^ generate_expr y ^ "]"
    | SDotExpr (x, y, s) ->
    generate_expr x ^ "->" ^ y
    | SBinop (x, op, y, s) ->
    "(" ^ (generate_expr x) ^ " " ^
    (string_of_op op) ^ " " ^ (generate_expr y) ^ ")"
    | SCall (obj, func_type, el, s) ->
    func_type.t_actual ^ "::eval(" ^
        (match obj with | None ->
        "DUMMY_SELF" | Some (expr) -> (generate_expr expr)) ^
        (List.fold_left (fun content x ->
         content ^ ", " ^ (generate_expr x)) "" el) ^ ")"
    | SBuildArray (ele_type, _, _) ->
    "create_array<" ^ (generate_type_modifier ele_type) ^ ">()"
    | SBuildClipArray (d, _) ->
    "createClips(" ^ (generate_expr d) ^ ")"
    | SArrayRange (smain, sst, sed, sem) ->
    "slice_array<" ^ (generate_type_modifier
    (extract_array_ele_type sem.type_def))
```

```
        ^ ">(" ^ (generate_expr smain) ^
        ", " ^ (generate_expr sst) ^ ", " ^
        (generate_expr sed) ^ ")"
    | SClipTimeCascade (scl1, scl2, stm, _) ->
    "layerClip(" ^ (generate_expr scl1) ^ ", " ^
  (generate_expr scl2) ^ ", "
        ^ (generate_expr stm) ^ ")"
    | SClipFrameCascade (scl1, scl2, stm, _) ->
    "layerClip(" ^ (generate_expr scl1) ^ ", " ^
  (generate_expr scl2) ^ ", "
          ^ (generate_expr stm) ^ ")"
    | SClipConcat (scl1, scl2, _) -> "addClip(" ^
    (generate_expr scl1) ^ ", " ^ (generate_expr scl2) ^ ")"
    | SClipFrameRange (smain, sst, sed, _) ->
    "clipRange(" ^ (generate_expr smain) ^  ", " ^
    (generate_expr sst) ^ ", " ^ (generate_expr sed) ^ ")"
    | SClipTimeRange (smain, sst, sed, _) ->
    "clipRange(" ^ (generate_expr smain) ^  ", " ^
    (generate_expr sst) ^ ", " ^ (generate_expr sed) ^ ")"
    | SClipFrameIndex (smain, sidx, _) ->
    "clipIndex(" ^ (generate_expr smain) ^  ", "
     ^ (generate_expr sidx) ^ ")"
    | SClipTimeIndex (smain, sidx, _) ->
    "clipIndex(" ^ (generate_expr smain) ^  ", "
    ^ (generate_expr sidx) ^ ")"
    | SClipPixel (clip, x, y, tm, _) ->
    "getPixel(" ^ (generate_expr clip) ^  ", "
    ^ (generate_expr tm) ^  ", "
        ^ (generate_expr x) ^  ", " ^ (generate_expr y) ^ ")"
    | SArrayOperation (smain, opname, el, s) ->
    "_Array_" ^ opname ^
        (match (extract_semantic smain).type_def with
            | ArrayTypeEntry t -> "<" ^
            (generate_type_modifier t) ^ ">"
            | _ ->
            raise (ProcessingError("Expecting ArrayTypeEntry here"))) ^
        "::eval(" ^ (String.concat ", " (List.map (fun x ->
        generate_expr x) (smain::el))) ^ ")"


let rec generate_cond = function
    | SCondExec (x, l) ->
        let rec generate_stmt_list = function
          [] -> ""
        | hd::tl -> generate_stmt hd ^ generate_stmt_list tl
        in
        (match x with
        | None -> "else if (true)
        {\n" ^ generate_stmt_list l ^ "}\n"
        | Some (expr) -> "else if ("
        ^ generate_expr expr ^ ") {\n" ^ generate_stmt_list l ^ "}\n"
        )

and generate_stmt = function
```

```
| SAssign (x, s) ->
    (match x with
    | None -> generate_expr s
    | Some (expr) ->
        let sem = extract_semantic expr in
        (try let _ = List.find (fun x ->
        match x with NewVar -> true
        | _ -> false) sem.actions in
            (*(generate_type_modifier sem.type_def)
            ^ *)"auto " ^ (generate_expr expr)
        with Not_found -> generate_expr expr)
        ^ " = " ^ (generate_expr s)
        (*let sem = extract_semantic expr in
        (try List.find (fun x ->
        match x with NewVar -> true | _ -> false) sem.actions;
            (match sem.type_def.t_name with
            | "Int" -> "int " ^
            generate_expr expr ^ " = " ^ generate_expr s
            | "Double" -> "double" ^
            generate_expr expr ^ " = " ^ generate_expr s
            | "String" -> "string" ^
            generate_expr expr ^ " = " ^ generate_expr s
            | "Bool" -> "bool" ^
            generate_expr expr ^ " = " ^ generate_expr s
            | _ -> "auto " ^ (generate_expr expr)
            ^ " = make_shared<" ^
                (let tm = (generate_type_modifier sem.type_def) in
                    String.sub tm 16 ((String.length tm)-16)) ^ ">()"
            )
        with Not_found ->
        (generate_expr expr) ^ " = " ^ (generate_expr s))*)
    )
    ^ ";\n"

| SFrameSetAttribute (sexpr, x, s_time, s_value) ->
    "setProperty(" ^
    (String.concat ", " [(generate_expr sexpr); "\""
    ^ x ^ "\""; (generate_expr s_time);
(generate_expr s_value)]) ^ ");\n"
| STimeSetAttribute (sexpr, x, s_time, s_value) ->
    "setProperty(" ^ (String.concat ", "
    [(generate_expr sexpr); "\"" ^ x ^ "\"";
    (generate_expr s_time); (generate_expr s_value)]) ^ ");\n"

| SIfStmt (l) -> "if(false) {}\n" ^
    (let rec generate_cond_list = function
      [] -> ""
    | hd::tl -> (generate_cond hd)
    ^ (generate_cond_list tl)
    in generate_cond_list l)

| SForIn (loop_var_name, loop_var_sem, array_expr, stmt_list) ->
    "for (auto " ^ loop_var_name ^
    " : *(" ^ (generate_expr array_expr) ^ ")) {\n" ^
```

```
        (generate_stmt_list  stmt_list) ^ "}\n\n"

    | SForRange (loop_var_name, sem, st, ed, stmt_list,sign) ->
        let startstr = generate_expr st in
        let endstr = generate_expr ed in
        "for (auto " ^ loop_var_name ^ "=" ^
        startstr ^ ";" ^ loop_var_name ^
        (if sign = Inc
            then "<" ^ endstr ^ ";" ^ loop_var_name ^ "++"
            else ">" ^ startstr ^ ";" ^ loop_var_name ^ "--") ^ ") {\n" ^
        (generate_stmt_list stmt_list)
        ^ "}\n\n"


    | SWhileStmt (x, l) -> "while (" ^
    generate_expr x ^ ")" ^ "{\n" ^
        (generate_stmt_list l) ^ "}\n\n"

    | SContinue -> "continue;\n"
    | SBreak -> "break;\n"
    | SReturn (x) -> "return " ^
        (match x with
        | None -> ""
        | Some (expr) ->
        generate_expr expr) ^ ";\n"

and generate_stmt_list stmt_list =
String.concat "\n" (List.map generate_stmt stmt_list)

let generate_var_decl = function
    | SVarDecl (name, s) ->
    (generate_type_modifier s.type_def) ^ " " ^ name

let generate_init_eval tname args stmts s =
    let initialObject =
        let extract_arg_name =
        function SVarDecl(arg_name, arg_sem) -> arg_name in
        match args with | [] ->
        raise (ProcessingError
        "Expecting a self argument in the eval function") | hd :: tail ->
            let selfName =
            extract_arg_name hd and typeName =
            (generate_type_modifier s.type_def) in
            selfName ^ " = " ^ typeName ^ "( new " ^ tname ^ "());\n"
        in
        "static " ^ (generate_type_modifier s.type_def) ^ " eval(" ^
        (String.concat ", " (List.map (
            fun x ->
             (match x with SVarDecl(arg_name, arg_sem) ->
             (generate_type_modifier arg_sem.type_def)
             ^ " " ^ arg_name )) args) )
        ^ ") {\n" ^ initialObject ^ (generate_stmt_list stmts) ^ "}\n\n"

