Yo: PLT Final Report

Mengqing Wang, Munan Cheng, Tiezheng Li, Yufei Ou

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

Contents

1 Introduction ........................................................................... 3
  1.1 Background .................................................................... 3
  1.2 Features ......................................................................... 3

2 Tutorial ............................................................................... 4
  2.1 Installation ...................................................................... 4
  2.2 Compiling and Running a Yo Program ............................... 4
  2.3 Yo examples .................................................................... 4

3 Language Reference Manual .................................................. 6
  3.1 Syntax Notations .............................................................. 6
  3.2 Lexical Conventions ......................................................... 6
    3.2.1 Comments ............................................................... 7
    3.2.2 Identifiers ............................................................... 7
    3.2.3 Reserved words ....................................................... 7
    3.2.4 Operators ............................................................... 7
    3.2.5 Separators ............................................................. 8
    3.2.6 New Line .............................................................. 8
    3.2.7 Whitespace ........................................................... 8
  3.3 Types .............................................................................. 8
    3.3.1 Built-in Types ....................................................... 8
    3.3.2 Clip type ............................................................... 10
    3.3.3 User Defined Type ................................................ 10
3.3.4 Type Constructor and Object Instantiation ............................................. 11
3.3.5 Array ..................................................................................................... 11
3.4 Expressions and Operators ......................................................................... 12
  3.4.1 Expressions .......................................................................................... 12
  3.4.2 Arithmetic operators ............................................................................ 12
  3.4.3 Array Access operators ......................................................................... 12
  3.4.4 Comparison operators .......................................................................... 13
  3.4.5 Logical operators ................................................................................ 13
  3.4.6 Assignment operators .......................................................................... 13
  3.4.7 Clip operations .................................................................................... 14
  3.4.8 Member Access operators ...................................................................... 15
  3.4.9 Operator Precedence and Associative Property ..................................... 15
3.5 Statements .................................................................................................. 15
  3.5.1 Overview .............................................................................................. 15
  3.5.2 log ......................................................................................................... 16
  3.5.3 id .......................................................................................................... 16
  3.5.4 while ..................................................................................................... 17
  3.5.5 for ......................................................................................................... 17
  3.5.6 continue ............................................................................................... 18
  3.5.7 break .................................................................................................... 18
  3.5.8 return .................................................................................................... 18
3.6 Functions ..................................................................................................... 18
  3.6.1 Function definition ............................................................................... 19
  3.6.2 Function calls ....................................................................................... 19
  3.6.3 Function Return Types ......................................................................... 19
  3.6.4 Recursion ............................................................................................. 20
  3.6.5 Built-in functions .................................................................................. 20
3.7 Program Structure and Scope .................................................................... 20
  3.7.1 Program Structure ............................................................................... 20
  3.7.2 Namespaces ......................................................................................... 20
  3.7.3 Scope .................................................................................................... 20

4 Project Plan .................................................................................................. 21
  4.1 Processes .................................................................................................. 21
    4.1.1 Planning ............................................................................................. 21
    4.1.2 Specification ...................................................................................... 21
    4.1.3 Development ..................................................................................... 21
    4.1.4 Testing ............................................................................................... 22
  4.2 Project Timeline ....................................................................................... 22
  4.3 Team Responsibility ................................................................................. 22
  4.4 Software Develop Environment ............................................................... 22

5 Architectural Design ..................................................................................... 23
  5.1 Preprocessor ............................................................................................. 23
  5.2 Scanner .................................................................................................... 23
  5.3 Parsed ...................................................................................................... 23
  5.4 TypeReader .............................................................................................. 24
  5.5 SAST ........................................................................................................ 24
  5.6 Code generation ........................................................................................ 24
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 libuinitittest+-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 compile 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)
```

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

```python
# 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
    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.

```python
# 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:")
```
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** * 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, b[a23]b will find baab and baaab but not bab or baaaaab. 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.

```plaintext
A ::= 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:

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

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

```plaintext
#( 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 reserved word (see next subsection). It starts with a letter or an underscore, optionally followed by a series of characters (letter, underscore, number). The length varies from 1 to 256.

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

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

Legal examples:

```plaintext
_number _number1 number2 number_3 Number
```

Illegal examples:

```plaintext
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.

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

Table 1: List of keywords in Yo

```plaintext
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 indentation 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:
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.

\[0 - 5 \# \text{i.e., } -5\]

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

\[
\text{DoubleLiteral ::= } [0-9]*.[0-9]+\]

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`.

\[
\text{BoolLiteral ::= true | false}\]

- **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.

\[
\text{StringLiteral ::= } "\text{StringCharacter}\"
\]

\[
\text{StringCharacter ::= } [^"\'\] \text{StringCharacter}
\quad \mid [^"\']
\quad \mid \text{EscapeSequence StringCharacter}
\quad \mid \text{EscapeSequence}
\]

\[
\text{EscapeSequence ::= } \backslash b \mid \backslash t \mid \backslash n \mid \backslash r \mid \" \mid \' \mid \\n
\text{Table 3: Escape Characters}
\]

<table>
<thead>
<tr>
<th>EscapeSequence</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>Backspace</td>
</tr>
<tr>
<td>\t</td>
<td>Horizontal tab</td>
</tr>
<tr>
<td>\n</td>
<td>New line</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage return</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double quote</td>
</tr>
<tr>
<td>'</td>
<td>Single quote</td>
</tr>
<tr>
<td>\</td>
<td>Backslash</td>
</tr>
</tbody>
</table>

Examples:

"abc" "9j32 f0kca0" "Hello\nYo!"
<table>
<thead>
<tr>
<th>MediaType</th>
<th>Extension Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Picture</td>
<td>jpg, jpeg, png, bmp, gif</td>
</tr>
<tr>
<td>Video</td>
<td>webm, f4v, mov, rmvb, mp4, rm, wmv, avi, flv, 3gp, mkv</td>
</tr>
</tbody>
</table>

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
```

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 initialization 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.

```plaintext
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,

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

Here is an example of object instantiation:

```plaintext
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.

