# LGA Final Report

Hang Qian
Pindan Hao
Yuanli Dong
Tian Xia
{hq2124, ph2389, yd2270, tx2126}@columbia.edu

# Contents

# Chapter 1

# Introduction

LGA, stands for *Language for Graphic and Animation*, is designed to make writing expressive manipulation of graphics and animation very easily while with a concise and intuitive syntax.

LGA is the final project of *Programming Languages and Translators* course, instructed by Prof. Stephen Edwards. It is designed and implemented by Hang Qian, Pindan Hao, Yuanli Dong and Alex Tian Xia.

In this report, we will go through the current version of design and implementation of LGA. In the following chapters, we will give a brief tutorial of the compiler and languge followed by the complete lexcial convention and syntax of current version LGA. After that we talk about what we learned in developing and with all source code appended at the end.

# Chapter 2

# Tutorial

## 2.1 Using `lgac` compiler

`lgac` supports two modes: LGA mode and Javascript mode. With LGA mode, which is also the default compiling mode, it will produce the javacript code that embedded in the LGA runtime. Then it can be rendered in browser directly. While with javascript mode, `lgac` will produce a equivalent javascript code.

Run `lgac --help` will get one the usage message.

```
lgac: [-js] [-o outputfile] inputfile
  -js Enable Javascript mode
  -o Output file
  -help  Display this list of options
  --help  Display this list of options
```

## 2.2 Elegant syntax

The syntax of LGA is inspired by R[1], Coffeescript[2] and Python[3]. It is designed to be minimalized and expressive, just focus on the target task. Compare to the target language Javascript, LGA provides a more succint, expressive and programmer-friendly syntax and core functions.

LGA treats white spaces as the indication of blocks (similar to Python). And developers can omit parentheses in many cases. For instances

```
f = fun () ->
      if a and b
         return 2
      else
         return 3
```

[1]R-Project: `http://www.r-project.org/`
[2]Coffeescript: `http://coffeescript.org/`
[3]Python: `http://python.org/`

provides a very succint and intuive syntax compare to the languages that uses braces and parentheses.

Furthurmore, LGA also provides more keywords to achieve a even more expressive syntax. The above example is equivalent to

```
f = fun () ->
      return 2 when a and b
      return 3
```

## 2.3 Intuitive sementic

LGA is designed the ease the pain of rendering graphic and animation on web. We want to provide developers a mechanism that they don't need to worry the different API that browser provided but focus on just describing what they want to draw and how those shapes move.

At the top level, LGA will recognize the object variable with its type and automatically add it to render list. By modify the builtin `run` function, one can define how the animation would like at 60 fps rendering speed.

```
rect = {
    type : Rectangle
    size : [100, 100]
    pos : [300, 300]
    scale : 0.8
    fill : "#FF8000"
    stroke : "orangered"
    rotate : fun (t) -> @rotation = (t % 60) * Math.PI / 60.0
    move : fun() -> @translation.x = @translation.x + 1
}

circle = {
    type : Circle
    size : [100]
    pos : [500, 300]
    scale : 0.9
    fill : "#0000FF"
    stroke : "black"
    move : fun() ->
               t = fun (o) ->
                   o.y  = o.y + 1
                   if o.y > 1080
                       o.y = 0
               return t(@translation)
}

run = fun (t) ->
    rect.move()
    rect.rotate(t)
    circle.move()
```

The above example defines two shapes: a rectangle and a circle. There are several builtin properties that one can set including `type`, `size`, `pos`, `scale`, `fill` and `scale`. Addition to that, there's no limit to add more properties on the fly and use them later.

By definition like with, LGA will render in browser like this

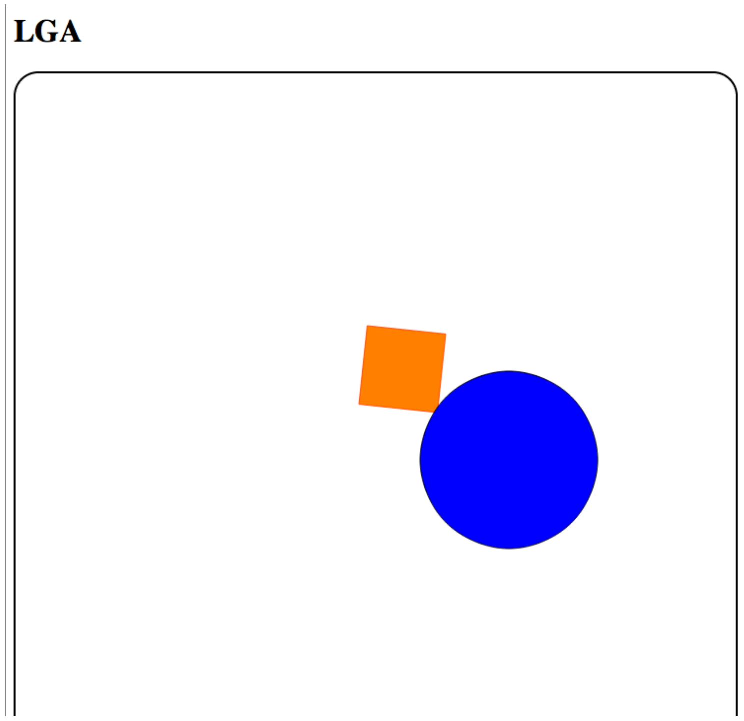

Figure 2.1: A Rectangle and A Circle

## 2.4 Compatible to Javascript

With `-js` mode set, LGA can produce a fully compatible javascript mode.

```
gcd = fun (a,b) ->
     return a when !b
     return gcd(b, a % b)

console.log(gcd(98, 21))
```

will produce

```
var gcd = function (a, b)
{
    if (!b)
    {
      return a;
    }
    return gcd(b, a % b);
}
console.log(gcd(98, 21));
```

and run it in a browser or a server side runtime liek Nodejs will print `7` to the console.

# Chapter 3

# Lexical Conventions

## 3.1 Tokens

There are five classes of tokens: `identifiers, keywords, literals, operators` and `separators`. LGA use whitespaces as `separators`(similar to Python programming language). If the input stream has been separated into tokens up to a given character, the next token is the longest string of characters that could constitute a token.

### Comments

The character `#` introduce a comment and the next `\n`(newline) terminates it. Comments do not occur within string or character literals.

### Identifiers

Identifiers(names) must start with _ or any lowercase and uppercase letters. The rest of the string can contain the same characters plus number digits (`'0' - '9'`).

### Keywords

The following identifiers are reserved for the use as keywords, and may not be used otherwise. These include the superset of both JavaScript keywords and reserved words.

### Numeric Literals

An integer constant consisting of a sequence of digits is taken to be octal if it begins with 0 (digit zero), decimal otherwise. A floating constant consists of an integer part, a decimal part, a fraction part, an `e` or `E`, and an optionally signed integer exponent. The integer and fraction parts both consist of a sequence of

|  |  |  |  |
|---|---|---|---|
| `true` | `false` | `not` | `this` |
| `when` | `return` | `isnt` | `is` |
| `fun` | `return` | `break` | `continue` |
| `if` | `else` | `or` | `and` |
| `while` | `in` |  |  |

Table 3.1: Keywords

digits. Either the integer part, or the fraction part (not both) may be missing; either the decimal point or the `e` and the exponent (not both) may be missing.

### String Literals

A string constant is a sequence of characters surrounded by double quotes. Double quotation marks can be contained in strings surrounded by single quotation marks, and single quotation marks can be contained in strings surrounded by double quotation marks.

## 3.2 Operators

### Computational operators

`+ - * / %`

### Logical operators

`&& || & | !`

### Comparison operators

`== != < > <= >=`

# Chapter 4

# Grammar

## 4.1 Syntax Notation

In the syntax notation used in this manual, syntactic categories are indicated by `typewriter style` and forms as a capitalized word, as `Expression`. Literal words, tokens, and characters are in all upper letter words, as `TERMINATOR`.

## 4.2 Program structure

### Root

The `Root` is the top-level node in the syntax tree. Since we parse bottom-up, all parsing must end here.

```
Root:
    NULL
    Body
    Block TERMINATOR
```

### Body

The `Body` node is any list of statements and expressions.

```
Body:
    Line
    Body TERMINATOR Line
    Body TERMINATOR
```

### Line

```
Line:
    Expression
    Statement
```

## Statement

```
Statement:
    Return
    Comment
    STATEMENT
```

## Expression

```
Expression:
    Value
    Invocation
    Code
    Operation
    Assign
    If
    While
    For
```

## Code

The Code node is the function literal. It's defined by an indented block of **Block** preceded by a function arrow, with an optional parameter list.

Function is a body of executable code which gets specific number of parameters then process statements inside the function body and return values if needed. The following is some examples for functions:

```
sum = fun (x, y) -> x+y
getdouble = fun (x) -> sum x, x
givemefive = fun () -> 5
sayhey = fun (hey) -> hey
```

In the first case, the function named sum, input parameters are x and y. For the second case which has name of getdouble, it has only one parameter which is x. Function body starts after the arrow operator. In the first case, statements in function body sum up two input parameters. In the second case, it calls sum, the function has been defined previously, and pass the input parameter x to sum. Of course, LGA supports functions without any input parameter like shown in the third case. In that case, no parameter will be defined and there will only be a pair of empty parentheses. Default parameters are also well covered in LGA just like the fourth case. A function could be called with its name followed by a pair of parentheses inside which contains parameters if defined. The second is an example of calling function sum and passing x as parameter. At the end of function, the final value will be returned to the caller. For example, in the first case, the returned value will be (x+y). Parameters will be passed to functions by value. Any modification on the parameter inside a function will not put any influence on the original object/variable

```
Code:
    PARAM_START ParamList PARAM_END FuncGlyph Bock
```

### Value

`Value` literal is the types of things that can be treated as values, which means they can be assigned to, invoked as functions, indexed into, etc.

```
Value:
    Assignable
    Literal
    Parenthetical
    This
```

### Block

LGA use whitespaces as levels of identation. A `Block` is an indented block of expressions.

```
Block:
    INDENT OUTDENT
    INDENT Body OUTDENT
```

## 4.3 Identifiers, Numerics and Literals

### Identifier

A literal identifier, which is a variable name or property.

```
Identifier:
     IDENTIFIER
```

### AlphaNumeric

`AlphaNumeric` is separated from the other `Literal` matchers because they can also serve as keys in object literals.

```
AlphaNumeric:
    NUMBER
    STRING
```

### Literal

All of immediate values. Generally these are fully campatible with our target language, which means can be printed

```
Literal:
    AlphaNumeric
    NULL
    BOOL
```

### This

```
This:
    THIS
    @
```

`ThisProperty` is a reference to a property on this

```
ThisProperty:
    @ Identifier
```

## 4.4 Control Flow

LGA supports common `If`, `While` and `For` loop as control flows.