let generate_eval args stmts s =
```

```
        "static " ^ (generate_type_modifier s.type_def) ^
        " eval(tr1::shared_ptr<Universal>" ^
            (if (List.length args)>0 then ", "
            else "") ^ (String.concat ", " (List.map (
                fun x ->
                (match x with SVarDecl(arg_name, arg_sem) ->
                (generate_type_modifier arg_sem.type_def)
                ^ " " ^ arg_name )) args) )
        ^ ") {\n" ^ (generate_stmt_list stmts) ^ "}\n\n"



let generate_func parent_name = function
    | SFuncDecl(name, args, stmts, s) ->
    "struct " ^ parent_name ^ "_" ^ name ^ " {\n" ^
        (generate_eval args stmts s) ^ "\n};\n\n"

let rec generate_type parent_name = function
    | STypeDecl (s, stml) ->
    let this_name = parent_name ^ "_" ^ s in
    "struct " ^ this_name ^ " {\n" ^

    (let generate_member content = function
        | SMemVarDecl v -> content ^
        (generate_var_decl v) ^ ";\n" | _ ->
        raise (GenerationError "Expecting SMemVarDecl here")
     in List.fold_left generate_member ""
     (List.filter (fun x ->
     match x with SMemVarDecl s -> true | _ -> false) stml)) ^

    (let generate_type_eval content = function
        | SMemFuncDecl f ->
         content ^
            (match f with SFuncDecl(name, args, stmts, s) ->
            if name <> "eval" then ""
            else (generate_init_eval this_name args stmts s))
        | _ -> raise (GenerationError
        "Expecting SMemVarDecl here")
     in List.fold_left generate_type_eval ""
     (List.filter (fun x -> match x with SMemFuncDecl s ->
     true | _ -> false) stml)) ^
    "\n};\n\n" ^

    (let generate_mem_func content = function
        | SMemFuncDecl f -> content ^
            (match f with SFuncDecl(name, args, stmts, s) ->
            if name="eval" then "" else (generate_func this_name f))
        | _ -> raise (GenerationError "Expecting SMemVarDecl here")
     in List.fold_left generate_mem_func ""
     (List.filter (fun x -> match x with SMemFuncDecl s -> true
     | _ -> false) stml)) ^

    (List.fold_left  (fun content x -> content ^
        (match x with SMemTypeDecl mtd ->
```

```
        generate_type this_name mtd | _ ->
        raise (ProcessingError ("Generation error"))) ) ""
        (List.filter (fun x -> match x with SMemTypeDecl s ->
         true | _ -> false) stml))

let generate context program =
    let header = ["\"../src/yolib.h\""] in
    let pre_defined = List.map (fun h ->"#include " ^ h ^ "\n")
    header in
    String.concat "\n" pre_defined ^
    "\n/********************INCLUDE END*****************/\n" ^
    (List.fold_left (fun content x ->
    if x.t_name="Array" || x.t_name="ArrayElementT"
     || x.t_name="Array_add" || x.t_name="Array_length"
    then content else content ^ "struct " ^ x.t_actual ^ ";\n")
        "" (NameMap.fold (fun k v lst -> v :: lst) context.typetab [])) ^
    "\n/********************DECLARATION END****************/\n" ^
    (List.fold_left (fun content x -> content ^ (match x with
        | SGlobalType t -> generate_type "" t
        | _ -> raise (ProcessingError ("Generation error"))))
        "" (List.filter (fun x -> match x with SGlobalType s ->
        true | _ -> false) program)) ^
    "\n/****************TYPE DECLARATION ENDED************/\n" ^
    (List.fold_left (fun content x -> content ^ (match x with
        | SGlobalFunc t -> generate_func "" t
        | _ -> raise (ProcessingError ("Generation error"))))
        "" (List.filter (fun x -> match x with SGlobalFunc s -> true
        | _ -> false) program)) ^
    "\n/***************FUNCTION DECLARATION ENDED***********/\n" ^
    "int main() {\n" ^
    (List.fold_left (fun content x -> content ^ (match x with
        | SGlobalStmt t -> generate_stmt t
        | _ -> raise (ProcessingError ("Generation error"))))
        "" (List.filter (fun x -> match x with SGlobalStmt s ->
        true | _ -> false) program)) ^
    "return 0;\n}"
```

8.9 sast.ml

```
open Ast

type action =
  | NewVar
    | NewArr

let string_of_action = function
  | NewVar -> "new"
    | NewArr -> "new array"



type sem = {
  mutable actions: action list;
```

```
  type_def: type_entry
}

let rec string_of_type = function
  | BaseTypeEntry t -> t.t_name
  | ArrayTypeEntry t ->
  (string_of_type t) ^ "[]"

let string_of_sem s = "$" ^ (string_of_type s.type_def) ^ " "
                      ^ (String.concat " | "
                      (List.map string_of_action s.actions)) ^ "$"

type s_expr =                                   (* Expressions*)
  | SLiteral of string * sem      (* int, double, bool, string *)
  | SArrayLiteral of s_expr list * sem
  | SArrayIndex of s_expr * s_expr * sem               (* A[B[3]]*)
  | SArrayRange of s_expr * s_expr * s_expr * sem
  | SVar of string * sem             (* foo *)
  | SDotExpr of s_expr * string * sem        (* A.B *)
  | SBinop of s_expr * op * s_expr * sem      (* 3+4 *)
  | SCall of s_expr option * base_type * s_expr list * sem (* foo(a, b) *)
  | SArrayOperation of s_expr * string * s_expr list * sem
  | SBuildArray of type_entry * s_expr list * sem
  | SBuildClipArray of s_expr * sem
  | SClipTimeIndex of s_expr * s_expr * sem
  | SClipFrameIndex of s_expr * s_expr * sem
  | SClipTimeRange of s_expr * s_expr * s_expr * sem
  | SClipFrameRange of s_expr * s_expr * s_expr * sem
  | SClipConcat of s_expr * s_expr * sem
  | SClipTimeCascade of s_expr * s_expr * s_expr * sem
  | SClipFrameCascade of s_expr * s_expr * s_expr * sem
  | SClipPixel of s_expr * s_expr * s_expr * s_expr * sem

type s_stmt =
  | SAssign of s_expr option * s_expr
  | STimeSetAttribute of s_expr * string * s_expr * s_expr
  | SFrameSetAttribute of s_expr * string * s_expr * s_expr
  | SIfStmt of s_cond_exec list
  | SForIn of string * sem * s_expr * s_stmt list
  | SForRange of string * sem * s_expr * s_expr * s_stmt list
  * for_range_dir
  | SWhileStmt of s_expr * s_stmt list
  | SContinue
  | SBreak
  | SReturn of s_expr option

and s_cond_exec =
   SCondExec of s_expr option * s_stmt list


type s_var_decl =
  | SVarDecl of string * sem

and s_func_decl =
```

```
  | SFuncDecl of string * s_var_decl list * s_stmt list * sem

and s_type_decl =
  | STypeDecl of string * s_type_mem_decl list

and s_type_mem_decl =
  | SMemVarDecl of s_var_decl
  | SMemFuncDecl of s_func_decl
  | SMemTypeDecl of s_type_decl

and s_global_ele_decl =
  | SGlobalStmt of s_stmt
  | SGlobalFunc of s_func_decl
  | SGlobalType of s_type_decl