```plaintext
anArrayForTypeT: T[]
```

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

```plaintext
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,

```plaintext
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 ::= Literals
| 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   | # subtraction
| 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
### 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.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Type 1</th>
<th>Type 2</th>
<th>Return type</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Int</td>
<td>Int</td>
<td>Int</td>
</tr>
<tr>
<td>-</td>
<td>Int</td>
<td>Int</td>
<td>Int</td>
</tr>
<tr>
<td>*</td>
<td>Int</td>
<td>Int</td>
<td>Int</td>
</tr>
<tr>
<td>/</td>
<td>Int</td>
<td>Int</td>
<td>Int</td>
</tr>
<tr>
<td>%</td>
<td>Int</td>
<td>Int</td>
<td>Int</td>
</tr>
<tr>
<td>==</td>
<td>Bool</td>
<td>Bool</td>
<td>Bool</td>
</tr>
<tr>
<td>!=</td>
<td>Int</td>
<td>Int</td>
<td>Bool</td>
</tr>
<tr>
<td>&gt;</td>
<td>Int</td>
<td>Int</td>
<td>Bool</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Int</td>
<td>Int</td>
<td>Bool</td>
</tr>
<tr>
<td>&lt;</td>
<td>Int</td>
<td>Int</td>
<td>Bool</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Int</td>
<td>Int</td>
<td>Bool</td>
</tr>
<tr>
<td>&lt; &lt; = &gt; =, &gt;</td>
<td>Int</td>
<td>Int</td>
<td>Bool</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Bool</td>
<td>Bool</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bool</td>
</tr>
</tbody>
</table>

Table 5: Basic operators

```plaintext
Comparison operators

- `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.

```plaintext
Logical operators

- `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.

```plaintext
Assignment operators

- `a = 42`
- `b = a`
```
3.4.7 Clip operations

Yo allows two clips to be concatenated easily.

```plaintext
# 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).

```plaintext
# 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, `^` to denote putting `b` on top of `a`, by an offset of `c` seconds if it is a Double or of `c` frames if it is a Int.

```plaintext
# 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.

```plaintext
# 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.

```plaintext
# p is a pixel from clip1 at time 2.4s, coordinate(20,30)
```
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.
7. Logical OR expressions.
8. Clip concatenation.
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 \hspace{1em} NEWLINE
       | log_stmt
       | return_stmt
       | continue_stmt
       | break_stmt

compound_stmt ::= if_stmt
```
while_stmt
| for_stmt
suite ::= \texttt{NEWLINE INDENT} statement+ \texttt{DEDENT}
statement ::= stmt \texttt{NEWLINE} | compound_stmt

3.5.2 \textbf{log}

\texttt{log_stmt ::= log (expr)}

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

\begin{verbatim}
log ("A string")
# Output: A string
a = 1
log (a)
# Output: 1
b = 1.01
log (b)
# Output: 1.01
\end{verbatim}

3.5.3 \textbf{if}

The \texttt{if} statement is used for conditional execution:

\begin{verbatim}
if_stmt ::= \texttt{if} expr : suite
(\texttt{elif} expr : suite)*
[\texttt{else} : suite]
\end{verbatim}

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

\begin{verbatim}
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
\end{verbatim}
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

```python
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

```python
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

```python
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.

```python
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:

```plaintext
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

```plaintext
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:

```plaintext
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

```plaintext
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:

```plaintext
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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>log</td>
<td>Int, Double, String, Bool, Clip</td>
</tr>
</tbody>
</table>

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.

```plaintext
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:

```python
def 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 Overleaf⁴. 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 codeGeneration. Tiezheng did almost all the other things such as wrote Makefile, test suites, main function and dealt 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

---

⁴https://www.overleaf.com/
Table 7: Project timeline

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

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>
<thead>
<tr>
<th>Table 8: Team responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>5 Architectural Design</strong></td>
</tr>
<tr>
<td><strong>5.1 Preprocessor</strong></td>
</tr>
<tr>
<td>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 semicolons to mark the end of statements, except for continue line separator ''. Lines ended with ' are 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.</td>
</tr>
</tbody>
</table>

**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.
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 re-constructed 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 ˆ 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:

```
  ▼ test
      ▶ codegen
      ▶ intermediate
      ▶ movie
      ▶ parser
      ▶ preprocessor
      ▶ semantic
      ▶ system
      ▶ typereader
```

6.2 Examples and Results

Here we give an example of system testing:
```ocaml
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-readfolders... 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 HelloWord 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