### If

Addition to regular if block, LGA supports `Post-If` style. For example `x = 2 if y > 3`. Code in block will be executed if evaluation result of expression is boolean true.

```
If:
    IfBlock
    IfBlock ELSE Block
    Statement WHEN Expression
    Expression WHEN Expression
```

```
IfBlock:
    IF Expression Block
    IfBlock ELSE IF Expression Block
```

### While

Similar to if block, `Post-While` style is support in LGA. For example `x = x * 2 while x < 100`. Code in block will be executed repeatedly if evaluation result of expression is boolean true.

```
While:
    WhileSource Block
    Statement WhileSource
    Expression WhileSource
```

```
WhileSource:
    WHILE Expression
```

### For

For block iterate through `ForValue`. For example, in `for x in [1, 2 ,3, 4]`, x is repeated assigned as elements in the array. To iterate an `Object`, use two `ForValue`s separated by comma.

```
For:
    ForBody Block

ForBody:
    ForStart ForSource

ForStart:
    FOR ForVar

ForVar:
    ForValue
    ForValue, ForValue

ForValue:
    Identifier
    Array
    Object

ForSource:
    FORIN Expression
```

## 4.5 Others

### Assign

Assignment of a variable, property, or index to a value

```
Assign:
    Assignable = Expression
    Assignable = TERMINATOR Expression
    Assignable = INDENT Expression OUTDENT
```

### Assignable

This catagory consists of everything that can be assigned to.

```
Assignable:
    SimpleAssignable
    Array
    Object

SimpleAssignable:
    Identifier
    ThisProperty
```

## AssignObj

Assignment when it happens within an object literal. The difference from the ordinary `Assign` is that these allow numbers and strings as keys. And we use `:` as assign operator here.

```
AssignObj:
    ObjAssignable
    ObjAssignable : Expression
    ObjAssignable : INDENT Expression OUTDENT
    Comment
```

## ObjAssignment

```
ObjAssignable:
    Identifiers
    AlphaNumeric
    ThisProperty
```

## Array

```
Array:
    [ ]
    [ ArgList ]
```

## Object

In LGA, an object literal is simply a list of assignments.

```
Object:
    { AssignList }
```

## AssignList

Assignment of properties within an object literal can be separated by comma, as in Javascript, or simply by newline

```
AssignList:
    NULL
    AssignObj
    AssignList OptComma TERMINATOR AssignObj
    AssignList OptComma INDENT AssignList OptComma OUTDENT
```

## OptComma

An Optional, trailing comma.

```
OptComma:
    NULL
    ,
```

## Return

In LGA, functions will always return their final values even though when we dont actually use any return statement or operator, as following: `sqr = (x) -> x*x`. The value of `x*x` will be returned to the caller as the final value of the function. As shown, flowing off the end of function then the final value will be returned. Of course, return statement with which a function can return to the caller is also provided. Examples of using return statement are as following:

```
return
return ( expression )
```

The first sample dose not return any value but just terminal the process. The second sample returns value of the expression to the caller. With a return statement, logical flow of a function could be easily controlled. When some specific cases are captured, function could be terminated with or without returning a value.

```
Return:
    RETURN Expression
    RETURN
```

## Comment

LGA only support inline comment, starting with a # and terminates with a newline

```
Comment:
    COMMENT
```

## Invocation

Ordinary function invocation.

```
Invocation:
    Value Arguments
    Invocation Arguments
```

## Arguments

The list of arguments to a function call.

```
Arguments:
    CALL_START CALL_END
    CALL_START ArgList OptComma CALL_END
```

## ArgList

The `ArgList` is both the list of objects passed into a function call, as well as the contents of an array literal.

```
ArgList:
    Expression
    ArgList , Expression
    ArgList OptComma TERMINATOR Expression
    INDENT ArgList OptComma OUTDENT
    ArgList OptComma INDENT ArgList OptComma OUTDENT
```

## FuncGlyph

```
FuncGlyph:
    ->
```

## ParamList

The list of parameters that a function accepts can be of any length

```
ParamList:
    NULL
    Param
    ParamList , Param
    ParamList OptComma TERMINATOR Param
    ParamList OptComma INDENT ParamList OptComma OUTDENT
```

## Param

```
Param:
    ParamVar
    ParamVar = Expression
```

```
ParamVar:
    Identifier
    Array
    Object
    ThisProperty
```

## Index

Indexing into an object or array using bracket notation.

```
Index:
    INDEX_START IndexValue INDEX_END
```

```
IndexValue:
    Expression
```

## Parenthetical

```
Parenthetical:
    ( Body )
    ( INDENT Body OUTDENT )
```

# Chapter 5

# Architectural Design

Fig 5.1 is a diagram to illustrate the main components of the LGA project.

## 5.1 Major components

LGA compiler takes LGA source code as input (with extension name `.lga`). It is designed to translate the LGA language which is defined by us to Javascript code which is related to graphic and animation. The code we generated can by plugged int any web page that can be shown on a browser. The architecture of LGA consists of the following 5 components.

### Scanner

This part contains the file `scanner.mll`. The scanner is the most front-ended part of the whole compiler, the source code is scanned here and generate tokens according to the token list defined by us. Comments are ignored here.

### Parser

This part contains file `ast.mli` and `parser.mly`. In file `ast.mli`, we define the interface that the parser should implement. Based on the AST's definition, the parser component will define constructors and make the whole grammar tree a recursively defined constructor.

### Semantic

This part contains file `semantic.ml`. Our semantic analyzer basically take the AST as input and did a pattern matching on its leveled defined structure, and translate the source to Javascript code snippets according the rule we defined here.

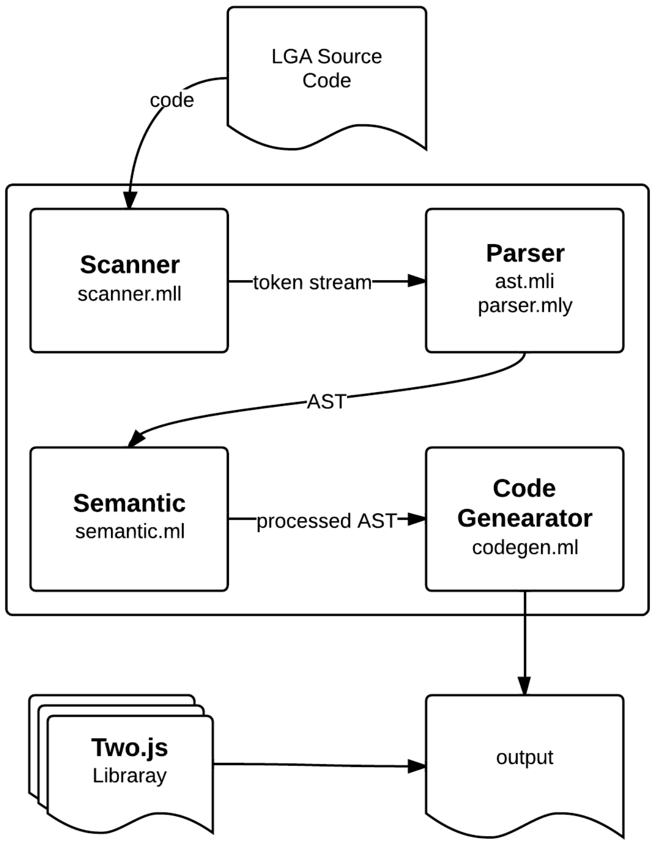

Figure 5.1: Architecture of LGA project

## Code generation

This part contains file `codegen.ml`. Our code generator take the processed AST as input and extract the objects in the tree, their attributes, their bounded functions and re-organize and formalize them into a way that can be used as input of animations' source code.

## Javascript tools

On the backend, we use a tool Two.js which is a tool used in JS to write animations, the in-built functions in LGA is a invocation to the functions provided by this tool.

## 5.2 Interface

The interface we used in LGA is `ast.mli`, this file provided the definition of our grammar and defined the overall architecture of the AST. This interface is implemented by parser, and used by Semantic component. And also, interface between semantic and code generation is `lga.mli`

Chapter 6

# Project Plan

For the whole design and developing processing, we tried to follow a synchronously development and a routinely meeting.

Before reference manual due, we exchanged emails about the language design details. There were roughly 2 emails per day that we cover each and every detail of syntax and functionality. Since the first week of november until the project due, we met 3 hours per week working together to make sure each one's part is keeping pace of the whole project.

## 6.1 Development

We manage our development using `git`. We maintain a `dev` branch that each one's working can be merged into this after passing the tests. The `master` branch is get merged after every milestone (lexer done, parser done, etc). We tag the `master` branch so that we can keep track where are we very easily. Hang, as the team director, is in charge of making technical decision and maintain the version control system.

## 6.2 Timeline

## 6.3 Roles

- **Hang Qian**: Team director. Major part of language design. Works on all parts of the project. Major contribute to semantic and code generation.
- **Pindan Hao**: In charge of scanner implementation and tests. In charge of all documentation.
- **Yuanli Dong**: Participate in language design. Co-in charge of parser implementation and tests with Alex. Work on semantic implementation also.

| Milestone | Tag in `git` | Date |
|---|---|---|
| Proposal | N/A | Sept 24th 2013 |
| Language Reference Manual | N/A | Oct 27th 2013 |
| Scanner freezed | v0.1 | Nov 7th 2013 |
| Test suite done | v0.1.5 | Nov 15th 2013 |
| Parser freezed | v0.2 | Nov 22th 2013 |
| Sementic done | v0.3 | Nov 25th 2013 |
| Code Generation alpha | v0.4 | Dec 2nd 2013 |
| First Demo working | v0.5 | Dec 3rd 2013 |
| All code freezed | v0.5.5 | Dec 14th 2013 |

Table 6.1: LGA Timeline

- **Tian Xia**: Participate in language design. Co-in charge of parser implementation and tests with Yuanli. Work on semantic implementation also.