type s_program = s_global_ele_decl list

let rec string_of_s_expr = function
  | SLiteral (str, s) -> str ^ (string_of_sem s)
  | SArrayLiteral (selist, sem) -> let s = (List.fold_left
  (fun a b ->
   a ^ ", " ^ b) "" (List.map string_of_s_expr selist)) in
  "[" ^ (String.sub s 2 ((String.length s) - 2))  ^ "]" ^
  (string_of_sem sem)
  | SArrayIndex (sout, sin, s) -> (string_of_s_expr sout) ^
  "[" ^ (string_of_s_expr sin) ^ "]" ^ (string_of_sem s)
    | SArrayRange (arr, st, ed, s) -> (string_of_s_expr arr) ^
    "[" ^ (string_of_s_expr st) ^ "," ^ (string_of_s_expr ed) ^
    "]" ^ (string_of_sem s)
  | SVar (id, s) -> id ^ (string_of_sem s)
  | SDotExpr (sexpr, id, s) -> (string_of_s_expr sexpr) ^
  "." ^ id ^ (string_of_sem s)
  | SBinop (lsexpr, op, rsexpr, s) -> (string_of_s_expr lsexpr) ^
  " " ^ (string_of_op op) ^ " " ^ (string_of_s_expr rsexpr)
  ^ (string_of_sem s)
  | SCall (obj, func_type, el, s) -> (match obj with
          | None -> "" | Some st ->
          (string_of_s_expr st) ^ "." ) ^ func_type.t_name ^ "(" ^
          (String.concat ", " (List.map string_of_s_expr el)) ^ ")"
          ^ (string_of_sem s)
  | SClipTimeRange (cl, st, ed, s) -> (string_of_s_expr cl) ^ "[" ^
  (string_of_s_expr st) ^ "," ^ (string_of_s_expr ed) ^ "]"
  ^ (string_of_sem s)
  | SClipFrameRange (cl, st, ed, s) -> (string_of_s_expr cl) ^ "[" ^
  (string_of_s_expr st) ^ "," ^ (string_of_s_expr ed) ^ "]"
  ^ (string_of_sem s)
  | SClipTimeCascade (cl1, cl2, tm, s) -> (string_of_s_expr cl1) ^
  "^" ^ (string_of_s_expr cl2) ^ "@" ^ (string_of_s_expr tm)
  ^ (string_of_sem s)
  | SClipFrameCascade (cl1, cl2, tm, s) -> (string_of_s_expr cl1) ^ "^"
  ^ (string_of_s_expr cl2) ^ "@" ^ (string_of_s_expr tm)
  ^ (string_of_sem s)
  | SClipConcat (cl1, cl2, s) -> (string_of_s_expr cl1) ^ "&" ^
   (string_of_s_expr cl2)  ^ (string_of_sem s)
```

```
  | SClipTimeIndex (cl, idx, s) -> (string_of_s_expr cl) ^
   "[" ^ (string_of_s_expr idx) ^ "]" ^ (string_of_sem s)
  | SClipFrameIndex (cl, idx, s) -> (string_of_s_expr cl) ^
  "[" ^ (string_of_s_expr idx) ^ "]" ^ (string_of_sem s)
  | SClipPixel (cl, x, y, tm, s) -> (string_of_s_expr cl) ^ "<"
  ^ (string_of_s_expr x) ^ (string_of_s_expr y) ^ ">" ^ "@"
   ^ (string_of_s_expr tm) ^ (string_of_sem s)
  | SBuildArray (t, el, s) -> (string_of_type t) ^ "(" ^
  (String.concat ", " (List.map string_of_s_expr el)) ^ ")"
  ^ (string_of_sem s)
  | SBuildClipArray (d, s) -> "Clips under \"" ^
  (string_of_s_expr d) ^ "\"" ^ (string_of_sem s)
  | SArrayOperation (smain, fname, el, s) ->
  (string_of_s_expr smain) ^ "." ^ fname ^ "(" ^
  (String.concat ", "
  (List.map string_of_s_expr el)) ^ ")" ^ (string_of_sem s)

and string_of_s_stmt = function
  | SAssign(None,rvalue) -> string_of_s_expr rvalue
  | SAssign(Some(lvalue), rvalue) -> (string_of_s_expr lvalue) ^
  " = " ^ (string_of_s_expr rvalue)
  | STimeSetAttribute(main, attr, time, value) ->
  (string_of_s_expr main) ^ attr ^ "@" ^
  (string_of_s_expr time) ^ " = " ^ (string_of_s_expr value)
  | SFrameSetAttribute(main, attr, time, value) ->
   (string_of_s_expr main) ^ attr ^ "@" ^
   (string_of_s_expr time) ^ " = " ^ (string_of_s_expr value)
  | SIfStmt(conds) ->
  string_of_s_first_cond_exec (List.hd conds) ^ "\n" ^
  (String.concat "\n" (List.map string_of_s_cond_exec (List.tl conds)))
  | SForIn(var, s, expr, stmts) -> "for " ^ var ^
  (string_of_sem s) ^ " in " ^ (string_of_s_expr expr)
    ^ ":\n" ^ (String.concat "\n" (List.map string_of_s_stmt stmts))
  | SForRange(var, s, exprst, expred, stmts, dir) ->
   "for " ^ var ^ (string_of_sem s) ^ " = " ^ (string_of_s_expr exprst)
    ^ (match dir with | Inc -> " to " | Dec -> " downto ") ^
    (string_of_s_expr expred) ^ ":\n" ^(String.concat "\n"
    (List.map string_of_s_stmt stmts))
  | SWhileStmt(expr, stmts) -> "while " ^ (string_of_s_expr expr) ^ ":\n"
    ^ (String.concat "\n" (List.map string_of_s_stmt stmts))
  | SContinue -> "continue"
  | SBreak -> "break"
  | SReturn(None) -> "return"
  | SReturn(Some(expr)) -> "return " ^ (string_of_s_expr expr)

and string_of_s_first_cond_exec = function
  | SCondExec(None, stmts) ->
  "else:" ^ (String.concat "\n" (List.map string_of_s_stmt stmts))
  | SCondExec(Some(expr), stmts) ->
  "if " ^ (string_of_s_expr expr) ^ ":\n" ^
  (String.concat "\n" (List.map string_of_s_stmt stmts))

and string_of_s_cond_exec = function
  | SCondExec(None, stmts) ->
```

```
  "else:" ^ (String.concat "\n" (List.map string_of_s_stmt stmts))
  | SCondExec(Some(expr), stmts) ->
  "elif " ^ (string_of_s_expr expr) ^ ":\n" ^
  (String.concat "\n" (List.map string_of_s_stmt stmts))


let extract_semantic = function
  | SLiteral (_, s) -> s
  | SArrayLiteral (_, s) -> s
  | SArrayIndex (_, _, s) -> s
  | SArrayRange (_, _, _, s) -> s
  | SVar (_, s) -> s
  | SDotExpr (_, _, s) -> s
  | SBinop (_, _, _, s) -> s
  | SCall (_, _, _, s) -> s
  | SClipTimeIndex (_, _, s) -> s
  | SClipFrameIndex (_, _, s) -> s
  | SClipTimeRange (_, _, _, s) -> s
  | SClipFrameRange (_, _, _, s) -> s
  | SClipTimeCascade (_, _, _, s) -> s
  | SClipFrameCascade (_, _, _, s) -> s
  | SClipConcat (_, _, s) -> s
  | SClipPixel (_,_,_,_,s) -> s
  | SBuildArray (_, _, s) -> s
  | SBuildClipArray (_, s) -> s
  | SArrayOperation (_, _, _, s) -> s


let rec string_of_s_var_decl = function
  | SVarDecl(id, s) -> id ^ (string_of_sem s)

and string_of_s_func_decl = function
  | SFuncDecl(name, args, stmts, s) ->
  "func " ^ name ^ " (" ^ (String.concat ", "
  (List.map string_of_s_var_decl args))
    ^ ")\n" ^ (String.concat "\n"
    (List.map string_of_s_stmt stmts)) ^ (string_of_sem s)

and string_of_s_type_decl = function
  | STypeDecl(name, args) ->
  "type " ^ name ^ "\n" ^ (String.concat "\n"
  (List.map string_of_s_type_mem_decl args))

and string_of_s_type_mem_decl = function
  | SMemVarDecl o -> string_of_s_var_decl o
  | SMemFuncDecl o -> string_of_s_func_decl o
  | SMemTypeDecl o -> string_of_s_type_decl o

and string_of_s_global_ele_decl = function
  | SGlobalStmt o -> string_of_s_stmt o
  | SGlobalFunc o -> string_of_s_func_decl o
  | SGlobalType o -> string_of_s_type_decl o

and string_of_s_program program =
```

```
  String.concat "\n" (List.map string_of_s_global_ele_decl program)
```

8.10 print_typetab.ml