<table>
<thead>
<tr>
<th>Detail</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b340046</td>
<td>LeeTZ</td>
<td>Initial commit</td>
</tr>
<tr>
<td>113b3c6</td>
<td>LeeTZ</td>
<td>add proposal</td>
</tr>
<tr>
<td>898b0bc</td>
<td>LeeTZ</td>
<td>add lrm</td>
</tr>
<tr>
<td>e4829c</td>
<td>LeeTZ</td>
<td>framework of lrm</td>
</tr>
<tr>
<td>a79179b</td>
<td>LeeTZ</td>
<td>add to framework of LRM</td>
</tr>
<tr>
<td>1a4a375</td>
<td>Yufei Ou</td>
<td>add something for test</td>
</tr>
<tr>
<td>h66193</td>
<td>Edward Cheng</td>
<td>add gitignore to doc</td>
</tr>
<tr>
<td>Commit</td>
<td>Author</td>
<td>Message</td>
</tr>
<tr>
<td>--------</td>
<td>--------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>9a7aca5</td>
<td>mengqwang</td>
<td>bug fix for statement</td>
</tr>
<tr>
<td>c9152c4</td>
<td>Edward Cheng</td>
<td>start semantic and test</td>
</tr>
<tr>
<td>a5152f2</td>
<td>Edward Cheng</td>
<td>merged with master</td>
</tr>
<tr>
<td>c66c530</td>
<td>Edward Cheng</td>
<td>refactoring the type resolution; add new _var</td>
</tr>
<tr>
<td>11c833a</td>
<td>mengqwang</td>
<td>string_of_expr</td>
</tr>
<tr>
<td>8367138</td>
<td>mengqwang</td>
<td>string_of_expr</td>
</tr>
<tr>
<td>4e1a096</td>
<td>Edward Cheng</td>
<td>ready for merge</td>
</tr>
<tr>
<td>00816d9</td>
<td>Edward Cheng</td>
<td>merged with update in ast</td>
</tr>
<tr>
<td>568411f</td>
<td>Edward Cheng</td>
<td>removed primary_expr</td>
</tr>
<tr>
<td>8400a39</td>
<td>Edward Cheng</td>
<td>revised parser and sat</td>
</tr>
<tr>
<td>e60529e</td>
<td>Yufei Ou</td>
<td>add output.ml</td>
</tr>
<tr>
<td>b1f1c15</td>
<td>mengqwang</td>
<td>read types in first pass</td>
</tr>
<tr>
<td>237076e</td>
<td>mengqwang</td>
<td>read types in the first pass</td>
</tr>
<tr>
<td>6c0e2aa</td>
<td>Edward Cheng</td>
<td>create sast</td>
</tr>
<tr>
<td>04a8223</td>
<td>Edward Cheng</td>
<td>add stmnt_semantic</td>
</tr>
<tr>
<td>9916c1b</td>
<td>Yufei Ou</td>
<td>add output again</td>
</tr>
<tr>
<td>333f2d1</td>
<td>mengqwang</td>
<td>type reader</td>
</tr>
<tr>
<td>55ad2ec</td>
<td>mengqwang</td>
<td>type read</td>
</tr>
<tr>
<td>57d3d74</td>
<td>Edward Cheng</td>
<td>fix bugs in semantic</td>
</tr>
<tr>
<td>6688630</td>
<td>Edward Cheng</td>
<td>merged</td>
</tr>
<tr>
<td>8ac77c4</td>
<td>Edward Cheng</td>
<td>update test semantic</td>
</tr>
<tr>
<td>1b61362</td>
<td>LeeTZ</td>
<td>add some parser_test</td>
</tr>
<tr>
<td>8b8636e</td>
<td>Yufei Ou</td>
<td>add generate.ml</td>
</tr>
<tr>
<td>6b725df</td>
<td>Yufei Ou</td>
<td>add generate.ml</td>
</tr>
<tr>
<td>95dc15</td>
<td>Yufei Ou</td>
<td>Merge branch 'yo'</td>
</tr>
<tr>
<td>3aa5c6</td>
<td>mengqwang</td>
<td>2nd pass</td>
</tr>
<tr>
<td>13addb</td>
<td>mengqwang</td>
<td>2nd pass</td>
</tr>
<tr>
<td>68ed600</td>
<td>mengqwang</td>
<td>2nd pass type</td>
</tr>
<tr>
<td>973f1e1</td>
<td>mengqwang</td>
<td>readme</td>
</tr>
<tr>
<td>9a3334</td>
<td>mengqwang</td>
<td>readme</td>
</tr>
<tr>
<td>209ec5e</td>
<td>mengqwang</td>
<td>readme</td>
</tr>
<tr>
<td>91b3b55</td>
<td>mengqwang</td>
<td>readme</td>
</tr>
<tr>
<td>dc65172</td>
<td>mengqwang</td>
<td>readme</td>
</tr>
<tr>
<td>443a93f</td>
<td>LeeTZ</td>
<td>continue some parser test</td>
</tr>
<tr>
<td>8aadbac</td>
<td>LeeTZ</td>
<td>remove output for Bro Fei</td>
</tr>
<tr>
<td>ad7f2fe</td>
<td>LeeTZ</td>
<td>modified parser and ast print func</td>
</tr>
<tr>
<td>115b4af</td>
<td>LeeTZ</td>
<td>lots of debug on ast and parser</td>
</tr>
<tr>
<td>14168a4</td>
<td>LeeTZ</td>
<td>modify sat</td>
</tr>
<tr>
<td>6cd9098</td>
<td>Yufei Ou</td>
<td>update generate.ml</td>
</tr>
<tr>
<td>7ebee8d</td>
<td>LeeTZ</td>
<td>finish ast and parser syntax debug</td>
</tr>
<tr>
<td>a121478</td>
<td>LeeTZ</td>
<td>Merge branch 'master' of github.com:LeeTZ/Yo</td>
</tr>
<tr>
<td>97b44d</td>
<td>LeeTZ</td>
<td>let s change it to semi</td>
</tr>
<tr>
<td>628b194</td>
<td>mengqwang</td>
<td>typereader exception</td>
</tr>
<tr>
<td>3867ad0</td>
<td>LeeTZ</td>
<td>finish testing of parser!</td>
</tr>
<tr>
<td>0b74a95</td>
<td>LeeTZ</td>
<td>Merge branch 'master' of github.com:LeeTZ/Yo</td>
</tr>
<tr>
<td>263d0aa</td>
<td>mengqwang</td>
<td>bug plz help</td>
</tr>
<tr>
<td>Commit</td>
<td>Author</td>
<td>Message</td>
</tr>
<tr>
<td>---------</td>
<td>----------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0495aaa</td>
<td>Edward Cheng</td>
<td>Merge branch 'master' of <a href="https://github.com/LeeTZ/Yo">https://github.com/LeeTZ/Yo</a></td>
</tr>
<tr>
<td>a4e2836</td>
<td>Edward Cheng</td>
<td>add more types to log</td>
</tr>
<tr>
<td>29a4a10</td>
<td>Yufei Ou</td>
<td>change op</td>
</tr>
<tr>
<td>8c4c16d</td>
<td>Yufei Ou</td>
<td>Merge branch 'master' of <a href="https://github.com/LeeTZ/Yo">https://github.com/LeeTZ/Yo</a></td>
</tr>
<tr>
<td>c832a40</td>
<td>Yufei Ou</td>
<td>change while</td>
</tr>
<tr>
<td>2b71a1c</td>
<td>LeeTZ</td>
<td>we successfully build helloworld</td>
</tr>
<tr>
<td>303c790</td>
<td>Yufei Ou</td>
<td>add yolib.h</td>
</tr>
<tr>
<td>9ce0909</td>
<td>Yufei Ou</td>
<td>Merge branch 'master' of <a href="https://github.com/LeeTZ/Yo">https://github.com/LeeTZ/Yo</a></td>
</tr>
<tr>
<td>e372d63</td>
<td>Yufei Ou</td>
<td>add yolib.h</td>
</tr>
<tr>
<td>8c6c10b</td>
<td>LeeTZ</td>
<td>add fullstach test!</td>
</tr>
<tr>
<td>a634e83</td>
<td>LeeTZ</td>
<td>modify gitignore and clean it</td>
</tr>
<tr>
<td>6169d8e</td>
<td>LeeTZ</td>
<td>add test</td>
</tr>
<tr>
<td>ab0f77f</td>
<td>LeeTZ</td>
<td>add preprocess test and parser test</td>
</tr>
<tr>
<td>2a9574f</td>
<td>LeeTZ</td>
<td>renew test shell</td>
</tr>
<tr>
<td>330c816</td>
<td>LeeTZ</td>
<td>clean</td>
</tr>
<tr>
<td>42eca40</td>
<td>mengqwang</td>
<td>bug fix</td>
</tr>
<tr>
<td>6b0b6b1</td>
<td>mengqwang</td>
<td>typereader bug fix</td>
</tr>