## 6.4 Project Log

The following log shows the commit on `master` branch.

```
commit 33feb2c0315251a19a420b63640130759c21207f
Merge: 4200c92 96f0960
Author: Hang Qian <hangqian90@gmail.com>

    Code freeze. Ready to go!

commit 96f0960109195bbd60a4f395f02f42b1b01eaa92
Author: Hang Qian <hangqian90@gmail.com>

    Add test files; Freeze code

commit 89cc431de4fab6835b68c6dde1ad9a05df9c2496
Author: Hang Qian <hangqian90@gmail.com>

    Fix semicolon

commit 5d97723cd1f478dc54aaaa13157cdad036917100
Merge: d698529 5aa043d
Author: Hang Qian <hangqian90@gmail.com>

    Merge branch 'dev' of https://bitbucket.org/lga/lga into dev

commit d6985294a970d1448a318bac0f27ef35299a0b2a
Author: Hang Qian <hangqian90@gmail.com>
```

```
    rm parser.ml

commit 5aa043d433443d376486f770f1f977b3cfb4aa88
Merge: 15ea38d 6cc1598
Author: Alex <alex.xiat@gmail.com>

    Merge branch 'dev' of https://bitbucket.org/lga/lga into dev

commit 15ea38d24a9f5e061b04782c568cf7ae42bc344b
Author: Alex <alex.xiat@gmail.com>

    fix semicolon

commit 6cc1598f622709e074f7bb072dfe58206012da02
Merge: a583902 14150e9
Author: Hang Qian <hangqian90@gmail.com>


    Merge branch 'dev' of https://bitbucket.org/lga/lga into dev

commit a583902147134b96d93f2f9aefb33f1a18699bfb
Author: Hang Qian <hangqian90@gmail.com>


    fix demo

commit 48b07955cd82bb6081dcaa88f56968c753f01cf1
Author: Hang Qian <hangqian90@gmail.com>


    Add inline function support

commit ca72b3b1bec01b0be029560d389feaa600193560
Author: Hang Qian <hangqian90@gmail.com>


    support inline code

commit 14150e998386c099477ccd1a3aa1493171b8769e
Author: Pindan <qingluo0707@gmail.com>


    Now tests for scanner all work.

commit e6726f540d2d4a7806a27970ce5a518c81bf48cc
```

```
Author: Hang Qian <hangqian90@gmail.com>


    clean up

commit eac685f2d6dc05551f5d4a7ca1314e2f30b45d3d
Author: Hang Qian <hangqian90@gmail.com>


    All working except scanner test

commit 0ce092150ea59a3032377c34f5077b519ec65524
Author: Alex <alex.xiat@gmail.com>


    semantic: remove semi colon after }

commit 8ed3b34c739082adf5434aa7649c6ada7351b8b2
Author: Hang Qian <hangqian90@gmail.com>


    fix

commit 4200c927d49d915f74a754534761e041f31b198e
Merge: 562b6ee e1d62fc
Author: Hang Qian <hangqian90@gmail.com>


    Praise God! Finally working!

commit e1d62fc47c5550b11d864cc2504b513624d3c42a
Merge: 41ffe06 878738d
Author: Hang Qian <hangqian90@gmail.com>


    Merge branch 'haq' into dev

commit 878738d309c5ab6cc202d0bc5eacb7f7f117df79
Merge: f0ead8a f1de9bd
Author: Hang Qian <hangqian90@gmail.com>


    Finally working

commit f1de9bddb31e1835816ecc55b268a75f724c3704
Author: Hang Qian <hangqian90@gmail.com>
```

```
    let’s merge

commit 41ffe067fdd850f8bf99a838003abf30d464eabe
Merge: 116b351 f0ead8a
Author: Hang Qian <hangqian90@gmail.com>


    Merge branch ’haq’ into dev

commit f0ead8ad7b76163200ef93a6bca824be9d74ebed
Author: Hang Qian <hangqian90@gmail.com>


    Change name

commit 1f57a5e7d0b6c55fccd122a57a103a2f7510a6cb
Author: Hang Qian <hangqian90@gmail.com>


    Code Generating works

commit 95a11149b50a0b315cce7eaba952b579a6156b02
Author: Alex <alex.xiat@gmail.com>


    code generate

commit b5c8a288621f92f4efd5ff2c1c064112ce79711b
Author: Alex <alex.xiat@gmail.com>


    code generation

commit 116b351d74a113f9d6153cf670b04339e2420839
Merge: 90ed030 7f31445
Author: Alex <alex.xiat@gmail.com>


    Merge branch ’parser_test’ into dev

commit 7f3144583d92f2cec4f33387500cbc31474d94a8
Author: Alex <alex.xiat@gmail.com>
```

```
    Test Parser done

commit 90ed030682dd9da3cb242342a64e890c4afc4801
Author: Hang Qian <hangqian90@gmail.com>


    remove parser.ml

commit ea4593557ab557f6be515bf9a00903a1b3053f67
Author: Hang Qian <hangqian90@gmail.com>


    fix obj

commit 8d43f6f1d4a38ea829fa433a806abcc9b0a145f0
Author: Hang Qian <hangqian90@gmail.com>


    fix obj syntax; change uitls

commit b94c26d52de7ba9042e5369894b4244236242742
Author: Hang Qian <hangqian90@gmail.com>


    Codegen alpha

commit 3f9951f94e7f72ee8cb57c67e235159fb03aaf5d
Merge: 0f760a8 fe7dfd0
Author: Alex <alex.xiat@gmail.com>


    Merge branch 'dev' of https://bitbucket.org/lga/lga into parser_test

commit fe7dfd035320b42ff54c4ce31b25d0efeaf7c871
Merge: be863c0 456eaa5
Author: Pindan <qingluo0707@gmail.com>


    Enable printing line number when encountering wrong indent.

commit 456eaa543cf5e2fe990ad3cbacdf9b257983cefe
Author: Pindan <qingluo0707@gmail.com>


    Enable printing line number when encountering wrong indent.
```

```
commit be863c0dde59b9ed79b9a963ee2ef83838294160
Merge: cb189b0 b8b4a49
Author: Pindan <qingluo0707@gmail.com>


    Merge branch 'pindan' into dev

commit b8b4a49ffdb2306a9afb9724fca091cb46231956
Author: Pindan <qingluo0707@gmail.com>


    Update scanner.mll.

commit cb189b0bc36aed5c2b677c246ad36ff7632e7d1d
Author: Hang Qian <hangqian90@gmail.com>


    Change OBJ syntax

commit 0f760a8cd54a67e11c258f74ff779ffc4f9cd926
Merge: 219c912 b47c056
Author: Alex <alex.xiat@gmail.com>


    Merge branch 'dev' of https://bitbucket.org/lga/lga into parser_test

commit 219c91230496f6629914e49a2c6014a5e004ed97
Merge: 578222b cd9585f
Author: Alex <alex.xiat@gmail.com>


    test_assign

commit b47c0567f7538cb56f63659235565b9d4965ea59
Merge: be817b1 a2593aa
Author: Hang Qian <hangqian90@gmail.com>


    merge

commit be817b1f95d17f3697795af6cf7d22a1573718be
Author: Hang Qian <hangqian90@gmail.com>


    fix IF block bug
```

```
commit a2593aa711dbb5c401a13965218c257f2008fdec
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    fix typo

commit a9926630187e89cb1c2d4ecf6e70cc59b90cc3d5
Merge: e60ecc9 9b4b910
Author: Pindan <qingluo0707@gmail.com>


    Fix conflicts in Utils.ml.

commit 9b4b910ceda0bdef70f097ad2c142ccdaf8d974b
Author: Pindan <qingluo0707@gmail.com>


    Update utils.ml

commit e60ecc94e97c542392a0e011c0747b341bc7afe0
Author: Hang Qian <hangqian90@gmail.com>


    Fix typo

commit 562b6ee3dbab8408a8c922478d278965ec34da97
Merge: 70bc113 cd9585f
Author: Hang Qian <hangqian90@gmail.com>


    It's time for v0.3

commit cd9585f6a62ba0dccae90e951300a2c017a6e754
Merge: 8adeeae 906d4af
Author: Hang Qian <hangqian90@gmail.com>


    Merge branch 'dev' of https://bitbucket.org/lga/lga into dev

commit 8adeeae4c6ec9a75b70e370edfce68d9cb6f17b7
Author: Hang Qian <hangqian90@gmail.com>


    Fix Indent bug

commit 906d4afeafe1341c068bb31ffa5c442a975241dc
```

```
Author: Pindan <qingluo0707@gmail.com>


    Add comments to utils.ml.

commit 0494d59640f808777f38268df26d45a212ba582c
Merge: c9f4caf ec929af
Author: Pindan <qingluo0707@gmail.com>


    Add back Makefile in src.

commit c9f4caf0145e5c950a2f694dd3de447d7c6b9810
Author: Pindan <qingluo0707@gmail.com>


    New makefile at root folder: generate documents enabled.

commit 578222b8687fea4b7d70bcf044f5daed10593011
Merge: fc8f108 d71b0f5
Author: Alex <alex.xiat@gmail.com>


    Merge branch 'parser_test' of https://bitbucket.org/lga/lga into parser_test

commit fc8f108322d55aa4e979ee17ffc607516f00ff0d
Author: Alex <alex.xiat@gmail.com>


    assign test

commit d71b0f5b203ade463f42710868167135ddec840e
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    semantics

commit 70bc11381d6e4623d1e2326359f74faef29b6b50
Merge: 448b47c ec929af
Author: Hang Qian <hangqian90@gmail.com>


    Sementic now working.

    But parser seems broken now.
```

```
commit 12701471004cca21775b2aab7804aaf3eb10c76e
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    no message

commit ec929af81dddc117aee8fb64dce8492817c6e583
Merge: 52228b0 f2b4704
Author: Hang Qian <hangqian90@gmail.com>


    Resolve conflicts

commit 52228b03fa19aca187ad9f115a07465ecfb47765
Merge: 234747d 6e9f602
Author: Hang Qian <hangqian90@gmail.com>


    Merge branch 'haq' into dev

commit 6e9f6029bfd788c0cc7210cd7621c11576b4d6a8
Author: Hang Qian <hangqian90@gmail.com>


    Ast now working

commit f2b4704fd5bcc992344f66f4f18fab44190a3cd2
Author: Pindan <qingluo0707@gmail.com>


    Update test_scanner: catch token list not matching exception.

commit d11add8a55650ff628795875f2a8ce924ee3d121
Author: Pindan <qingluo0707@gmail.com>


    Update scanner.mll.

commit 40ee3cdf59add7ddcaf41c7884a9d97e310e6741
Author: Pindan <qingluo0707@gmail.com>


    Update utils.ml and tests.

commit 8a70a09ad763a625ee67a87c00b15b74e70f0e6f
Merge: 37e095b 234747d
```

```
Author: Alex <alex.xiat@gmail.com>


    Merge branch 'dev' of https://bitbucket.org/lga/lga into parser_test

commit 10ec87f3597a6adf6901a6200dde55ae01cfa15e
Merge: b770617 234747d
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    Merge branch 'dev' into parser_test

commit 234747dd32dddd7e146a4d130f4c99e13b05553f
Author: Pindan <qingluo0707@gmail.com>


    Fix scanner test

commit b7706172b704898e18b8eb8fc0ae9376857516ca
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    no message

commit 949ff378f1cbc8a0044befd4795373be03f54471
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    test files

commit 763ee69677777f88f520b488053b7a2809092dcd
Merge: 84563d3 37e095b
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    Merge branch 'parser_test' of https://bitbucket.org/lga/lga into parser_test

commit 84563d3c2a608f60eb74c93a681f7f6642c0b722