```
open Ast

let rec print_typeentry curtype = function
    | BaseTypeEntry(be) -> (curtype ^ be.t_name)
    | ArrayTypeEntry(ae) -> (print_typeentry (curtype^"array of ") ae)

and print_varentry k =
    print_string("v_name: " ^ k.v_name ^ ", v_actual: " ^ k.v_actual);
    print_string (print_typeentry "" k.v_type);
    print_string "\n";

and print_memberentry k v =
    print_string ("Key: "^k^"\n");
    print_string (print_typeentry "" v);
    print_string "\n";

and print_basetype k (bt:base_type) =
    print_string ("Typetab KEY: " ^ k ^ "\n");
    print_string ("t_name: "^bt.t_name^", ");
    print_string ("t_actual: "^bt.t_actual^", "^"\n");
    print_string "Eval:\n";
    List.iter (fun x -> print_evalentry x) bt.evals;
    print_string ("Member: "^"\n");
    NameMap.iter print_memberentry bt.members;
    print_string ("\n");

and print_evalentry ee =
    print_string "args: ";
    List.iter (fun k -> print_varentry k;()) ee.args;
    print_string "ret: ";
    print_string (print_typeentry "" ee.ret);
    print_string "\n";

and printtypetab t =
   NameMap.iter (fun k v-> print_basetype k v) t;
```

8.11 makefile

```
#For those machine doesn't have ocamlbuild,
#build the project with this makefile
#
OBJ=ast.cmo \
    scanner.cmo \
    parser.cmo \
    parser_test.cmo \
    sast.cmo \
    semantic.cmo \
    type_reader.cmo \
```

```
    print_typetab.cmo \
    generate.cmo \
    generate_test.cmo \
    typereader_test.cmo  \
    semantic_test.cmo

YO=semantic_test

FLAGS:=-g

$(YO): $(OBJ)
    ocamlc -g -o parser_test parser.cmo \
    scanner.cmo ast.cmo parser_test.cmo
    ocamlc -g -o typereader_test parser.cmo scanner.cmo \
    ast.cmo sast.cmo type_reader.cmo print_typetab.cmo\
     typereader_test.cmo
    ocamlc -g -o generate_test parser.cmo scanner.cmo \
    ast.cmo sast.cmo semantic.cmo type_reader.cmo \
    print_typetab.cmo generate.cmo generate_test.cmo
    ocamlc -g -o semantic_test parser.cmo scanner.cmo \
    ast.cmo sast.cmo semantic.cmo type_reader.cmo \
    print_typetab.cmo semantic_test.cmo

.SUFFIXES: .ml .cmo .cmi .mll .mly .mli
.PRECIOUS: %.ml %.mli %.cmo

.ml.cmo:
    ocamlc -c $(FLAGS) $<

.mli.cmi:
    ocamlc -c $(FLAGS) $<

.mll.ml:
    ocamllex $<

.mly.ml:
    ocamlyacc -v $<

.mly.mli:
    ocamlyacc -v $<

clean:
    rm -f *.cmi *.cmo parser.ml scanner.ml \
    *.output parser.mli parser_test semantic_test generate_test

# Generated by ocamldep
ast.cmo:
ast.cmx:
parser.cmo: ast.cmo parser.cmi
parser.cmx: ast.cmx parser.cmi
parser.cmi: ast.cmo
scanner.cmo: parser.cmi
scanner.cmx: parser.cmx
parser_test.cmo: scanner.cmo parser.cmi ast.cmo
```

```
parser_test.cmx: scanner.cmx parser.cmx ast.cmx
sast.cmo: ast.cmo
sast.cmx: ast.cmx
semantic.cmo: sast.cmo ast.cmo
semantic.cmx: sast.cmx ast.cmx
type_reader.cmo: ast.cmo
type_reader.cmx: ast.cmx
print_typetab.cmo: ast.cmo
print_typetab.cmx: ast.cmx
generate.cmo: sast.cmo ast.cmo
generate.cmx: sast.cmx ast.cmx
generate_test.cmo: type_reader.cmo semantic.cmo \
scanner.cmo parser.cmi generate.cmo ast.cmo
generate_test.cmx: type_reader.cmx semantic.cmx \
scanner.cmx parser.cmx generate.cmx ast.cmx
typereader_test.cmo: type_reader.cmo scanner.cmo \
print_typetab.cmo parser.cmi ast.cmo
typereader_test.cmx: type_reader.cmx scanner.cmx \
print_typetab.cmx parser.cmx ast.cmx
semantic_test.cmo: type_reader.cmo semantic.cmo \
scanner.cmo sast.cmo parser.cmi ast.cmo
semantic_test.cmx: type_reader.cmx semantic.cmx \
scanner.cmx sast.cmx parser.cmx ast.cmx
```

8.12 test.sh

```
#!/bin/sh

YO="./parser_test"
binaryoutput="./a.out"
preproc_path="preprocessor.py"

# Set time limit for all operations
ulimit -t 30

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

keep=0

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

SignalError() {
    if [ $error -eq 0 ] ; then
    echo "\033[31m FAILED \033[0m"
    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 $* || {
        if [[ $5 != *fail* ]]; then
           SignalError "$1 failed on $*"
           return 1
        fi
    }
}

CheckPreprocessor() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                              s/.yo//'`
    reffile=`echo $1 | sed 's/.yo$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."
    echo -n "$basename................."
    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    Compare "../test/preprocessor/intermediate/ \
    $basename.yo" ${reffile}.out ${basename}.a.diff

    if [ $error -eq 0 ] ; then
    if [ $keep -eq 0 ] ; then
        rm -f $generatedfiles
    fi
    echo "\033[32m OK \033[0m"
    echo "###### SUCCESS" 1>&2
    else
    echo "###### FAILED" 1>&2
    globalerror=$error
    fi
}
```

```
CheckParser() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                             s/.yo//'`
    reffile=`echo $1 | sed 's/.yo$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."
    echo -n "$basename.................."
    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    YO="./parser_test"
    generatedfiles="$generatedfiles ${basename}.a.out" &&
    Run "$YO" "<" "../test/parser/intermediate/$basename.yo" ">" \
    ${basename}.a.out "2>" ${basename}.a.out &&

    Compare ${basename}.a.out ${reffile}.out ${basename}.a.diff

    if [ $error -eq 0 ] ; then
    if [ $keep -eq 0 ] ; then
        rm -f $generatedfiles
    fi
    echo "\033[32m OK \033[0m"
    echo "###### SUCCESS" 1>&2
    else
    echo "###### FAILED" 1>&2
    globalerror=$error
    fi
}


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

    echo -n "$basename..."
    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    # old from microc - interpreter
    # generatedfiles="$generatedfiles ${basename}.i.out" &&
    # Run "$YO" "-i" "<" $1 ">" ${basename}.i.out &&
    # Compare ${basename}.i.out ${reffile}.out ${basename}.i.diff

    generatedfiles="$generatedfiles ${basename}.c.out" &&
    Run "$YO" "-c" $1 ">" ${basename}.c.out &&
```

```
    Compare ${basename}.c.out ${reffile}.out ${basename}.c.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
    if [ $keep -eq 0 ] ; then
        rm -f $generatedfiles
    fi
    echo "OK"
    echo "###### SUCCESS" 1>&2
    else
    echo "###### FAILED" 1>&2
    globalerror=$error
    fi
}
CheckFail() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                              s/.yo//'`
    reffile=`echo $1 | sed 's/.yo$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

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

    generatedfiles=""

    # old from microc - interpreter
    # generatedfiles="$generatedfiles ${basename}.i.out" &&
    # Run "$YO" "-i" "<" $1 ">" ${basename}.i.out &&
    # Compare ${basename}.i.out ${reffile}.out ${basename}.i.diff

    generatedfiles="$generatedfiles ${basename}.c.out" &&
    {
        Run "$YO" "-b" $1 "2>" ${basename}.c.out ||
        Run "$binaryoutput" ">" ${basename}.b.out
    } &&
    Compare ${basename}.c.out ${reffile}.out ${basename}.c.diff

    # Report the status and clean up the generated files

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

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

    echo -n "$basename..."