<tr>
<td>1661d76</td>
<td>mengqwang</td>
<td>bug fix typereader</td>
</tr>
<tr>
<td>7e9a08a</td>
<td>mengqwang</td>
<td>makefile bug</td>
</tr>
<tr>
<td>651596b</td>
<td>LeeTZ</td>
<td>add util</td>
</tr>
<tr>
<td>36f7e3d</td>
<td>Edward Cheng</td>
<td>fix makefile dependency</td>
</tr>
<tr>
<td>49211d</td>
<td>Yufei Ou</td>
<td>add HAT and AT</td>
</tr>
<tr>
<td>b6f5d64</td>
<td>Edward Cheng</td>
<td>add new array</td>
</tr>
<tr>
<td>600566</td>
<td>Edward Cheng</td>
<td>add array literal type checking</td>
</tr>
<tr>
<td>de7293b</td>
<td>Edward Cheng</td>
<td>eval_entry ret not None; op rename; Binop semantic analysis</td>
</tr>
<tr>
<td>f6818d</td>
<td>Edward Cheng</td>
<td>add ForIn semantic</td>
</tr>
<tr>
<td>2017316</td>
<td>Edward Cheng</td>
<td>modify ForEq to ForRange</td>
</tr>
<tr>
<td>048567</td>
<td>Edward Cheng</td>
<td>add return type to function signature</td>
</tr>
<tr>
<td>8bb142c</td>
<td>Edward Cheng</td>
<td>add ArrayIndex semantic analysis; remove Noexpr</td>
</tr>
<tr>
<td>9193c42</td>
<td>LeeTZ</td>
<td>add openshot example</td>
</tr>
<tr>
<td>8e4196c</td>
<td>Edward Cheng</td>
<td>modify array type in ast, sast, built_in_type</td>
</tr>
<tr>
<td>3534d0e</td>
<td>mengqwang</td>
<td>Merge branch 'typename' of <a href="https://github.com/LeeTZ/Yo">https://github.com/LeeTZ/Yo</a></td>
</tr>
<tr>
<td>6100e6c</td>
<td>LeeTZ</td>
<td>add definition of clip</td>
</tr>
<tr>
<td>0485738</td>
<td>LeeTZ</td>
<td>Merge branch 'master' of github.com:LeeTZ/Yo</td>
</tr>
<tr>
<td>eb97c8c</td>
<td>Edward Cheng</td>
<td>make array type recursive</td>
</tr>
<tr>
<td>8c4526c</td>
<td>Edward Cheng</td>
<td>separate array_constructor</td>
</tr>
<tr>
<td>6500865</td>
<td>Edward Cheng</td>
<td>update all type related semantic analysis</td>
</tr>
<tr>
<td>6541309</td>
<td>Edward Cheng</td>
<td>Merge with branch typename</td>
</tr>
<tr>
<td>94c390c</td>
<td>LeeTZ</td>
<td>clean</td>
</tr>
<tr>
<td>1b9ba1c</td>
<td>Yufei Ou</td>
<td>forange</td>
</tr>
<tr>
<td>c420360</td>
<td>Mengqing Wang</td>
<td>array resolve</td>
</tr>
<tr>
<td>1293271</td>
<td>Mengqing Wang</td>
<td>delete mengqingtest</td>
</tr>
<tr>
<td>fa2a0f</td>
<td>Mengqing Wang</td>
<td>type test</td>
</tr>
<tr>
<td>e29c91d</td>
<td>Yufei Ou</td>
<td>move tabs to sast</td>
</tr>
<tr>
<td>14be3ac</td>
<td>Yufei Ou</td>
<td>Merge branch 'master' of <a href="https://github.com/LeeTZ/Yo">https://github.com/LeeTZ/Yo</a></td>
</tr>
<tr>
<td>07be179</td>
<td>Yufei Ou</td>
<td>add test executable to gitignore</td>
</tr>
<tr>
<td>Commit</td>
<td>Author</td>
<td>Message</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>87a5d42</td>
<td>Yufei Ou</td>
<td>work from another computer</td>
</tr>
<tr>
<td>10a20d</td>
<td>Edward Cheng</td>
<td>reverse entry move</td>
</tr>
<tr>
<td>42da383</td>
<td>LeeTZ</td>
<td>add clip, frame libs</td>
</tr>
<tr>
<td>64a3ac3</td>
<td>LeeTZ</td>
<td>Merge branch ‘master’ of github.com:LeeTZ/Yo</td>
</tr>
<tr>
<td>63cc20</td>
<td>Mengqing Wang</td>
<td>type test</td>
</tr>
<tr>
<td>7cb0d7</td>
<td>Edward Cheng</td>
<td>generate almost finished</td>
</tr>
<tr>
<td>1a30bb3</td>
<td>Mengqing Wang</td>
<td>array resolve</td>
</tr>
<tr>
<td>97b2c9a</td>
<td>Mengqing Wang</td>
<td>print type</td>
</tr>
<tr>
<td>31a81d4</td>
<td>Mengqing Wang</td>
<td>print type</td>
</tr>
<tr>
<td>3be32c6</td>
<td>Mengqing Wang</td>
<td>my new typerheader</td>
</tr>
<tr>
<td>3d1a559</td>
<td>Edward Cheng</td>
<td>codegen finished</td>
</tr>
<tr>
<td>dd13d2</td>
<td>Edward Cheng</td>
<td>merged</td>
</tr>
<tr>
<td>cb16e9</td>
<td>LeeTZ</td>
<td>fix type_entry comparson</td>
</tr>
<tr>
<td>48da81f</td>
<td>LeeTZ</td>
<td>merged</td>
</tr>
<tr>
<td>652cd4a</td>
<td>LeeTZ</td>
<td>big move</td>
</tr>
<tr>
<td>f4938a4</td>
<td>Edward Cheng</td>
<td>add array operation to generate</td>
</tr>
<tr>
<td>1bce32e</td>
<td>LeeTZ</td>
<td>finish dream function of clipindex and cliprange</td>
</tr>
<tr>
<td>e30200e</td>
<td>LeeTZ</td>
<td>add testplan</td>
</tr>
<tr>
<td>7b8a5c5</td>
<td>LeeTZ</td>
<td>clean scanner test</td>
</tr>
<tr>
<td>6e1be89</td>
<td>LeeTZ</td>
<td>clean it</td>
</tr>
<tr>
<td>5a8b954</td>
<td>LeeTZ</td>
<td>renew fullstack testsuits</td>
</tr>
<tr>
<td>8513a63</td>
<td>LeeTZ</td>
<td>renew absolutely</td>
</tr>
<tr>
<td>768ba05</td>
<td>LeeTZ</td>
<td>add logclip for testing</td>
</tr>
<tr>
<td>9bde66a</td>
<td>LeeTZ</td>
<td>remove main function in libclip, it is part of lib now</td>
</tr>
<tr>
<td>924c11c</td>
<td>Edward Cheng</td>
<td>add set attribute</td>
</tr>
<tr>
<td>5a89b35</td>
<td>Edward Cheng</td>
<td>Merge branch ‘master’ of <a href="https://github.com/LeeTZ/Yo">https://github.com/LeeTZ/Yo</a></td>
</tr>
<tr>
<td>c002085</td>
<td>Mengqing Wang</td>
<td>type redefine exception</td>
</tr>
<tr>
<td>0b49a38</td>
<td>Mengqing Wang</td>
<td>type redefine exception</td>
</tr>
<tr>
<td>592dc28</td>
<td>mengqwang2</td>
<td>typereader test</td>
</tr>
<tr>
<td>8b561b6</td>
<td>mengqwang2</td>
<td>test sh</td>
</tr>
<tr>
<td>6a46b32</td>
<td>Edward Cheng</td>
<td>add static to function generation</td>
</tr>
<tr>
<td>ade4f3d</td>
<td>Edward Cheng</td>
<td>merged</td>
</tr>
<tr>
<td>7a08f8</td>
<td>mengqwang2</td>
<td>typereader bug fix</td>
</tr>
<tr>
<td>0b88e34</td>
<td>mengqwang2</td>
<td>bug fix</td>
</tr>
<tr>
<td>6c01b63</td>
<td>mengqwang2</td>
<td>bug fix</td>
</tr>
<tr>
<td>577d6cd</td>
<td>LeeTZ</td>
<td>fix lib and generate</td>
</tr>
<tr>
<td>29b6e90</td>
<td>Edward Cheng</td>
<td>modify youlib</td>
</tr>
<tr>
<td>0b1a0e6</td>
<td>Edward Cheng</td>
<td>Merge branch ‘master’ of <a href="https://github.com/LeeTZ/Yo">https://github.com/LeeTZ/Yo</a></td>
</tr>
<tr>
<td>a34c21d</td>
<td>LeeTZ</td>
<td>munan test suits</td>
</tr>
<tr>
<td>91a5303</td>
<td>LeeTZ</td>