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    changes to test_in files

commit 2d75a2f46bd026ee6cec59de5f595d88cc7aa2aa
Merge: 3a8eabe f99db5c
Author: Pindan <qingluo0707@gmail.com>
```

```
    Merge branch 'dev' into pindan

commit f99db5c0b3d90f54a8bd7b8cee36fa38725f0f6e
Author: Hang Qian <hangqian90@gmail.com>


    Fix EOF bug

commit 3a8eabe976889e1416856d6e0d6a5ac856479243
Merge: f71a70d c63f933
Author: Pindan <qingluo0707@gmail.com>


    Merge branch 'dev' into pindan

commit f71a70d364d940aa41d91d345854f67c8919932e
Author: Pindan <qingluo0707@gmail.com>


    Update util.ml.

commit 37e095bf006ab044b499bab7068a2d434d2d7474
Merge: 9861a2e cbdf8aa
Author: Alex <alex.xiat@gmail.com>


    Merge branch 'parser_test' of https://bitbucket.org/lga/lga into parser_test

commit 9861a2efe6d91371d39e06d04635e9a18501902d
Author: Alex <alex.xiat@gmail.com>


    parser testing FOR

commit cbdf8aabfb546549f12cc93742b77979b09ace9a
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    update utils file

commit 98473d3c6bd772b1e7b46bcf5c1e3f7764e70be9
Author: Yuanli Dong <cu.dongyuanli@gmail.com>
```

```
    create test function file

commit 641d79c42c05d64eebdd949dbe830bd749fd2ea6
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    test_out files added

commit c63f933f67dd5aa15ce725d63434dbe1e210360b
Author: Hang Qian <hangqian90@gmail.com>


    udpate test_scanner

commit 0e583817953f6a5609861fbf3e55af1843ab2cf9
Author: Hang Qian <hangqian90@gmail.com>


    Implement some util function

commit 68a7ad33be7e7114802ea258bacaa43638a545b9
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    parser test_in

commit d022d4e6672f0fe9f04ddf3104dadba0f6ad70b1
Author: Hang Qian <hangqian90@gmail.com>


    Add rectype flag

commit 448b47cc6ca9a730de4ac7b950fdb54d4fa55ac4
Merge: cf9c11a 0b34c01
Author: Hang Qian <hangqian90@gmail.com>


    Fix

commit 0b34c010a126434247564a1fb3d3cecf30baddaf
Author: Hang Qian <hangqian90@gmail.com>


    fix reduce/reduce conflicts

commit cf9c11aceed0411694b80ed8d7332c51966140f4
```

```
Merge: 74463c4 185fa45
Author: Hang Qian <hangqian90@gmail.com>


    Parser done!

commit 185fa45899518285a7fe31c5e6f1e564872d212c
Author: Hang Qian <hangqian90@gmail.com>


    fix typesetting

commit d9728d89423637e9cf8fd92b5827bb3112a20deb
Author: Hang Qian <hangqian90@gmail.com>


    Good Lord!
    All compile!

commit bfddb46b361f46aea1ee9db354a714109ced83a3
Author: Hang Qian <hangqian90@gmail.com>


    fix

commit 96c3ddd26703985b2eaf57952c6828803ff22df8
Author: Hang Qian <hangqian90@gmail.com>


    Add root to ast.mli

commit 74463c40bd00f49cf20f9b20e7a3d72691135738
Merge: 066c511 60e2bda
Author: Hang Qian <hangqian90@gmail.com>


    Ast compiles!

commit 60e2bda27c8fe6d7bee47e2fa09616516efec6a1
Merge: d6b298c ddb17e0
Author: Hang Qian <hangqian90@gmail.com>


    Merge branch 'dev' of https://bitbucket.org/lga/lga into dev

commit d6b298c4f90e06cc913f2e0e4f94c7496ba840d0
```

```
Author: Hang Qian <hangqian90@gmail.com>


    Dear Lord,

    I thank you for offering us polymorphic types in OCaml.

    Yours,
            Hang

commit ddb17e0be62a41e3c45abfa0b95616d7a16cd17e
Merge: f911cbf a3fe2b1
Author: Pindan <qingluo0707@gmail.com>


    Merge branch 'dev' of https://bitbucket.org/lga/lga into dev

commit f911cbf55fe4dab93c0c6580073464b8af1b7281
Author: Pindan <qingluo0707@gmail.com>


    Scanner test suite change due to language change.

commit a3fe2b139999fb4ca08db4bd9e21d9bd5b1393b3
Author: Hang Qian <hangqian90@gmail.com>


    Fix missing FUN

commit cb9cd8c6b7e9ebc87aa74b7d7f9ea874a1d3e0df
Merge: e13d901 f9f60ca
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    No conflict in parser

commit f9f60ca31b55322651d331765d9f08862fea28e2
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    No conflict in parser

commit e13d9016a681707b39badf72c73588692f999dff
Author: Hang Qian <hangqian90@gmail.com>
```

```
    Change token definition in parser.mly.
    Change bool handling in scanner.mll.

commit a5d6905e5315997a5efb2739322f292728b5be31
Author: Hang Qian <hangqian90@gmail.com>


    Change on scanner;

    1. Change NUM token to string
       Because we Literal to handle NUM and STRING, there's no need to to conversion
       at the lexing stage.
    2. Add FUN keyword to denote function declaration.
       We use "fun" keyword to resolve potential reduce/reduce conflict in parser.

commit 9d5f6c2e10226b3fa21b88bd3a021ea8c871704b
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    no message

commit 2557e807068b8fcd9236b2ac412603a74fc64365
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    no message

commit 706880a96b9c444ec8c15b7ca0058b02e75296d1
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    no message

commit a731719e9fefbeff1f58672519f6f17d124281b3
Merge: 066c511 7e933cc
Author: Hang Qian <hangqian90@gmail.com>


    Merge branch 'dev' into haq

commit 7e933cce4d6757423ec37fdfd23198fc8fe0cd5e
Author: Hang Qian <hangqian90@gmail.com>


    keep debugging
```

```
commit 6b4b195cc190b92b0ce6595eede84a0333255680
Author: Hang Qian <hangqian90@gmail.com>


    fix Return error

commit d314b8d97e4b627dadd71bf8b6c9033f720793fd
Author: Alex <alex.xiat@gmail.com>


    debug parser

commit 06f17dcc5ba76e0a8f12e00f3bd0f7170c61fd5e
Merge: e80987e 5307925
Author: Alex <alex.xiat@gmail.com>


    Merged but broken

commit 98a2bb18ba4aa79837d589e573f5cadc41269d63
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    xx

commit 5307925e28b986d62a9cac2a259f3e7aa9746f01
Author: Alex <alex.xiat@gmail.com>


    parser alpha

commit 5ceecd3659b7571fc6821450890290307400b484
Merge: 2b1ca85 545be03
Author: Alex <alex.xiat@gmail.com>


    Merge branch 'parser_dev' of https://bitbucket.org/lga/lga into parser_dev

commit 2b1ca853e7baaf6f54103691c226a1a9ff6715d5
Author: Alex <alex.xiat@gmail.com>


    parser_new

commit 545be033c3bc81a5ca2829a32a949506e7e3587e
Merge: 1de3d83 7116c4e
```

```
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    Merge branch 'parser_dev' of https://bitbucket.org/lga/lga into parser_dev

commit 1de3d8350765933e8eceef5ba1e930aa779b9e26
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    xx

commit 7116c4e9c7d540ab001ce2472c1f42f0568ebb5a
Merge: 90683f9 b78b766
Author: Alex <alex.xiat@gmail.com>


    Merge branch 'parser_dev' of https://bitbucket.org/lga/lga into parser_dev

commit b78b76685b4f58e1b885d430c16476d717e2d64c
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    xxx

commit 90683f901f436f9a22afabf13593aada8c1e8ba5
Author: Alex <alex.xiat@gmail.com>


    parser

commit 066c5112ffbcf616cd21e0081d4a71dc16769336
Merge: e1eb445 e80987e
Author: Hang Qian <hangqian90@gmail.com>


    Scanner looks good now.

commit e80987e38e80c864812ddf0d234fb342870722d0
Author: Hang Qian <hangqian90@gmail.com>


    fix Makefile

commit b624e606147ca4d5625a19599f9b3396e321eec4
Author: Hang Qian <hangqian90@gmail.com>
```

```
    update Makefile; Add test

commit 481855dc70389af1fc366e0681dd96970d372109
Author: Hang Qian <hangqian90@gmail.com>


    suppress warning

commit 97a230e03ada414c0c4a90c68580a7b148356a26
Merge: 8b602aa 122a37d
Author: Hang Qian <hangqian90@gmail.com>


    Merge branch 'dev' of https://bitbucket.org/lga/lga into dev

commit 8b602aaff636be19f58b9da361235453e7b66a98
Author: Hang Qian <hangqian90@gmail.com>


    remove duplicate utils.ml

commit 122a37d3d033016f356e565994a151776e29bc55
Merge: 1eedd62 448e057
Author: Pindan <qingluo0707@gmail.com>


    Merge branch 'dev' of https://bitbucket.org/lga/lga into dev

commit 1eedd62773af0c8dd3471e8e2c6a6c7975204da2
Author: Pindan <qingluo0707@gmail.com>


    Add new test file.

commit 3332d92e12e73c9ac723a008a18af6c932701ab1
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    modify op

commit 1ca1f8632a99b2e7233b8f0f2824a483a23ab95c
Merge: dc250d5 e6261be
Author: Alex <alex.xiat@gmail.com>
```

```
    Merge branch 'parser_dev' of https://bitbucket.org/lga/lga into parser_dev

commit dc250d593bcb652c15456339deedefca9c05a745
Author: Alex <alex.xiat@gmail.com>


    for parser

commit 448e0572579069f204180ef28303e43c0f5ba268
Author: Hang Qian <hangqian90@gmail.com>


    remove old test files

commit e6261be4dc262220b17b96462c5e3741be9e14ca
Merge: 8a75d3e 934ed78
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    Merge branch 'parser_dev' of https://bitbucket.org/lga/lga into parser_dev

commit 8a75d3e66a1115c70601d3027d9045fb8845df74
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    new ast Implement

commit 53dc8a3d2f52154a8b6f9b06820c292372d070ed
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    no message

commit 41da1f4b14db11f8ab3a82e3c4a7b3ab4fab4a4c
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    no message

commit 3b409ad9322a4c44136380083a117279057a534d
Author: Pindan <qingluo0707@gmail.com>


    Update utils.ml

commit 7fdba8f6d57910db7098307fac1d64786a164b03
```

```
Author: Hang Qian <hangqian90@gmail.com>


    Oh Shi*t

commit 5a99700851fdf636c7bf6a86fd6af4493172dbaf
Author: Hang Qian <hangqian90@gmail.com>


    remove debug info

commit 761c9c070662221abd99064aef44582bf434297e
Author: Hang Qian <hangqian90@gmail.com>


    fix scanner_test