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

    generatedfiles=""

    YO="./semantic_test"
    generatedfiles="$generatedfiles ${basename}.f.cpp \
    ${basename}.f.out yo.prog"
    Run "$YO" "<" "../test/semantic/intermediate/$basename.yo" ">" \
    ${basename}.s.out "2>" ${basename}.s.out &&
    Compare ${basename}.s.out ${reffile}.out ${basename}.s.diff

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

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

    echo -n "$basename..."
    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""
    tmpfiles=""
    # old from microc - interpreter
    # generatedfiles="$generatedfiles ${basename}.i.out" &&
    # Run "$YO" "-i" "<" $1 ">" ${basename}.i.out &&
    # Compare ${basename}.i.out ${reffile}.out ${basename}.i.diff
    YO="./typereader_test"
    generatedfiles="$generatedfiles ${basename}.f.cpp \
```

```
    ${basename}.f.out yo.prog"
    Run "$YO" "<" "../test/typereader/intermediate/$basename.yo" ">" \
     ${basename}.f.out "2>" ${basename}.f.out &&
    #g++ ${basename}.f.cpp libclip.cpp yolib.h -lstdc++ \
    -lopenshot-audio -lopenshot -I/usr/local/include/libopenshot \
    -I/usr/local/include/libopenshot-audio -lconfig++ \
    -lavdevice -lavformat  \
    -lavcodec -lavutil -lz `pkg-config --cflags --libs libconfig++ \
    Qt5Gui Qt5Widgets Magick++` -fPIC -std=c++11 -o yo.prog
    #g++ -o yo.prog ${basename}.f.cpp yolib.h -std=c++11 &&
    #Run "./yo.prog" ">" ${basename}.f.out &&
    Compare ${basename}.f.out ${reffile}.out ${basename}.f.diff


    #generatedfiles="$generatedfiles ${basename}.f.out" &&
    #tmpfiles="$tmpfiles tests/${basename}.lrx_lrxtmp.c a.out" &&
    #Run "$YO" "-b" $1 &&
    #Run "$binaryoutput" ">" ${basename}.f.out &&
    #Compare ${basename}.f.out ${reffile}.out ${basename}.f.diff

    #rm -f $tmpfiles

    # Report the status and clean up the generated files

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

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

    echo -n "$basename..."
    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""
    tmpfiles=""
    # old from microc - interpreter
    # generatedfiles="$generatedfiles ${basename}.i.out" &&
    # Run "$YO" "-i" "<" $1 ">" ${basename}.i.out &&
    # Compare ${basename}.i.out ${reffile}.out ${basename}.i.diff
    YO="./generate_test"
```

```
    generatedfiles="$generatedfiles ${basename}.f.cpp \
    ${basename}.f.out yo.prog"
    Run "$YO" "<" "../test/intermediate/$basename.yo" ">" \
    ${basename}.f.cpp &&
    g++ ${basename}.f.cpp yolib.h -lstdc++ -lopenshot-audio
    -lopenshot -I/usr/local/include/libopenshot \
    -I/usr/local/include/libopenshot-audio -lconfig++ -lavdevice \
    -lavformat  -lavcodec -lavutil -lz `pkg-config --cflags --libs \
    libconfig++ Qt5Gui Qt5Widgets Magick++` -fPIC \
    -std=c++11 -o yo.prog
    #g++ -o yo.prog ${basename}.f.cpp yolib.h -std=c++11 &&
    Run "./yo.prog" ">" ${basename}.f.out &&
    Compare ${basename}.f.out ${reffile}.out ${basename}.f.diff


    #generatedfiles="$generatedfiles ${basename}.f.out" &&
    #tmpfiles="$tmpfiles tests/${basename}.lrx_lrxtmp.c a.out" &&
    #Run "$YO" "-b" $1 &&
    #Run "$binaryoutput" ">" ${basename}.f.out &&
    #Compare ${basename}.f.out ${reffile}.out ${basename}.f.diff

    #rm -f $tmpfiles

    # Report the status and clean up the generated files

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


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

    echo -n "$basename..."
    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""
    tmpfiles=""

    YO="./generate_test"
    generatedfiles="$generatedfiles ${basename}.f.cpp \
```

```
    ${basename}.f.out yo.prog"
    echo "\n"
    echo "\n"
    Run "$YO" "<" "../test/intermediate/$basename.yo" &&
    #g++ ${basename}.f.cpp libclip.cpp yolib.h -lstdc++ \
    -lopenshot-audio -lopenshot -I/usr/local/include/libopenshot \
    -I/usr/local/include/libopenshot-audio \
    -lconfig++ -lavdevice -lavformat  -lavcodec \
    -lavutil -lz `pkg-config --cflags --libs libconfig++ \
    Qt5Gui Qt5Widgets Magick++` -fPIC -std=c++11 -o yo.prog
    #g++ -o yo.prog ${basename}.f.cpp yolib.h -std=c++11 &&
    #Run "./yo.prog" ">" ${basename}.f.out &&
    #Compare ${basename}.f.out ${reffile}.out ${basename}.f.diff

    if [ $error -eq 0 ] ; then
    if [ $keep -eq 0 ] ; then
        rm -f $generatedfiles
    fi
    echo "\n"
    globalerror=$error
    fi
}


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

shift `expr $OPTIND - 1`

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

for file in $files
do
    case $file in
    *test-preprocess*)
        echo "##### Now Testing Preprocessor #####"
        echo "preprocessing....."
        python $preproc_path $file
        echo "\033[32m OK \033[0m"
        CheckPreprocessor $file 2>> $globallog
        ;;
```

```
    *test-parser*)
        echo "##### Now Testing Parser #####"
        echo "preprocessing....."
        python $preproc_path $file
        echo "\033[32m OK \033[0m"
        CheckParser $file 2>> $globallog
        ;;
    *test-semantic*)
        echo "##### Now Testing Semantic Analysis #####"
        echo "preprocessing....."
        python $preproc_path $file
        echo "\033[32m OK \033[0m"
        CheckSemanticAnalysis $file 2>> $globallog
        ;;
    *test-full*)
        echo "##### Now Testing FullStack #####"
        echo "preprocessing....."
        python $preproc_path $file
        echo "\033[32m OK \033[0m"
        TestRunningProgram $file 2>> $globallog
        ;;
    *test-munan*)
        echo "##### Now Testing Munan #####"
        echo "preprocessing....."
        python $preproc_path $file
        echo "\033[32m OK \033[0m"
        MunanTest $file
        ;;
    *test-typereader*)
        echo "##### Now Testing Single FullStack #####"
        echo "preprocessing....."
        python $preproc_path $file
        echo "\033[32m OK \033[0m"
        TestTypeReader $file 2>> $globallog
        ;;
    *test-fail*)
        CheckFail $file 2>> $globallog
        ;;
    *test-*)
        Check $file 2>> $globallog
        ;;

        #echo "unknown file type $file"
        #globalerror=1
        #;;
    esac
done

exit $globalerror
```

8.13 yolib.h

```
#include "/usr/local/include/libopenshot/OpenShot.h"
```

```
#include <dirent.h>
#include <libconfig.h++>
#include <iostream>
#include <fstream>
#include <sys/types.h>
#include <errno.h>
#include <vector>
#include <string>
#include <algorithm>
#include <memory>

using namespace std;
using namespace libconfig;
/*Global output configuration*/
int V_FPS=24;
int V_WIDTH=640;
int V_HEIGHT=360;
int V_PIXEL_RATIO=1;
int V_BIT_RATE=240000;

struct _Pixel{
    int R;
    int G;
    int B;
    _Pixel(int r, int g, int b){
        R =r; G =g; B=b;
    }
};


void readConfig(){
    Config cfg;
    /*Returns all parameters in this structure */
    try{
        cfg.readFile("config.ini");
    }
    catch(const FileIOException &fioex)
    {
        std::cerr << "No config found, use default." \
        << std::endl;
        return;
    }
    catch(const ParseException &pex)
    {
        std::cerr << "Parse error at " << pex.getFile() \
        << ":" << pex.getLine()
                  << " - " << pex.getError() << std::endl;
        return;
    }