<td>Merge branch ‘master’ of github.com:LeeTZ/Yo</td>
</tr>
<tr>
<td>8c6a28c</td>
<td>Edward Cheng</td>
<td>memory bug fix</td>
</tr>
<tr>
<td>c510b8a</td>
<td>LeeTZ</td>
<td>compiled with libopenshot!</td>
</tr>
<tr>
<td>839e0b9</td>
<td>mengqwang2</td>
<td>typereader eval</td>
</tr>
<tr>
<td>1da7971</td>
<td>Edward Cheng</td>
<td>add forward; declaration; fix naming convention</td>
</tr>
<tr>
<td>1bb1c03</td>
<td>Edward Cheng</td>
<td>Merge branch ‘master’ of <a href="https://github.com/LeeTZ/Yo">https://github.com/LeeTZ/Yo</a></td>
</tr>
<tr>
<td>451a257</td>
<td>mengqwang2</td>
<td>t_actual</td>
</tr>
<tr>
<td>61dce37</td>
<td>Edward Cheng</td>
<td>fix type name convention</td>
</tr>
<tr>
<td>15ff013</td>
<td>Edward Cheng</td>
<td>Merge branch 'master' of <a href="https://github.com/LeeTZ/Yo">https://github.com/LeeTZ/Yo</a></td>
</tr>
<tr>
<td>867109d</td>
<td>Edward Cheng</td>
<td>fix name convention in generate</td>
</tr>
<tr>
<td>8b231c3</td>
<td>Edward Cheng</td>
<td>Clip creation works</td>
</tr>
<tr>
<td>41353aa</td>
<td>mengqiang2</td>
<td>parser test</td>
</tr>
<tr>
<td>44e2201</td>
<td>mengqiang2</td>
<td>tr test</td>
</tr>
<tr>
<td>26aad6c</td>
<td>LeeTZ</td>
<td>modify test.sh to log error in file</td>
</tr>
<tr>
<td>6362285</td>
<td>Edward Cheng</td>
<td>use base_type instead of string for function name in SCall</td>
</tr>
<tr>
<td>3b1518a</td>
<td>Edward Cheng</td>
<td>merged test.sh</td>
</tr>
<tr>
<td>2a119a6</td>
<td>Edward Cheng</td>
<td>add clip read/write</td>
</tr>
<tr>
<td>4a7e55d</td>
<td>Edward Cheng</td>
<td>use auto</td>
</tr>
<tr>
<td>64a203b</td>
<td>LeeTZ</td>
<td>add a new bash to run prog</td>
</tr>
<tr>
<td>03c3888</td>
<td>LeeTZ</td>
<td>Merge branch 'master' of github.com:LeeTZ/Yo</td>
</tr>
<tr>
<td>1743392</td>
<td>Edward Cheng</td>
<td>fix video attribute editing</td>
</tr>
<tr>
<td>8d3217b</td>
<td>Edward Cheng</td>
<td>Merge branch 'master' of <a href="https://github.com/LeeTZ/Yo">https://github.com/LeeTZ/Yo</a></td>
</tr>
<tr>
<td>b5dce22</td>
<td>LeeTZ</td>
<td>add new bin</td>
</tr>
<tr>
<td>8d38b4d</td>
<td>LeeTZ</td>
<td>add a new disto</td>
</tr>
<tr>
<td>2a6744b</td>
<td>Edward Cheng</td>
<td>add wrapper to clip</td>
</tr>
<tr>
<td>3459437</td>
<td>Edward Cheng</td>
<td>Merge branch 'master' of <a href="https://github.com/LeeTZ/Yo">https://github.com/LeeTZ/Yo</a></td>
</tr>
<tr>
<td>5df69ae</td>
<td>mengqiang2</td>
<td>test</td>
</tr>
<tr>
<td>5199b4f</td>
<td>LeeTZ</td>
<td>debug yolib and finish rename!</td>
</tr>
<tr>
<td>9769787</td>
<td>LeeTZ</td>
<td>Merge branch 'master' of github.com:LeeTZ/Yo</td>
</tr>
<tr>
<td>05308b</td>
<td>LeeTZ</td>
<td>finish a fadeout effect</td>
</tr>
<tr>
<td>6f831c0</td>
<td>LeeTZ</td>
<td>add logclip</td>
</tr>
<tr>
<td>7be4673</td>
<td>Edward Cheng</td>
<td>add log to clip</td>
</tr>
<tr>
<td>b6d82c0</td>
<td>LeeTZ</td>
<td>debug cliprange</td>
</tr>
<tr>
<td>c1737b1</td>
<td>LeeTZ</td>
<td>Merge branch 'master' of github.com:LeeTZ/Yo</td>
</tr>
<tr>
<td>c63f3c5</td>
<td>Edward Cheng</td>
<td>add void to basic type in codegen; add clip func test</td>
</tr>
<tr>
<td>2f86e2c</td>
<td>Edward Cheng</td>
<td>Merge branch 'master' of <a href="https://github.com/LeeTZ/Yo">https://github.com/LeeTZ/Yo</a></td>
</tr>
<tr>
<td>ad86c81</td>
<td>LeeTZ</td>
<td>debug for alpha value</td>
</tr>
<tr>
<td>55f6c1b</td>
<td>LeeTZ</td>
<td>Merge branch 'master' of github.com:LeeTZ/Yo</td>
</tr>
<tr>
<td>8dd138c</td>
<td>Edward Cheng</td>
<td>fix another compare_type bug in semantic</td>
</tr>
<tr>
<td>60f5804</td>
<td>Edward Cheng</td>
<td>Merge branch 'master' of <a href="https://github.com/LeeTZ/Yo">https://github.com/LeeTZ/Yo</a></td>
</tr>
<tr>
<td>49278a1</td>
<td>LeeTZ</td>
<td>debugging layerclip</td>
</tr>
<tr>
<td>81183d</td>
<td>LeeTZ</td>
<td>Merge branch 'master' of github.com:LeeTZ/Yo</td>
</tr>
<tr>
<td>3da32a</td>
<td>LeeTZ</td>
<td>clean output</td>
</tr>
<tr>
<td>21b8f4c</td>
<td>Edward Cheng</td>
<td>rename ClipConcat to ClipCascade; add ClipConcat</td>
</tr>
<tr>
<td>180e32f</td>
<td>Edward Cheng</td>
<td>Merge branch 'master' of <a href="https://github.com/LeeTZ/Yo">https://github.com/LeeTZ/Yo</a></td>
</tr>
<tr>
<td>6370556</td>
<td>LeeTZ</td>
<td>add move in x array</td>
</tr>
<tr>
<td>c64907e</td>
<td>LeeTZ</td>
<td>Merge branch 'master' of github.com:LeeTZ/Yo</td>
</tr>
<tr>
<td>3e2a666</td>
<td>Edward Cheng</td>
<td>add video concat test</td>
</tr>
<tr>
<td>2c06466</td>
<td>Edward Cheng</td>
<td>Merge branch 'master' of <a href="https://github.com/LeeTZ/Yo">https://github.com/LeeTZ/Yo</a></td>
</tr>
<tr>
<td>b30e1b</td>
<td>Edward Cheng</td>
<td>change Void type name</td>
</tr>
<tr>
<td>a3e480b</td>
<td>LeeTZ</td>
<td>a very guiyi de error</td>
</tr>
<tr>
<td>3d1e277</td>
<td>LeeTZ</td>
<td>Merge branch 'master' of github.com:LeeTZ/Yo</td>
</tr>
<tr>
<td>6f8d5f3</td>
<td>mengqiang2</td>
<td>seman test</td>
</tr>
</tbody>
</table>
Edward Cheng     add eval init
Edward Cheng     fix pointer
LeeTZ             hack for pixel
LeeTZ             finish all properties to set
Edward Cheng     add pixel
Edward Cheng     Merge branch 'master' of https://github.com/LeeTZ/Yo
LeeTZ             modify test cases
Edward Cheng     change return type to pixel
Edward Cheng     Merge branch 'master' of https://github.com/LeeTZ/Yo
Edward Cheng     fix log name in yolib.h
LeeTZ             hack
LeeTZ             Merge branch 'master' of github.com:LeeTZ/Yo
LeeTZ             debugging pixels
LeeTZ             set yo.sh again, delete intermediate files
LeeTZ             add new test cases
LeeTZ             clean
LeeTZ             add keyframe tests
Edward Cheng     add array ops
Edward Cheng     Merge branch 'master' of https://github.com/LeeTZ/Yo
Edward Cheng     fix array add return type in cpp
LeeTZ             fix typename