commit c0d9717c962e5233ad9b2c7f280e20cce013c145
Author: Pindan <qingluo0707@gmail.com>


    add parser.mly for test

commit a161289faf49c1713b7a9a469b2c0a37143abfcb
Author: Pindan <qingluo0707@gmail.com>


    fix scanner

commit 934ed78c37d221dc6a8ca59cb2fb353fc6ddb27d
Author: Alex <alex.xiat@gmail.com>


    move if while for to expression

commit 34b2260aeb1130bcd599fb1a1f8b70b97a447c16
Merge: 883cfbc ec04eba
Author: Pindan <qingluo0707@gmail.com>


    Merge branch 'parser_dev' into pindan

commit 883cfbc62ee9a0ec6e839e3f6b481b0fc3e5f461
Author: Pindan <qingluo0707@gmail.com>
```

```
    Change parser a little bit.

commit 5b458f140d7793bda68d174ae3f3c62abb4cb5bc
Author: Pindan <qingluo0707@gmail.com>


    Update utils.ml

commit 7d41912c5661b65583ef582a3e3bac6bf6ec87ce
Author: Pindan <qingluo0707@gmail.com>


    Complete the tokens in scanner.mll

commit ec04eba03aa677fe22604bb1e074a1681dc88b1e
Author: Alex <alex.xiat@gmail.com>


    parser_new change object -> obj

commit 8da4e6ddb0103bf4a92302231da41f2de3981155
Author: Alex <alex.xiat@gmail.com>


    parser in dev

commit d8a22a65041011f20965d8d1b673e8cd34fbe430
Author: Alex <alex.xiat@gmail.com>


    changes on Tokens

commit 6c6164322e67f95921a1eea79f659a117f002572
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    xx

commit 7e6e89e011dbc18413573b50d33078fb80e39ce8
Author: Alex <alex.xiat@gmail.com>


    to parser_dev

commit f9a83c2311d3fd7cbf246df555d934856449ad4e
Author: Bank Qian <ibank.qh@gmail.com>
```

```
    remove scanner.ml

commit e19f4ee3ae5108659ace42ed22c61918edbb5401
Merge: c1fe90e 9d26473
Author: Pindan <qingluo0707@gmail.com>


    Merge branch 'dev' of https://bitbucket.org/lga/lga into dev

commit c1fe90e74165d7ea4626b591166391a954956d0e
Merge: beb953c 4722cec
Author: Pindan <qingluo0707@gmail.com>


    merge

commit 9d2647346765d620439fa259e6898e4d650cdc5f
Author: Bank Qian <ibank.qh@gmail.com>


    remove binary

commit 1a6591b4e3d96be1b3c2b2439f49865df1bba9d3
Author: Bank Qian <ibank.qh@gmail.com>


    Auto test done

commit beb953c5aa27ac05ae6b7f779b233fb11c37798b
Author: Pindan <qingluo0707@gmail.com>


    Updated scanner.mll.

commit 4722cec63361e3e7e13b046f28c2f55dde082fb2
Author: Bank Qian <ibank.qh@gmail.com>


    lga renamed to semantic

commit 2a9d15a10e40c37550574c3141665e0ea62c38d5
Merge: 51ee213 b43cc17
Author: Bank Qian <ibank.qh@gmail.com>
```

```
    Merge branch 'scanner_test' into dev
    Fix TERMINATOR handling

    Now scanner has no problem with existing tokens

commit b43cc1706cf12b1516ab1555fa728d965fc419c1
Author: Bank Qian <ibank.qh@gmail.com>


    fix TERMINATOR

commit 51ee2134efd4f82e613c857b40946d96318d68e2
Author: Bank Qian <ibank.qh@gmail.com>


    fix

commit edc0657e72e94cebf65dd6a619c739c28b976cbd
Author: Bank Qian <ibank.qh@gmail.com>


    fix

commit 018f41dce1f0786937a387634212b5120b7e031f
Author: Bank Qian <ibank.qh@gmail.com>


    Now test_scanner have the complete token list

commit 3018d2adb07c5242279c4d9c7565bd5d293e57e2
Author: Bank Qian <ibank.qh@gmail.com>


    Successfully implement scanner for indent and dedent

commit 3be22f8501743c84b713ee76b17a3ca44842b601
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    add Alexs code

commit 3841e017b4bfc339930079d87f1d8baf11abec13
Merge: 375f485 415a7eb
Author: Bank Qian <ibank.qh@gmail.com>
```

```
    Merge branch 'dev' into scanner_test

commit 375f485ba82c26a78dc9e90e4aaf12efcca36ee2
Author: Pindan <qingluo0707@gmail.com>


    Pindan is awesome

commit 415a7eb1c40108c319cfad4682fc2f6717615915
Merge: d2a3656 f635352
Author: Bank Qian <ibank.qh@gmail.com>


    Merge branch 'haq' into dev

commit f635352f5587036f2d27add928bce93745cf24db
Author: Bank Qian <ibank.qh@gmail.com>


    refactor test_scanner

commit d2a3656dc3b0ac39c3db5b6d79b710a3eec315f1
Merge: 7915444 256cff8
Author: Bank Qian <ibank.qh@gmail.com>


    Merge branch 'haq' into dev

commit 256cff8b7ce372f3eaff188c777e44675aa1b674
Author: Bank Qian <ibank.qh@gmail.com>


    Add gitignore

commit 7915444da56025ec87d17aed15554c35463cd86e
Merge: 3b1797a a70c220
Author: Bank Qian <ibank.qh@gmail.com>


    Merge branch 'haq' into dev.

commit a70c22040a60b9c685636a3f79e93d1d5ae7c43c
Author: Bank Qian <ibank.qh@gmail.com>
```

```
    test_scanner done

commit 3b1797af8ee84720e3afd99c7b81e4fe15636698
Author: Bank Qian <ibank.qh@gmail.com>


    Update doc makefile

commit 27753f25a55e6e8a81d095ce78c91ef44031cbdc
Author: Bank Qian <ibank.qh@gmail.com>


    Experiment with types

commit 8ccc477652239d2a2f8ac8787c1f8ca924944eca
Author: Bank Qian <ibank.qh@gmail.com>


    working

commit d37659b42b9e846b6de91c61ba1882ed4f5b9a09
Merge: dd61e56 1ef1c8d
Author: Bank Qian <ibank.qh@gmail.com>


    Merge branch 'dev' of https://bitbucket.org/lga/lga into haq

commit dd61e56b3eaaadef5b76aeb58701b8f6ceea9049
Author: Bank Qian <ibank.qh@gmail.com>


    Minor error

commit cdccbe83c601a86001119c05299f76b5137e2b03
Author: Bank Qian <ibank.qh@gmail.com>


    new Makefile

commit 1ef1c8dbeeefd23df6c2773aea59382912419cd1
Merge: 24a4527 cf0431b
Author: Alex <alex.xiat@gmail.com>


    Merge branch 'dev' of https://bitbucket.org/lga/lga into dev
```

```
commit cf0431bd2b38f90741034b01eeb50226074d9ebc
Merge: bb02b01 e80011d
Author: Ubuntu <carter@ubuntu.(none)>


    Merge branch 'dev' of https://bitbucket.org/lga/lga into dev

commit bb02b01c1d7cbde04ca029bc77c39ff1f62a19e1
Author: Ubuntu <carter@ubuntu.(none)>


    ast

commit 24a4527ef6be3faf52288104b2b467a7213da783
Author: Alex <alex.xiat@gmail.com>


    parser.mly

commit 381bc4f750723d5a566c1998a9982ff752f2ae02
Merge: 1e68419 e80011d
Author: Bank Qian <ibank.qh@gmail.com>


    Merge branch 'dev' into haq

commit e80011d077f64cd53b47cab71d94241442cd7d57
Author: Bank Qian <ibank.qh@gmail.com>


    Move files to LGA/src

commit 1e684198bc5a5afee8b78e05d759034f3c406746
Merge: e1eb445 28f2046
Author: Bank Qian <ibank.qh@gmail.com>


    Merge branches 'dev' and 'dev' of https://bitbucket.org/lga/lga into haq

commit 28f2046eb11989391033fc90f9c283a7516f239b
Merge: e9da708 ef2a2f5
Author: Pindan <qingluo0707@gmail.com>


    Merge branch 'dev' of https://bitbucket.org/lga/lga into dev
```

```
commit e9da7084a745a94865f0a6dc72b5b7211445c1a2
Author: Pindan <qingluo0707@gmail.com>


    Update scanner.

commit ef2a2f5041079d7c0dc1ae6955e8664b9b7a7d93
Author: Ubuntu <carter@ubuntu.(none)>


    ast.mli

commit 7e639d9a7049dee0a5fbe6b6acb5a15cc9fb681e
Author: Alex <alex.xiat@gmail.com>


    remove scanner.ml

commit 68d9b9fb691d6d64b159517f9c8f711aded02098
Merge: 683bcd6 43e517d
Author: Alex <alex.xiat@gmail.com>


    Merge branch 'dev' of https://bitbucket.org/lga/lga into dev

commit 683bcd60b5ea24863050f79b5bc8583529ca6852
Author: Alex <alex.xiat@gmail.com>


    makefile

commit 43e517dfccc4172ed97ddaa833cace413acbae99
Author: Pindan <qingluo0707@gmail.com>


    Updated scanner.mll.

commit 3616b33b529744ca95525b4fcf45d203882a6557
Author: Ubuntu <carter@ubuntu.(none)>


    ast.mli

commit afb3369cfc95b7963911def7d00fe84aecc02bd5
Merge: 0f283cb 03efa64
Author: Alex <alex.xiat@gmail.com>
```

```
    Merge branch 'dev' of https://bitbucket.org/lga/lga into dev

commit 03efa64850f1cf2dc81b6b54444eee846c63ef0a
Merge: 7a3c962 2068a68
Author: Pindan <qingluo0707@gmail.com>


    Merge branch 'dev' of https://bitbucket.org/lga/lga into dev

commit 7a3c96211919b78b7f8103732f4af488a9f03466
Author: Pindan <qingluo0707@gmail.com>


    Initialize scanner.mll.

commit 0f283cbcb560c16f72094b97a9f3c01316abac5f
Author: Alex <alex.xiat@gmail.com>


    parser

commit 2068a68af412f40ae78f865ac4db2f5af2853562
Author: Alex <alex.xiat@gmail.com>


    parser - beginning

commit e1eb445d2e16f63234be131762579d5e3c1a0ffb
Merge: e03dd6e d487d5e
Author: Bank Qian <ibank.qh@gmail.com>


    Merge branch 'dev'; Language Reference Manuall draft.

commit d487d5efd2d1ee97309f459aec878e64b8e84da7
Author: Yuanli Dong <cu.dongyuanli@gmail.com>


    Some description added in LRM control flow part

commit dd84b4f381f1f16945f72d8f3c2a5ffcb7d67230
Author: Bank Qian <ibank.qh@gmail.com>
```

```
    omit date

commit 8d7045e91e036559973ae41ab5625eddb021eab7
Author: Bank Qian <ibank.qh@gmail.com>


    minor change in sample code

commit e6d8e7a659c6e2f706b396917e7513ede218e746
Author: Bank Qian <ibank.qh@gmail.com>


    Correct UNI; Change structure

commit 80a72df029483954454a6c822edba302b3a3dfd5
Author: Bank Qian <ibank.qh@gmail.com>


    Add UNI; Add This property

commit e77e1df8855b4a9fe35567d4ee134c292a8dc690
Author: Bank Qian <ibank.qh@gmail.com>


    Corrections

commit fa28f935bf7b96f699b2b9912f3de93c2c7e9694
Author: Bank Qian <ibank.qh@gmail.com>


    file directory change

commit d3720f8cfdffbf1a5fbfe0fea6b8c83c9fba67e7
Author: Bank Qian <ibank.qh@gmail.com>


    minor change

commit 11109bd2dba3f77f52213fb6d3d84af282b1878c
Author: Bank Qian <ibank.qh@gmail.com>


    Add Reference Manual Draft

commit 9463e6248c8a00ef244753602904bbd1984bfa50
Author: Bank Qian <ibank.qh@gmail.com>
```

```
    Add Programming Guidelines

     new file:   docs/guidelines.txt

commit e03dd6e7ea718514c17a68952a5c8a5a44fb9a81
Author: Bank Qian <ibank.qh@gmail.com>


     Initial commit
```