    // Get the store name.
    try{
        int tmp = cfg.lookup("fps");
        V_FPS = tmp;
```

```
    }
    catch(const SettingNotFoundException &nfex){}
    try{
        int tmp = cfg.lookup("width");
        V_WIDTH = tmp;
    }
    catch(const SettingNotFoundException &nfex){}
    try{
        int tmp = cfg.lookup("height");
        V_HEIGHT = tmp;
    }
    catch(const SettingNotFoundException &nfex){}
    try{
        int tmp = cfg.lookup("pixel_ratio");
        V_PIXEL_RATIO = tmp;
    }
    catch(const SettingNotFoundException &nfex){}
    try{
        int tmp = cfg.lookup("bit_rate");
        V_BIT_RATE = tmp;
    }
    catch(const SettingNotFoundException &nfex){}
}

template<typename T>
void pop_front(std::vector<T>& vec)
{
    assert(!vec.empty());
    vec.erase(vec.begin());
}


int getdir (string dir, vector<string> &files)
{
    DIR *dp;
    struct dirent *dirp;
    if((dp  = opendir(dir.c_str())) == NULL) {
        cout << "Error(" << errno << ") opening " << dir << endl;
        return errno;
    }

    while ((dirp = readdir(dp)) != NULL) {
        files.push_back(string(dirp->d_name));
    }
    closedir(dp);
    // sort by filename
    sort(files.begin(),files.end());
    //remove . and ..
    pop_front(files);
    pop_front(files);
    return 0;
}

string getextension(string filename){
```

```
    int idx = filename.rfind('.');
    string extension = "";
    if (idx != string::npos){
        extension = filename.substr(idx + 1);
    }
    transform(extension.begin(), extension.end(), \
    extension.begin(), ::tolower);
    return extension;
}

int isVideo(string filename){
    string imagetype[5] = {"jpg","jpeg","png","bmp","gif"};
    string videotype[11] =
    {"webm","f4v","mov","rmvb","mp4","rm",\
    "wmv","avi","flv","3gp","mkv"};
    string extension = getextension(filename);
    for (int i = 0; i < 5; i++)
        if (extension == imagetype[i])
            return 0;
    for (int i = 0; i < 11; i++)
        if (extension == videotype[i])
            return 1;
    return -1;
}


struct Universal {};
using std::string;
struct _Clip : Universal  {

    tr1::shared_ptr<Timeline> __instance__;

    _Clip(tr1::shared_ptr<Timeline> tl) {
        __instance__ = tl;
    }
    static tr1::shared_ptr<_Clip> \
    eval(tr1::shared_ptr<Universal> obj, string fileName);

};

tr1::shared_ptr<_Clip> fromTimeline(tr1::shared_ptr<Timeline> \
tlptr) {
    return tr1::shared_ptr<_Clip>(new _Clip(tlptr));
}

/*create a clip from a file,
  yo prog:
  r = Clip("output.webm")
  generate as:
  tr1::shared_ptr<Timeline> r = createClip("output.webm");
*/

std::string logClip(tr1::shared_ptr<_Clip> _clip){
    list<Clip*> cliplists = _clip->__instance__->Clips();
```

```
    for (std::list<Clip*>::const_iterator iterator = \
    cliplists.begin(), end = cliplists.end(); iterator != end; \
  ++iterator){
        return (*iterator)->Json();
    }
}

tr1::shared_ptr<_Clip> createClip(string filename){
    //check the file type, an image or a video
    int filetype = isVideo(filename);
    if (filetype == 1){
        FFmpegReader* reader = new FFmpegReader(filename);
        // essential: Open the reader otherwise you cannot read
        reader->Open();
        Clip* clip = new Clip(reader);
        tr1::shared_ptr<Timeline> r (new \
        Timeline(V_WIDTH, V_HEIGHT, Fraction(V_FPS, 1), \
        44100, 2, LAYOUT_STEREO));
        r->AddClip(clip);
        r->Open();
        reader->Close();
        return fromTimeline(r);
    }else if (filetype == 0){
        ImageReader* reader = new ImageReader(filename);
        reader->Open();
        Clip* clip = new Clip(reader);
        tr1::shared_ptr<Timeline> r (new Timeline(V_WIDTH, \
        V_HEIGHT, Fraction(V_FPS, 1), 44100, 2, LAYOUT_STEREO));
        r->AddClip(clip);
        r->Open();
        reader->Close();
        return fromTimeline(r);
    }else{
        exit(-1);
    }

}
/* create clips from a dir
   yo:prog:
   clips = readclips("dirname/")
   generate as:
   std::vector<tr1::shared_ptr<Timeline>> clips = \
   createClips("dir/");
*/

tr1::shared_ptr<std::vector<tr1::shared_ptr<_Clip>>> \
createClips(string dirname){
    //read files into Filenames
    auto Filenames = vector<string>();
    getdir(dirname,Filenames);

    auto res = tr1::shared_ptr<vector<tr1::shared_ptr<_Clip>> \
    >(new vector<tr1::shared_ptr<_Clip>>());
```

```
    //auto res = \
    tr1::shared_ptr<std::vector<tr1::shared_ptr<_Clip>>>();
    int len = Filenames.size();
    for (int i = 0; i < len; i++){
        //std::cout << Filenames[i] << endl;
        // form clips
        res->push_back(createClip(dirname + Filenames[i]));
    }
    return res;
}

/* clips addition
   yo:prog:
   clip = clip1 + clip2
   generate as:
   tr1::shared_ptr<Timeline> clip = addClip(clip1,clip2);
*/

tr1::shared_ptr<_Clip> addClip(tr1::shared_ptr<_Clip> \
lop, tr1::shared_ptr<_Clip> rop){
    list<Clip*> leftlists = lop->__instance__->Clips();
    list<Clip*> rightlists = rop->__instance__->Clips();
    tr1::shared_ptr<Timeline> res (new \
    Timeline(V_WIDTH, V_HEIGHT, Fraction(V_FPS, 1), \
    44100, 2, LAYOUT_STEREO));

    double maxpos = 0.0;
    for (std::list<Clip*>::const_iterator iterator = \
    leftlists.begin(), end = leftlists.end(); iterator != end; \
  ++iterator) {
        // directly add clip to the timeline
        res->AddClip(*iterator);
        // find out how much time clip2 should shift
        if ((*iterator)->Position() + ((*iterator)->End()- \
    (*iterator)->Start()) > maxpos){
            maxpos = (*iterator)->Position()+ \
        ((*iterator)->End() - (*iterator)->Start());
        }
    }

    for (std::list<Clip*>::const_iterator iterator = \
    rightlists.begin(), end = rightlists.end(); iterator != end; \
  ++iterator) {
        // shift clip2
        (*iterator)->Position((*iterator)->Position() + maxpos);
        res->AddClip(*iterator);
    }
    res->Open();
    return fromTimeline(res);
}