Edward Cheng     add Clip||(DIR_STRING)
Edward Cheng     Merge branch 'master' of https://github.com/LeeTZ/Yo
Edward Cheng     fix function eval
LeeTZ             add mengqing test
LeeTZ             fix array add
Edward Cheng     fix function eval signature
Edward Cheng     Merge branch 'master' of https://github.com/LeeTZ/Yo
LeeTZ             gcd test
LeeTZ             what
Edward Cheng     fix array indexing
Edward Cheng     Merge branch 'master' of https://github.com/LeeTZ/Yo
Edward Cheng     add webm to gitignore
Edward Cheng     exclude array length in forward decl
LeeTZ             a
LeeTZ             add test cases
Edward Cheng     add array ops
LeeTZ             add test cases
Edward Cheng     Merge branch 'master' of https://github.com/LeeTZ/Yo
Edward Cheng     correct array func name
LeeTZ             help munan debugging with array
LeeTZ             array finish
Edward Cheng     add log bool
Edward Cheng     merge
Edward Cheng     add array literal back
LeeTZ             finish all tests
LeeTZ             Merge branch 'master' of github.com:LeeTZ/Yo
8.1 preprocessor.py

```python
#!/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 = "}" + 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] == ‘:\n’:
    linejoin = True
    lineout = lineout[:i]

else:
    lineout = lineout + ‘;\n’
    linejoin = False

    output.write(lineout)

    while 0 < stack[-1]:
        output.write("') + lineout"
        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 *)
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 RBRACKET
%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) }
dot_expr:
  | 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 }

43
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)
     " [" ^ (String.sub s 2 ((String.length s) - 2)) ^ "]")
    (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 "", ")")

45
```ocaml
(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)
```

47


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"
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) ->

49
exists_types typetab tail (arrcnt+1) at

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; 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 newid e)
    tt
    type_element

in

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

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
let first_pass tt program =  
  let listpass = List.fold_left (fun tt e ->  
    walkDecl_1 tt e) tt program  
  in  
  listpass  
and second_pass tt program =  
  let listpass = List.fold_left (fun tt e ->  
    walkDecl_2 tt e) tt program  
  in  
  listpass  
and third_pass tt =  
  let listpass =  
    (NameMap.iter  
      (fun k v -> addTypeeval k v) tt tt) in  
  listpass  
in let t = context.typtab in  
let t = first_pass t program in  
let t = second_pass t program in  
let t = third_pass t in  
{vsymtab=[NameMap.empty]; typtab=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 =  
      (* Some type generating code here *)  

let t = context.typtab in  
let t = first_pass t program in  
let t = second_pass t program in  
let t = third_pass t in  
{vsymtab=[NameMap.empty]; typtab=t}
begin
  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 |
    (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 used 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})
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)

let smain = build_expr_semantic ctx main and sidx = build_expr_semantic ctx idx
(let main_type = (extract_semantic smain).type_def in
  match main_type with
  | BaseTypeEntry t ->
    (if compare_type main_type clip_type then SClipTimeIndex(smain, sidx, {actions= []; type_def=frame_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 (" has to be of either Double or Int type"))
    else raise (SemanticError (" has to be of Array 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"))) |
| ArrayTypeError _ | -> |
| (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

56
| 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,
| ClipCascade (c11, c12, tm) ->
| let s11 =
| buildExprSemantic ctx c11 and s12 =
| buildExprSemantic ctx c12 and stm =
| buildExprSemantic ctx tm in
|   if compareType (extractSemantic s11).type_def clip_type
|     && compareType (extractSemantic s12).type_def clip_type
|     then (if compareType
|       (extractSemantic stm).type_def double_type
|       then SClipTimeCascade(s11, s12, stm, {actions=[]};
|         type_def=clip_type})
|     else if compareType (extractSemantic stm).type_def int_type
|       then SClipFrameCascade(s11, s12, 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 (c11, c12) ->
| let s11 =
| buildExprSemantic ctx c11 and s12 =
| buildExprSemantic ctx c12 in
|   if compareType (extractSemantic s11).type_def clip_type
|     && compareType (extractSemantic s12).type_def clip_type
|     then SClipConcat(s11, s12, {actions=[]};
|         type_def=clip_type})
|   else raise (SemanticError
|     "ClipConcat operation expects type of Clip, Clip")

| ClipPixel (clip, coord, tm) ->
| let s1 =
| buildExprSemantic ctx clip and stm =
| buildExprSemantic ctx tm in
|   if compareType (extractSemantic s1).type_def clip_type
|     && ((compareType (extractSemantic stm).type_def int_type)
|       || (compareType (extractSemantic 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 =

(* 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

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 = build_expr_semantic ctx main
and s_time = build_expr_semantic ctx time
and s_value = build_expr_semantic ctx value

(match s_main with SDotExpr (sexpr, x, sem) ->
| _ -> 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
        SAssign (Some(l_expr_sem), r_expr_sem)