# Chapter 7

# Test Plan

During our development, we conducted unit test for different functionalities every time we implemented a functional part. We processed separated tests for scanner and parser as well as demo tests for the whole compiler. Different key words, structures as well as formats were tested carefully.

## 7.1 Unit Test Cases

Unit test was conducted for each funcitonal part. We built a automatic testing procedure by writing Ocaml programs to read lga test files and do testing by comparing output with expected results written by ourselves. Here is a list of automatic testing program in Ocaml we wrote for unit test:

```
LGA/tests/testscanner/testscanner.ml
LGA/tests/testparser/testparser.ml
LGA/tests/testparser/testparserhandler.ml
```

Table 7.1: Automatic Testing Programs

testscanner.ml was built to test our scanner and testparser.ml was for our parser testing. testparserhandler.ml provides ast lists with information returned from our parser to support for testing conducted by testparser.ml. Code for these above programs will be presented in our appendix part. We also wrote unit test programs in our language – lga to work with our automatic testing system. Details of those testing programs are illustrated as following:

`test_obj.lga`, `test_function.lga` and `test_nest.lga` were built for testing our scanner.

`test_obj.lga`

```
square = {
    str : "\thello world!\tI'm a square"
    run : circle
```

```
      pos : [-2.5e1, 5]
      vec : [1, 1]
      delay : 5
  }
```

will produce

```
IDENTIFIER<square>
ASSIGN
LBRACE
INDENT
IDENTIFIER<str>
COLON
STRING<"\thello world!\tI'm a square">
TERMINATOR
IDENTIFIER<run>
COLON
IDENTIFIER<circle>
TERMINATOR
IDENTIFIER<pos>
COLON
LBK
NUM<-2.5e1>
COMMA
NUM<5>
RBK
TERMINATOR
IDENTIFIER<vec>
COLON
LBK
NUM<1>
COMMA
NUM<1>
RBK
TERMINATOR
IDENTIFIER<delay>
COLON
NUM<5>
OUTDENT
TERMINATOR
RBRACE
EOF
```

test_function.lga

```
  move_forward = fun(l) ->
      if @pos && @vec
      a = math.atan(@vec[0], @vec[1])
      @pos[0] = math.cos(angle) * l
      @pos[1] = math.sin(angle) * l
  return
```

will produce

```
IDENTIFIER<move_forward>
ASSIGN
FUN
LPAREN
IDENTIFIER<l>
RPAREN
ARROW
INDENT
IF
THIS
IDENTIFIER<pos>
AND
THIS
IDENTIFIER<vec>
TERMINATOR
IDENTIFIER<a>
ASSIGN
IDENTIFIER<math>
DOT
IDENTIFIER<atan>
LPAREN
THIS
IDENTIFIER<vec>
LBK
NUM<0>
RBK
COMMA
THIS
IDENTIFIER<vec>
LBK
NUM<1>
RBK
RPAREN
TERMINATOR
THIS
IDENTIFIER<pos>
LBK
NUM<0>
RBK
ASSIGN
```

```
IDENTIFIER<math>
DOT
IDENTIFIER<cos>
LPAREN
IDENTIFIER<angle>
RPAREN
TIMES
IDENTIFIER<l>
TERMINATOR
THIS
IDENTIFIER<pos>
LBK
NUM<1>
RBK
ASSIGN
IDENTIFIER<math>
DOT
IDENTIFIER<sin>
LPAREN
IDENTIFIER<angle>
RPAREN
TIMES
IDENTIFIER<l>
OUTDENT
TERMINATOR
RETURN
TERMINATOR
EOF
```

test_nest.lga

```
if(a == 1)
    if (b and c)
        d = a + b
        while (a isnt 0)
            b = a * 2
            a = a - 1
    a = d/3
return
```

will produce

```
IF
LPAREN
IDENTIFIER<a>
EQ
NUM<1>
```

```
RPAREN
INDENT
IF
LPAREN
IDENTIFIER<b>
AND
IDENTIFIER<c>
RPAREN
INDENT
IDENTIFIER<d>
ASSIGN
IDENTIFIER<a>
PLUS
IDENTIFIER<b>
TERMINATOR
WHILE
LPAREN
IDENTIFIER<a>
NEQ
NUM<0>
RPAREN
INDENT
IDENTIFIER<b>
ASSIGN
IDENTIFIER<a>
TIMES
NUM<2>
TERMINATOR
IDENTIFIER<a>
ASSIGN
IDENTIFIER<a>
MINUS
NUM<1>
OUTDENT
OUTDENT
TERMINATOR
IDENTIFIER<a>
ASSIGN
IDENTIFIER<d>
DIVIDE
NUM<3>
OUTDENT
TERMINATOR
RETURN
TERMINATOR
EOF
```

We also wrote a few lga programs to test our parser functionalities. We present some lga test programs here together with output produced in our testing system.

`test_code.lga`

```
fun () ->
    print(a)
fun (a, b) ->
    c = a + b
return c + 1
```

will produce

```
Body
ExpressionLine
CodeExpression
Code
ParamList
Block
Body
ExpressionLine
InvocationExpression
Invocation
AssignableValue
IdentifierAssignable
Identifier
Literal
print
Arguments
ArgList
ValueExpression
AssignableValue
IdentifierAssignable
Identifier
Literal
a
ExpressionLine
CodeExpression
Code
ParamList
Identifier
Literal
a
Identifier
Literal
b
Block
```

```
Body
ExpressionLine
AssignExpression
Assign
IdentifierAssignable
Identifier
Literal
c
OperationExpression
Binop
ValueLop
AssignableValue
IdentifierAssignable
Identifier
Literal
a
OP
ValueExpression
AssignableValue
IdentifierAssignable
Identifier
Literal
b
StatementLine
ReturnStatement
Return
OperationExpression
Binop
ValueLop
AssignableValue
IdentifierAssignable
Identifier
Literal
c
OP
ValueExpression
LiteralValue
Literal
1
```

test_assign.lga

```
a = (1 + 2)
a = b = c = 1
@a = 3
```

will produce

```
Body
ExpressionLine
AssignExpression
Assign
IdentifierAssignable
Identifier
Literal
a
ValueExpression
ParentheticalValue
Parenthetical
Body
ExpressionLine
OperationExpression
Binop
ValueLop
LiteralValue
Literal
1
OP
ValueExpression
LiteralValue
Literal
2
ExpressionLine
AssignExpression
Assign
IdentifierAssignable
Identifier
Literal
a
AssignExpression
Assign
IdentifierAssignable
Identifier
Literal
b
AssignExpression
Assign
IdentifierAssignable
Identifier
Literal
c
ValueExpression
LiteralValue
Literal
```

```
1
ExpressionLine
AssignExpression
Assign
ThisPropertyAssignable
ThisProperty
Identifier
Literal
a
ValueExpression
LiteralValue
Literal
3
```

test_array.lga

```
a = [1, 2, 3]
b = a[1]
c = [a, 2, b+1]
```

will produce

```
Body
ExpressionLine
AssignExpression
Assign
IdentifierAssignable
Identifier
Literal
a
ValueExpression
AssignableValue
ArrayAssignable
Array
ArgList
ValueExpression
LiteralValue
Literal
1
ValueExpression
LiteralValue
Literal
2
ValueExpression
LiteralValue
Literal
3
```

```
ExpressionLine
AssignExpression
Assign
IdentifierAssignable
Identifier
Literal
b
ValueExpression
AssignableValue
ValueAccessorAssignable
AssignableValue
IdentifierAssignable
Identifier
Literal
a
IndexAccessor
Index
IndexValue
Literal
1
ExpressionLine
AssignExpression
Assign
IdentifierAssignable
Identifier
Literal
c
ValueExpression
AssignableValue
ArrayAssignable
Array
ArgList
ValueExpression
AssignableValue
IdentifierAssignable
Identifier
Literal
a
ValueExpression
LiteralValue
Literal
2
OperationExpression
Binop
ValueLop
AssignableValue
```

```
IdentifierAssignable
Identifier
Literal
b
OP
ValueExpression
LiteralValue
Literal
1
```