/* clips layering
   yo:prog:
   clip = clip1 ^ clip2 @ 1.0
```

```
   generate as:
   tr1::shared_ptr<Timeline> clip = layerClip(clip1,clip2,1.0);
*/

tr1::shared_ptr<_Clip> layerClip(tr1::shared_ptr<_Clip> \
bottom, tr1::shared_ptr<_Clip> top, double shifttime){
    list<Clip*> bottomlists = bottom->__instance__->Clips();
    list<Clip*> toplists = top->__instance__->Clips();
    tr1::shared_ptr<Timeline> res(new\
     Timeline(V_WIDTH, V_HEIGHT, Fraction(V_FPS, 1), \
     44100, 2, LAYOUT_STEREO));

    int maxlayer = 0;
    for (std::list<Clip*>::const_iterator iterator = \
    bottomlists.begin(), end = bottomlists.end(); \
    iterator != end; ++iterator){
        //std::cout << (*iterator)->Layer() << std::endl;
        if ((*iterator)->Layer() > maxlayer){
            maxlayer = (*iterator)->Layer();
        }
        res->AddClip(*iterator);
    }

    for (std::list<Clip*>::const_iterator iterator =\
     toplists.begin(), end = toplists.end(); iterator != end; \
   ++iterator) {
        (*iterator)->Layer((*iterator)->Layer() + maxlayer);
        (*iterator)->Position((*iterator)->Position() + shifttime);
        res->AddClip(*iterator);
    }
    res->Open();
    return fromTimeline(res);
}

tr1::shared_ptr<_Clip> layerClip(tr1::shared_ptr<_Clip> \
bottom, tr1::shared_ptr<_Clip> top, int shiftframe){
    double shifttime = double(shiftframe) / V_FPS;
    return layerClip(bottom,top,shifttime);
}
/* write clips to a file
   yo:prog:
   write(clip,"filename.mp4")
   generate as:
   writeClips(clip,"filename.mp4");
*/

void writeClips(tr1::shared_ptr<_Clip> _clip, \
string filename){
    auto clip = _clip->__instance__;
    //std::cout << (clip)->Json() << std::endl;
    FFmpegWriter w(filename);
    string extension = getextension(filename);
    w.SetAudioOptions(true, "libvorbis", \
    44100, 2, LAYOUT_STEREO, 188000);
```

```
    //if (extension == "webm")
    w.SetVideoOptions(true, "libvpx", \
    Fraction(V_FPS,1), V_WIDTH, V_HEIGHT, \
    Fraction(V_PIXEL_RATIO,1), false, false, V_BIT_RATE);

    w.Open();
    // calculate the ending time
    double totaltime = 0;
    list<Clip*> lists = clip->Clips();
    for (std::list<Clip*>::const_iterator \
    iterator = lists.begin(), end = lists.end();\
     iterator != end; ++iterator) {
        if ((*iterator)->Position() + ((*iterator)->End() - \
    (*iterator)->Start()) > totaltime){
            totaltime = (*iterator)->Position()+\
         ((*iterator)->End() - (*iterator)->Start());
        }
        //std::cout << (*iterator)->Position() << " " \
    << (*iterator)->Start() << " " << (*iterator)->End() << \
  std::endl;
        //std::cout << (*iterator)->Json() << endl;
    }
    int totalframe = int(V_FPS * totaltime) + 1;
    std::cout << "Rendering... Totalframe:" \
    << totalframe << std::endl;
    w.WriteFrame(&(*clip), 1, totalframe);
    w.Close();
}

/*
    clipRange argument is double (seconds)
    yo:prog:
    a = clip[2.0:3.0]
    generate as:
    a = clipRange(clip,2.0,3.0);
*/

tr1::shared_ptr<_Clip> clipRange(tr1::shared_ptr<_Clip>\
 _clip,double starttime, double endtime){
    assert(starttime <= endtime);
    list<Clip*> cliplists = _clip->__instance__->Clips();
    tr1::shared_ptr<Timeline> res(new \
    Timeline(V_WIDTH, V_HEIGHT, Fraction(V_FPS, 1), \
    44100, 2, LAYOUT_STEREO));
    for (std::list<Clip*>::const_iterator iterator = \
    cliplists.begin(), end = cliplists.end(); iterator != end; \
  ++iterator){
        bool modifyhead = false;
        bool modifyend = false;
        if ((*iterator)->Position() < starttime){
            if ((*iterator)->Position() + \
            ((*iterator)->End() - (*iterator)->Start()) < starttime){
                continue;
            }
```

```
            modifyhead = true;
        }
        if ((*iterator)->Position() < endtime){
            if ((*iterator)->Position() + \
            ((*iterator)-> End() - (*iterator)->Start()) > endtime){
                modifyend = true;
            }
        }
        if ((*iterator)->Position() >= endtime){
            continue;
        }
        if (modifyhead){
            (*iterator)->Start(starttime - (*iterator)->Position());
        }
        if (modifyend){
            (*iterator)->End(endtime - (*iterator)->Position());
        }
        res->AddClip(*iterator);
    }
    res->Open();
    return fromTimeline(res);
}

/*
    clipRange argument is integer (frames)
    yo:prog:
    a = clip[24:48]
    generate as:
    a = clipRange(clip,24,48);
*/

tr1::shared_ptr<_Clip> clipRange(tr1::shared_ptr<_Clip> \
_clip,int startframe, int endframe){
    double starttime = double (startframe) / V_FPS;
    double endtime = double (endframe) / V_FPS;
    return clipRange(_clip, starttime, endtime);
}

/*
    clipIndex argument is integer (frame)
    yo:prog:
    a = clip[24]
    generate as:
    a = clipIndex(clip,24);
*/

tr1::shared_ptr<Frame> clipIndex(tr1::shared_ptr<_Clip> \
_clip,int frame){
    return _clip->__instance__->GetFrame(frame);
}


/*
    clipIndex argument is double (time)
```

```
    yo:prog:
    a = clip[2.4]
    generate as:
    a = clipIndex(clip,2.4);
*/

tr1::shared_ptr<Frame> clipIndex(tr1::shared_ptr<_Clip> \
_clip,double frametime){
    int frame = int(frametime * V_FPS);
    return _clip->__instance__->GetFrame(frame);
}


/*
return a matrix of
pixel : R G B{}
pixel = getpixel(clip,frame,i,j)
*/

tr1::shared_ptr<_Pixel> getPixel(tr1::shared_ptr<_Clip> \
_clip, int frame, int x, int y){
    tr1::shared_ptr<Frame> f = _clip->__instance__->\
    GetFrame(frame);
    const unsigned char* pixels = f->GetPixels(x);
    tr1::shared_ptr<_Pixel> \
    res(new _Pixel(int(pixels[3 * y]),int(pixels[3 * y + 1]), \
    int(pixels[3 * y + 2])));
    return res;
}

tr1::shared_ptr<_Pixel> getPixel(tr1::shared_ptr<_Clip> _clip, \
double frametime, int x, int y){
    int frame = int (frametime*V_FPS);
    return getPixel(_clip,frame,x,y);
}

/*
set pixel
no use now
*/

template< typename T >
std::string int_to_hex( T i )
{
  std::stringstream stream;
  stream << std::setfill ('0') << std::setw(2)
         << std::hex << i;
  return stream.str();
}

/*
void setPixel(tr1::shared_ptr<_Clip> _clip, int frame, \
int x, int y, pixel res){
    tr1::shared_ptr<Frame> f = _clip->__instance__->GetFrame(frame);
```

```
    const unsigned char* pixels = f->GetPixels();
    unsigned char* dest;
    strcpy(dest,pixels);
    int index = x * f->GetWidth() + y;
    dest[3*index] = char(res.R);
    dest[3*index + 1] = char(res.G);
    dest[3*index + 2] = char(res.B);
    const unsigned char* p = (const char*) dest;
    f->AddImage(f->GetWidth(), f->GetHeight(), 4, \
    QImage::Format_RGBA8888, p);
    //string color = "#" + int_to_hex(p.R) + int_to_hex(p.G) + \
     int_to_hex(p.B);
    //cout << color << endl;
    //f->AddColor(x,y,color);
}
*/

/* write clips to a file
   yo:prog:
   clip.alpha@1 = 255
   generate as:
   setProperty(clip,"alpha",1,255);