    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" ^
" has to be of Int of Double type")
) else raise (SemanticError ((string_of_expr time) ^
": 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"))
)
| 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) ^
" 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
(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) ^
" in the for loop has to be of Array type")))
let s_stmt_list = List.map (build_stmt_semantic nested_env) stmts in
SFORange(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
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)) ^ ">(){{" ^ (generate_type_modifier s.type_def) ^ "}
    | 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))

66
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
  | Some (expr) -> "else if (" ^ generate_expr expr ^ ") {

let rec generate_stmt = function
67

| 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
| | | | match sem.type_def.t_name with
| | | | | "Int" -> "int " ^ (generate_expr expr)
| | | | | "Double" -> "double" ^ (generate_expr expr)
| | | | | "String" -> "string" ^ (generate_expr expr)
| | | | | "Bool" -> "bool" ^ (generate_expr expr)
| | | | | _ -> "auto " ^ (generate_expr expr)
| | | | ^ " = " ^ (generate_type_modifier sem.type_def) in
| | | | String.sub tm 16 ((String.length tm)-16)) ^ "">
| | | 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)
| | | "Double" -> "double" ^ (generate_expr expr)
| | | "String" -> "string" ^ (generate_expr expr)
| | | "Bool" -> "bool" ^ (generate_expr expr)
| | | _ -> "auto " ^ (generate_expr expr)
| | | ^ " = " ^ (generate_expr s)
| | ^ " = 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))
| |
| ^ ";
|
| 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) {}
| (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) ^ "))) {" ^
| ^ "} ^`
| 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 ^ "--") ^ ") {
| (generate_stmt_list stmt_list) ^ "}

| SWhileStmt (x, l) -> |
| "while (" ^ generate_expr x ^ ")" ^ "}

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

and generate_stmt_list stmt_list = String.concat 
| (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" |
| "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) ) |
| ^ ")

let generate_eval args stmts s =
"static " ^ (generate_typeModifier 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_typeModifier arg_sem.type_def)
     ^ " " ^ arg_name )) args)) ^ "") {
" ^ (generate_stmtList 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";

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 ^
           ";\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"

      (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 ->
          "\n" ^ (generate_typeModifier mtd.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_typeModifier arg_sem.type_def)
                ^ " " ^ arg_name )) args)) ^ "") {
" ^ (generate_stmtList stmts) ^ "\n\n")}

70
let generate context program =

let generate context program =

let header = """"..../src/yolib.h""

let pre_defined = List.map (fun h ->""#include ""h"""

in

String.concat "\n" pre_defined ^
"\n/********************INCLUDE END******************/
" ^
(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 ^ ";
"
)
"" (NameMap.fold (fun k v lst -> v :: lst) context.typetab [])) ^
"\n/********************DECLARATION END*******************/
" ^
(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*************/
" ^
(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**************/
" ^
"int main() {
(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

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

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

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

let string_of_action = function
| NewVar -> "new"
| NewArr -> "new array"
let rec string_of_type = function
    | BaseTypeEntry t -> t.t_name
    | ArrayTypeEntry t -> (string_of_type t) ^ "[]"

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

(** Expressions *)
| SLiteral of string * sem
| SArrayLiteral of s_expr list * sem
| SArrayIndex of s_expr * s_expr * sem
| SArrayRange of s_expr * s_expr * s_expr * sem
| SVar of string * sem
| SDotExpr of s_expr * string * sem
| SBinop of s_expr * op * s_expr * sem
| SCall of s_expr option * base_type * s_expr list * sem
| 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

(** Statements *)
| SASign 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
| 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

(** Declaration *)
| SVarDecl of string * sem
| SVarDecl of string * sem

and s_func_decl =
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 s)
| 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_sem s)
| SClipFrameCascade (cl1, cl2, tm, s) -> (string_of_s_expr cl1) ^ 
  "=" ^ (string_of_s_expr cl2) ^ 
  "@" ^ (string_of_sem s)
| SClipConcat (cl1, cl2, s) -> (string_of_s_expr cl1) ^ 
  "+" ^ (string_of_s_expr cl2) ^ (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_of_s_stmt exec (List.tl conds))
| SForIn(var, s, expr, stmts) -> "for " ^ var ^ " in " ^ (string_of_s_expr expr) ^ ":
" ^ (String.concat "\n" (List.map string_of_s_stmt stmts))
| SForRange(var, s, exprst, expred, stmts, dir) -> "for " ^ var ^ (string_of_s_expr s) ^ " = " ^ (string_of_s_expr exprst) ^ (match dir with | Inc -> " to " | Dec -> " downto ") ^ (string_of_s_expr expred) ^ ":
" ^ (String.concat "\n" (List.map string_of_s_stmt stmts))
| SWhileStmt(expr, stmts) -> "while " ^ (string_of_s_expr expr) ^ ":
" ^ (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) ^ ":
" ^ (String.concat "\n" (List.map string_of_s_stmt stmts))

and string_of_s_stmt_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 ^ "(" ^
(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 =