## 7.2 Demo Test

To test our whole compiler system, we wrote some lga programs to make animation as well as calculation.

testjs.lga

```
gcd = fun (a,b) ->
     return a when !b
    return gcd(b, a % b)

console.log(gcd(98, 21))
```

This program was used to test the ability our compiler transfer lga into runnable javascript code. It is a program looking for greatest common dividen for 98 and 21 which should be 7. During our testing, this program ran perfectly and returned a value of 7.

testlga.lga

```
rect = {
   type : Rectangle
   size : [100, 100]
   pos : [300, 300]
   scale : 0.8
   fill : "#FF8000"
   stroke : "orangered"
   rotate : fun (t) -> @rotation = (t % 60) * Math.PI / 60.0
   move : fun() -> @translation.x = @translation.x + 1
}

circle = {
   type : Circle
   size : [100]
   pos : [500, 300]
   scale : 0.9
   fill : "#0000FF"
   stroke : "black"
   move : fun() ->
               t = fun (o) ->
                   o.y  = o.y + 1
                   if o.y > 1080
                       o.y = 0
               return t(@translation)
}

run = fun (t) ->
   rect.move()
   rect.rotate(t)
   circle.move()
```

This program defines a orange rectangle together with a blue circle. The orange rectangle will keep rotating and moving from left to right. The build circle will move from upside to the bottom and repeat that when it disappear at the bottom. During our demonstration, That code generated the animation we expected which proved our compiler system could make animation based on programs written in lga.

# Chapter 8

# Lesson Learned

Overall, the whole team had a great time together creating this fun project. Addition to compiler and computer language knowledge, we also learn how to cooperate on working on a fairly complicated software.

## 8.1 Hang Qian

There's no easy job to be a team director. During the whole developing process, I learned that the best way to help the team and keep everything on track is try to be coordinator rather than a dictator. Though I am the one who makes the decision when we have disagreement, this rarely happened. For most of time, reasoning a decision is always better than just making the call.

Technically speaking, I would suggest that always get familiar with your tools earlier. For this project, it's Ocaml and git. I spent most early developing time getting familiar with every corner of Ocaml and later some detais of the language did save us a huge amount of time so that we can manage to pull out what we originally want to do 100 percently. Also, don't bother to perfect everything at the beginning because it's almost inevitable that you have to change something or compromise along the way. Done is always better than perfect, which means get your "hello world" running first before you bother if you want anonymous functions.

I'm lucky enough to work with 3 intelligent and dedicated men and woman. This always give a good mood while we stuck in somewhere. So last but not least, pick your teammates wisely.

## 8.2 Yuanli Dong

As a CS student, I have been working on several team projects, this project is among the best ones. Although the workload of this project is tough, I really enjoyed it, thanks to my great teammates.

Other than saying the lesson I learned it that we should start early, like most of the teams that took this course in the last few years, I would say you should get "determined" earlier. It is not only about the tools and languages you will use, it's also about determining the details of the language you are designing. When we start doing the project, one teammate of mine and I are looking at the Parser component of the project, while we were not 100 percent sure about the grammar, we made our assumption and start working on our own. This turned out to be a waste of time because we have to change it from begin to end after seeing too much errors in it. So when you are doing this project, try to discuss about it early and definitely try to perfect it, it will save you a lot of time.

The final advice is that it's a good idea to plan something fun after the project, it will boost your working efficiency greatly. I first proposed a hot pot gathering after the project, and it works well.

## 8.3 Pindan Hao

I would say this project is the coolest one I have ever worked on, and the process is efficiency and enjoyable. Special thanks to every teammate!

One important thing I have learned is that although it is alway hard to distinguish whether its a waste of time or we could do it if persist, do not say "impossible" too early. Cool things are always created among the minority and sometimes you could manage it by thinking outside the box. I almost gave up on it because Ocamllex said no to returning tokens without any lexing terms, and parser said no again to a list of tokens–finally we made it. So glad that we persisted on the python-like indenting style, which makes our LGA so neat!

What's more important, choose cool teammates (on which I was lucky), then choose a fun topic to work on together (on which we did), work hard and have fun!

## 8.4 Tian Xia

I must say, this is the most exciting academic project I have ever been working on. To be honest, this is not a easy project, instead of following the way most past-year teams have stepped, we explored a new design and finally made it work.

Starting earlier, like almost every past-year team said, is a truly important thing which could let you have more space for implementing a great idea. Starting earlier also means you should be familiar with the language you will use as well as the structure you decide to go with for your system. Another important thing I would say is have a runnable model in hand and keep moving forward to your dream result. Having a runnable model will make you at least have something to submit. But that will never be enough, keep improving it, like a real geek, all the way to something you think you will be proud of.

And another important thing will be, have someone cool and intelligent to work with (I'm really lucky to have my teammates).

# Chapter 9

# Appendix

```
{
  open Parser
  open Utils
  open Lexing
  let _indent_stack = Stack.create()
}

let punc = ['~' '`' '!' '@' '#' '$' '%' '^' '&' '*' '(' ')' '-' '+' '=' ',' '.' '?' '/' '<' '>' ':' '''  ';' '{' '}' '[' ']' '|' ' ']
let spacetab = [' ' '\t']*
let newline = ['\r' '\n']+
let letter = ['a'-'z' 'A'-'Z']
let digit = ['0'-'9']
let id = ('_'|letter) (letter|digit|'_')*
let exp = 'e'('+'|'-')?['0'-'9']+
let num = '-'? (digit)+ ('.'? (digit)* exp?|exp)
let string = '"' (letter|digit|punc|"\\t")*  '"'

rule token = parse
    | newline     { incr_linenum lexbuf; indent lexbuf }
    | [' ' '\t']        { token lexbuf }
    | '#'               { comment lexbuf }
    | '@'               { THIS }
    | '='               { ASSIGN }
    | '('               { LPAREN }
    | ')'           { RPAREN }
    | '{'           { LBRACE }
    | '}'           { RBRACE }
    | '['       { LBK }
    | ']'       { RBK }
    | ','           { COMMA }
    | ':'           { COLON }
    | '='           { ASSIGN }
    | '+'       { PLUS }
    | '-'       { MINUS }
    | '*'       { TIMES }
    | '/'       { DIVIDE }
    | '%'       { MOD }
    | '+'       { PLUS }
    | '<'           { LT }
    | '>'           { GT }
    | '.'       { DOT }
    | '!'       { NOT }
        | "not"               { NOT }
    | "=="        { EQ }
        | "is"                { EQ}
    | "!="        { NEQ }
        | "isnt"              { NEQ }
    | "<="        { LEQ }
    | ">="        { GEQ }
        | "&&"                { AND }
    | "and"       { AND }
        | "||"                { OR }
    | "or"        { OR }
    | "->"          { ARROW }
    | "false" as lxm    { BOOL(lxm) }
    | "true" as lxm   { BOOL(lxm) }
    | "if"          { IF }
        | "when"              { WHEN }
    | "else"        { ELSE }
    | "return"      { RETURN }
    | "while"       { WHILE }
    | "for"       { FOR }
    | "in"        { FORIN }
    | "break"     { STATEMENT("break") }
    | "continue"    { STATEMENT("continue") }
        | "fun"               { FUN }
    | id as lxm         { IDENTIFIER(lxm) }
    | num as lxm        { NUM(lxm) }
    | string as lxm   { STRING(lxm) }
    | eof               { EOF }
    | _ as char         { raise (Failure("SCANNER: illegal input"^
    Char.escaped char)) }

and indent = parse
       | spacetab as s
        {
       let len = (String.length s) in
       let top_pos = (Stack.top _indent_stack) in
       if len > top_pos then
         begin
         Stack.push len _indent_stack;
         INDENT
         end
       else if len = top_pos then TERMINATOR
       else
         let _count = (Utils.outdent_count len _indent_stack) in
         if _count = -1 then raise (Failure("SCANNER: wrong indent
    on line "^string_of_int lexbuf.lex_curr_p.pos_lnum))
         else OUTDENT_COUNT(_count)
      }

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


{
  Stack.push 0 _indent_stack

}
```