*/

void setProperty(tr1::shared_ptr<_Clip> _clip, \
string attname, int frame, double value){
    list<Clip*> lists = _clip->__instance__->Clips();
    if (attname == "alpha"){
        for (std::list<Clip*>::const_iterator iterator = \
        lists.begin(), end = lists.end(); iterator != end; ++iterator) {
            //double keytime = double(frame) / V_FPS + (*iterator)->Start();
            (*iterator)->alpha.AddPoint(frame + \
        (*iterator)->Start() * V_FPS,value);
            //std::cout << (*iterator)->Position() << " " \
         << (*iterator)->Start() <<" " << (*iterator)->End() << " " \
         << keytime <<" " << value << std::endl;
        }
    }if (attname == "location_x"){
        for (std::list<Clip*>::const_iterator iterator = \
         lists.begin(), end = lists.end(); iterator != end; ++iterator) {
            (*iterator)->location_x.AddPoint(frame +\
         (*iterator)->Start() * V_FPS,value);
        }
    }if (attname == "location_y"){
        for (std::list<Clip*>::const_iterator iterator = \
        lists.begin(), end = lists.end(); iterator != end; ++iterator) {
            (*iterator)->location_y.AddPoint(frame + \
        (*iterator)->Start() * V_FPS,value);
        }
    }if (attname == "scale_x"){
        for (std::list<Clip*>::const_iterator iterator = lists.begin(),\
         end = lists.end(); iterator != end; ++iterator) {
            (*iterator)->scale_x.AddPoint(frame + \
```

```
        (*iterator)->Start() * V_FPS,value);
        }
    }if (attname == "scale_y"){
        for (std::list<Clip*>::const_iterator iterator = lists.begin(),\
         end = lists.end(); iterator != end; ++iterator) {
            (*iterator)->scale_y.AddPoint(frame + \
        (*iterator)->Start() * V_FPS,value);
        }
    }if (attname == "rotate"){
        for (std::list<Clip*>::const_iterator iterator = lists.begin(),\
         end = lists.end(); iterator != end; ++iterator) {
            (*iterator)->rotation.AddPoint(frame + \
        (*iterator)->Start() * V_FPS,value);
        }
    }

    // add more..
}

template<typename T>
tr1::shared_ptr<vector<T>> \
slice_array(tr1::shared_ptr<vector<T>> \
 vec, int start, int end) {
    assert(end <= vec->size());
    tr1::shared_ptr<vector<T>> n_vec;
    while (start < end) {
        n_vec->push_back(*vec[start]);
        ++start;
    }
    return n_vec;
}


template<typename T>
tr1::shared_ptr<vector<T>> create_array()
{
   auto n_vec = tr1::shared_ptr<vector<T> >(new vector<T>());

    //for (auto e : elements)
    //  n_vec->push_back(e);
    return n_vec;
}



tr1::shared_ptr<Universal> DUMMY_SELF;

struct _log
{
    template <typename T>
    static void eval (tr1::shared_ptr<Universal> obj, T str) {
        std::cout << str;
    }
};
```

```
tr1::shared_ptr<_Clip> _Clip::eval(tr1::shared_ptr<Universal> \
obj, string fileName) {
        return createClip(fileName);
    }


struct _Clip_save : Universal {
    static void eval(tr1::shared_ptr<_Clip> _clip, string fileName) {
        writeClips(_clip, fileName);
    }
};

struct _Clip_log : Universal {
    static void eval(tr1::shared_ptr<_Clip> _clip) {
        std::cout << logClip(_clip) << std::endl;
    }

    static void eval(tr1::shared_ptr<_Clip> _clip, string fileName) {
        std::ofstream fout(fileName);
        fout << logClip(_clip);
        fout.close();
    }
};

template<typename T>
struct _Array_add {
    static tr1::shared_ptr<std::vector<T>> \
    eval(tr1::shared_ptr<std::vector<tr1::shared_ptr<T>>> \
     arr, tr1::shared_ptr<T> obj) {
        arr->push_back(obj);
        return arr;
    }

    static tr1::shared_ptr<std::vector<T>> \
    eval(tr1::shared_ptr<std::vector<T>> arr, T obj) {
        arr->push_back(obj);
        return arr;
    }
};

template<typename T>
struct _Array_length {
    static int length(tr1::shared_ptr<std::vector<tr1::shared_ptr<T>>> \
    arr) {
        return arr->size();
    }

    static int eval(tr1::shared_ptr<std::vector<T>> arr) {
        return arr->size();
    }
};
```

8.14 yoheader.yo

```
type Universal:{

}

type Int:{
    a: Int;
    func eval(a: Double) -> Int:{
        return;
}
    func eval(a: String) -> Int:{
        return;
}
    func eval(a:Int) -> Int:{
        return;
}
}
type String:{
    b: String;
    func eval(a: Int) -> String:{
        return;
}
    func eval(a: Double) -> String:{
        return;
}
    func eval(a: String) -> String:{
        return;
}
}
type Double:{
    c: Double;
    func eval(a: Int) -> Double:{
        return;
}
    func eval(a: String) -> Double:{
        return;
}
    func eval(a: Double) -> Double:{
        return;
}
}
type Bool:{
    func eval(a: Bool) -> Bool:{
        return;
}
}
type Void:{
    func eval(a: Void) -> Void:{
        return;
    }
}

type ArrayElementT:{
```

```
}

type Attribute:{
    second: Double;
    value: Double;
}
type Clip:{
    filename: String;
    starttime : Double;
    endtime : Double;
    position : Double;
    alpha : Attribute[];
    location_x : Attribute[];
    location_y : Attribute[];
    scale_x : Attribute[];
    scale_y : Attribute[];
    rotate : Attribute[];
    volume : Attribute[];
    clips:  Clip[];

    func eval(a: String) -> Clip: {
        return;
    }

    func eval(dir: String, ext: String) -> Clip[]: {
        return;
    }

    func save(self: Clip, a: String) -> Void: {
        return;
    }

    func log(self: Clip) -> Void: {
        return;
    }

    func log(self: Clip, a: String) -> Void: {
        return;
    }

}

type Pixel:{
    R: Int;
    G: Int;
    B: Int;
}

type Frame:{
    width: Int;
    height: Int;

}
```

```
func log(a: Int) -> Int:{
    return;
}

func log(a: Double) -> Double: {
    return;
}

func log(a: String) -> String: {
    return;
}

func log(a: Bool) -> Bool: {
    return;
}


func log(a: Clip) -> Clip: {
    return;
}

func log(a: Frame) -> Frame: {
    return;
}

func log(a: Attribute) -> Attribute: {
    return;
}


func __Add(a: Int, b: Int) -> Int:{
    return;
}

func __Add(a: Double, b: Double) -> Double:{
    return;
}

func __Add(a: String, b: String) -> String:{
    return;
}

func __Sub(a: Int, b: Int) -> Int:{
    return;
}

func __Sub(a: Double, b: Double) -> Double:{
    return;
}

func __Mult(a: Int, b: Int) -> Int:{
    return;
}
```

```
func __Mult(a: Double, b: Double) -> Double:{
    return;
}

func __Div(a: Int, b: Int) -> Int:{
    return;
}

func __Div(a: Double, b: Double) -> Double:{
    return;
}

func __Mod(a: Int, b: Int) -> Int:{
    return;
}

func __Equal(a: Int, b: Int) -> Bool:{
    return;
}

func __Equal(a: String, b: String) -> Bool:{
    return;
}

func __Equal(a: Double, b: Double) -> Bool:{
    return;
}

func __Equal(a: Bool, b: Bool) -> Bool:{
    return;
}


func __Neq(a: Int, b: Int) -> Bool:{
    return;
}

func __Neq(a: String, b: String) -> Bool:{
    return;
}

func __Neq(a: Double, b: Double) -> Bool:{
    return;
}

func __Neq(a: Bool, b: Bool) -> Bool:{
    return;
}

func __Less(a: Int, b: Int) -> Bool:{
    return;
}
```

```
func __Less(a: Double, b: Double) -> Bool:{
    return;
}

func __Leq(a: Int, b: Int) -> Bool:{
    return;
}

func __Leq(a: Double, b: Double) -> Bool:{
    return;
}

func __Geq(a: Int, b: Int) -> Bool:{
    return;
}

func __Geq(a: Double, b: Double) -> Bool:{
    return;
}

func __Gt(a: Int, b: Int) -> Bool:{
    return;
}

func __Gt(a: Double, b: Double) -> Bool:{
    return;
}

func __And(a: Bool, b: Bool) -> Bool:{
    return;
}

func __Or(a: Bool, b: Bool) -> Bool:{
    return;
}

type Array:{
    func length(self: Array) -> Int:{
        return;
    }

    func add(self: Array, ele: ArrayElementT) -> Array:{
        return;
    }
}
```