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  

76
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 \n   scanner.cmo ast.cmo parser_test.cmo
   ocamlc -g -o typereader_test parser.cmo scanner.cmo \n   ast.cmo sast.cmo type_reader.cmo print_typetab.cmo\n   typereader_test.cmo
   ocamlc -g -o generate_test parser.cmo scanner.cmo \n   ast.cmo sast.cmo semantic.cmo type_reader.cmo \n   print_typetab.cmo generate.cmo generate_test.cmo
   ocamlc -g -o semantic_test parser.cmo scanner.cmo \n   ast.cmo sast.cmo semantic.cmo type_reader.cmo \n   print_typetab.cmo semantic_test.cmo

.SUFFIXES: .ml .cmo .cmi .ml1 .mly .ml
.PRECIOUS: %.ml %.mli %.cmo

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

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

.ml.ml:
   ocamllex $<

.mly.ml:
   ocamlyacc -v $<

.mly.mli:
   ocamlyacc -v $<

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

# Generated by ocamldep
ast.cmo:
ast.cmx:
parsing.cmo: ast.cmo parsing.cmi
parsing.cmx: ast.cmx parsing.cmo
parsing.cmi: ast.cmo
scanner.cmo: parsing.cmi
scanner.cmx: parsing.cmx
parser_test.cmo: scanner.cmo parser.cmi ast.cmo
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
}
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" "$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/.*\///'`
    reffile=`echo $1 | sed 's/.yo//'`
    basedir=`echo $1 | sed 's/\/[\^/]+$/\//'`
    echo -n "$basename.........................." 1>&2
    echo "$basename........................." 1>&2
    echo "$reffile............................." 1>&2
    echo "$generatedfiles="" 1>&2

    Compare "../test/preprocessor/intermediate/`echo $1 | sed 's/\/[\^/]+$/\//'` "$reffile" $1 1>&2
    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 "#" 1>&2
    echo
    echo
    echo "Testing $basename"
    echo $1>&2
    generatedfiles=""
    YO=./parser_test
    generatedfiles="$generatedfiles $(basename).a.out" &&
    Run "$YO" ">
    $basename.a.out 2> $basename.a.diff &&
    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 $1>&2
        globalerror=$error
        fi
    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 "#" 1>&2
    echo "Testing $basename"
    echo $1>&2
    generatedfiles=""
    # old from microc - interpreter
    # generatedfiles=$(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" "$1 "" "$({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 $basename
  #
  #####
  echo 1>&2
  generatedfiles=""
  YO=./semantic_test
  generatedfiles="$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/.*\:///\n                s/.yo//'
  reffile=`echo $1 | sed 's/.yo$//'`
  basedir=`'echo $1 | sed 's/\/[\^/]+$/\///'`
  echo -n "$basename..."
  echo 1>&2
  echo 
  #
  #####
  Testing $basename
  #
  #####
  echo 1>&2
  generatedfiles=""
  tmpfiles=""
  # old from microc - interpreter
  # generatedfiles="$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 \
  \n  \n  82
${basename}.f.out yo.prog"
Run "$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" "-i" "$1" &&
#Run "$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 l>&2
echo "######## Testing $basename" l>&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 -lavdevice
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/.*/'\// $/yo/\//'
    reffile=`echo $1 | sed 's/.yo$//'`
    basedir=`echo $1 | sed 's/[\^/]*/$/'`

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

    generatedfiles=
    tmpfiles=

    YO="/generate_test"
    generatedfiles="$generatedfiles $(basename).f.cpp \
Run "$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 getopt 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>> $globallog
  ;;
  esac
  echo "$file"
  ""
*test-parser*)
    echo "##### Now Testing Parser #####"
    python $preproc_path $file
    echo "\033[32m OK \033[0m"
    CheckParser $file 2>> $globallog

*test-semantic*)
    echo "##### Now Testing Semantic Analysis #####"
    python $preproc_path $file
    echo "\033[32m OK \033[0m"
    CheckSemanticAnalysis $file 2>> $globallog

*test-full*)
    echo "##### Now Testing FullStack #####"
    python $preproc_path $file
    echo "\033[32m OK \033[0m"
    TestRunningProgram $file 2>> $globallog

*test-munan*)
    echo "##### Now Testing Munan #####"
    python $preproc_path $file
    echo "\033[32m OK \033[0m"
    MunanTest $file

*test-typereader*)
    echo "##### Now Testing Single FullStack #####"
    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"
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> \
    tlp)
        { return tr1::shared_ptr<_Clip>(new _Clip(tlp));
    }

/*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/");
*/

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

    auto res = trl::shared_ptr<vector<tr1::shared_ptr<_Clip>> 
    >(new vector<tr1::shared_ptr<_Clip>>());
    for (auto filename : Filenames){
        //create clip from filename
        tr1::shared_ptr<_Clip> clip = createClip(filename);
        //add clip to timeline
        res.value().push_back(clip);
    }
    return res;
}
//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
*/
tr1::shared_ptr<Timeline> 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() << end1;
}
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;
            }
        }
    }
}

93
modifyhead = true;
}
if ((*iterator)->Position() < endtime){
    if ((*iterator)->Position() + \n        ((*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);
}

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);
}

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);
      }
   }
}
(*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();
    }
};
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;
}
```swift
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
```