Listing 9.1: scanner.mll

```
%{open Ast %}

%token ASSIGN TERMINATOR RETURN INDENT OUTDENT
%token ARROW LBK RBK LBK RBK
%token LPAREN RPAREN PLUS MINUS TIMES DIVIDE
%token EQ NEQ MOD AND OR LT LEQ GT GEQ NOT COLON
%token LBRACE RBRACE COMMA DOT BOOL THIS
%token IF ELSE WHILE FOR FORIN WHEN
%token <string> IDENTIFIER
%token <string> NUM
%token <string>  STRING
%token <string> STATEMENT
%token <int> OUTDENT_COUNT
%token <string> NULL
%token <string> BOOL
%token FUN
%token EOF

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

%start root
%type <Ast.root> root

%%

root:
  | /* nothing */ { [] }
  | body { $1 }
  | block TERMINATOR { $1 }

body:
  | line { [$1] }
  | body TERMINATOR line { List.append $1 [$3] }
  | body TERMINATOR { $1 }

line:
  | expression { ExpressionLine($1) }
  | statement { StatementLine($1) }
```

```
statement:
  | return { ReturnStatement($1) }
  | STATEMENT { LiteralStatement(Literal($1)) }

return:
  | RETURN expression { Return($2) }

expression:
  | value { ValueExpression($1) }
  | invocation { InvocationExpression($1) }
  | code { CodeExpression($1) }
  | operation { OperationExpression($1) }
  | assign { AssignExpression($1) }
  | iftype { IfExpression($1) }
  | whiletype { WhileExpression($1) }
  | fortype { ForExpression($1) }

operation:
  | lop PLUS expression   { Binop($1, Plus,$3) }
  | lop MINUS expression    { Binop($1, Minus, $3) }
  | lop TIMES expression    { Binop($1, Times, $3) }
  | lop DIVIDE expression   { Binop($1, Divide, $3)}
  | lop EQ  expression    { Binop($1, Eq, $3) }
  | lop NEQ expression    { Binop($1, Neq, $3) }
  | lop MOD expression    { Binop($1, Mod, $3) }
  | lop AND expression    { Binop($1, And, $3) }
  | lop OR expression       { Binop($1, Or, $3) }
  | lop LT expression       { Binop($1, Less, $3) }
  | lop LEQ expression    { Binop($1, Leq, $3) }
  | lop GT expression       { Binop($1, Greater, $3) }
  | lop GEQ expression    { Binop($1, Geq, $3) }
  | NOT expression      { Neg($2) }

lop:
  | value { ValueLop($1) }
  | invocation { InvocationLop($1) }

code:
  | FUN LPAREN paramList RPAREN ARROW block { BlockCode($3, $6) }
    | FUN LPAREN paramList RPAREN ARROW expression { ExpressionCode
    ($3, $6) }

value:
    | assignable { AssignableValue($1) }
  | literal { LiteralValue($1) }
  | parenthetical { ParentheticalValue($1) }

assign:
  | assignable ASSIGN expression                { Assign ($1, $3) }
  | assignable ASSIGN TERMINATOR expression     { Assign ($1, $4) }
  | assignable ASSIGN INDENT expression OUTDENT { Assign ($1, $4) }

assignable:
  | array { ArrayAssignable($1) }
  | obj { ObjAssignable($1) }
  | identifier { IdentifierAssignable($1) }
  | thisProperty { ThisPropertyAssignable($1) }
  | value accessor { ValueAccessorAssignable($1, $2) }
  | invocation accessor { InvocationAccessorAssignable($1, $2) }

assignObj:
  | objAssignable COLON expression { AssignObj($1, $3) }
  | objAssignable COLON INDENT expression OUTDENT { AssignObj($1,
    $4) }

objAssignable:
  | identifier { IdentifierObjAssignable($1)}
  | thisProperty { ThisPropertyObjAssignable($1) }

accessor:
  | DOT identifier { DotAccessor($2) }
  | index { IndexAccessor($1) }

obj:
    | LBRACE assignList TERMINATOR RBRACE { Object($2) }
    | LBRACE assignList RBRACE { Object($2) }

assignList:
  | /* nothing */ { [] }
  | assignObj { [$1] }
  | assignList TERMINATOR assignObj { List.append $1 [$3] }
/*  | assignList TERMINATOR assignObj { List.append $1 [$4] } */
  | assignList INDENT assignList OUTDENT { List.append $1 $3 }

invocation:
  | value arguments { Invocation($1, $2) }

arguments:
  | LPAREN RPAREN { [] }
  | LPAREN argList RPAREN { $2 }

argList:
  | expression { [$1] }
  | argList COMMA expression { List.append $1 [$3] }
  | argList COMMA TERMINATOR expression { List.append $1 [$4] }
  | INDENT argList OUTDENT { $2 }
  | argList COMMA INDENT argList OUTDENT { List.append $1 $4 }

paramList:
  | /* nothing */   { [] }
  | identifier { [$1] }
  | paramList COMMA identifier { List.append $1 [$3] }
  | paramList COMMA TERMINATOR identifier { List.append $1 [$4] }
  | paramList COMMA INDENT paramList OUTDENT {List.append $1 $4 }

index:
  | LBK indexValue RBK { Index($2) }

indexValue:
  | NUM { Literal($1) }

parenthetical:
  | LPAREN body RPAREN { Parenthetical($2) }
  | LPAREN INDENT body OUTDENT RPAREN { Parenthetical($3) }
```

```

array:
    | LBK RBK { [] }
    | LBK argList RBK { $2 }

block:
  | INDENT OUTDENT {[]}
  | INDENT body OUTDENT { $2 }

identifier:
  | IDENTIFIER  { Literal($1) }

literal:
  | NUM   { Literal($1) }
  | STRING  { Literal($1) }
  | NULL    { Literal($1) }
  | BOOL    { Literal($1) }

thisProperty:
    | THIS identifier { $2}

iftype:
  | ifBlock           { IfOnly($1) }
  | ifBlock ELSE block  { IfElse($1, $3) }
    | statement WHEN expression { PostIfStatement($3, $1) }

ifBlock:
  | IF expression block          { IfBlock($2, $3) }
  | ifBlock ELSE IF expression block  { IfBlockSeq($1, $4, $5) }

whiletype:
  | whileSource block   { While($1, $2) }

whileSource:
  | WHILE expression    { WhileSource($2) }

fortype:
  | forBody block     { For($1, $2) }

forBody:
  | forStart forSource    { ForBody($1, $2) }

forStart:
  | FOR forVar      { ForStart($2) }

forVar:
    | identifier      { ForVar($1) }

forSource:
  | FORIN expression    { ForSource($2) }
```
Listing 9.2: parser.mly

```
open Printf
open Ast
open Scanner
open Parser
open Utils
open Lga

let makestr = fun a ->
  String.concat "" a

let op_to_string = fun a ->
  match a with
  | Plus -> "+"
  | Minus -> "-"
  | Times -> "*"
  | Divide -> "/"
  | Eq -> "=="
  | Neq -> "!="
  | Mod -> "%"
  | And -> "&&"
  | Or -> "||"
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | Not -> "!"


let handle_op a =
  op_to_string a

let handle_literal a =
  match a with
  | Literal(x) -> x

let handle_identifier a =
  handle_literal a

let handle_index_value a =
  handle_literal a

let handle_index a =
  match a with
  | Index(x) -> "[" ^ handle_index_value x ^ "]"

let handle_accessor a =
  match a with
  | DotAccessor(x) -> "." ^ (handle_identifier x)
  | IndexAccessor(x) -> handle_index x

let handle_arg_list f a =
  String.concat ", " (List.map f a)

let handle_array f a =
  "[" ^ (handle_arg_list f a) ^ "]"

let handle_this_property a =
  "this." ^ (handle_identifier a)

let handle_obj_assignable a =
  match a with
  | IdentifierObjAssignable(x) -> handle_identifier x
  | ThisPropertyObjAssignable(x) -> handle_this_property x

let handle_assign_obj f a =
  match a with
  | AssignObj(x, y) -> (handle_obj_assignable x) ^ " : " ^ (f y)

let handle_assign_list f a =
  String.concat ",\n" (List.map (handle_assign_obj f) a)

let handle_obj f a =
  match a with
  | Object(x) -> "{\n" ^ (handle_assign_list f x) ^ "\n}"

let handle_return f a =
  match a with
  | Return(x) -> "return " ^ (f x)

let handle_statement f a =
  match a with
  | ReturnStatement(x) -> handle_return f x
  | LiteralStatement(x) -> handle_literal x

let handle_line f a =
  match a with
  | ExpressionLine(x) -> f x
  | StatementLine(x) -> handle_statement f x

let rec explode = function
    "" -> []
  | s  -> (String.get s 0) ::
          explode (String.sub s 1 ((String.length s) - 1))

let rec implode = function
    []        -> ""
  | charlist -> (String.make 1 (List.hd charlist)) ^
                (implode (List.tl charlist))

let rec remove x =
  match x with
  | a :: (b :: c)  ->
    if a == '}' && b == ';' then a :: (remove c)
    else if a == ';' && b == ')' then b :: (remove c)
    else a :: (remove (List.tl x))
  | _ -> x

let add_semicom = fun a ->
  let len = String.length a in
    if a.[len-1] = '}' then a else a ^ ";"

let handle_body f a =
  String.concat "\n" (List.map add_semicom (List.map (handle_line f
    ) a))

let handle_parenthetical f a =
  match a with
  | Parenthetical(x) -> implode (remove ( explode ("(" ^ (
    handle_body f x) ^ ")")))

let handle_arguments f a =
  "(" ^ (handle_arg_list f a) ^ ")"

let handle_invocation fe fv a =
  match a with
  | Invocation(x, y) -> (fv x) ^ (handle_arguments fe y)

let handle_assignable fe fv a =
  match a with
  | ArrayAssignable(x) -> handle_array fe x
  | ObjAssignable(x) -> handle_obj fe x
  | IdentifierAssignable(x) -> handle_identifier x
  | ThisPropertyAssignable(x) -> handle_this_property x
  | ValueAccessorAssignable(x, y) -> (fv x) ^ (handle_accessor y)
  | InvocationAccessorAssignable(x, y) -> (handle_invocation fe fv
    x) ^ (handle_accessor y)

let rec handle_value f a =
  match a with
  | AssignableValue(x) -> handle_assignable f (handle_value f) x
  | LiteralValue(x) -> handle_literal x
  | ParentheticalValue(x) -> handle_parenthetical f x

let handle_lop f a =
  match a with
  | ValueLop(x) -> handle_value f x
  | InvocationLop(x) -> handle_invocation f (handle_value f) x

let handle_operation f a =
  match a with
  | Binop(x, y, z) -> (handle_lop f x) ^ " " ^ (handle_op y) ^ " "
    ^ (f z)
  | Neg(x) -> "!" ^ (f x)

let handle_block f a =
  handle_body f a

let handle_param_list a =
  String.concat ", " (List.map handle_identifier a)

let handle_code f a =
  match a with
  | BlockCode(x, y) -> let paraliststr = handle_param_list x in
                       let blockstr = handle_block f y in
                       "function (" ^ paraliststr ^ ")\n{\n" ^
    blockstr ^ "\n}"
  | ExpressionCode(x, y) -> let paraliststr = handle_param_list x
    in
                                let exprstr = f y in
                                "function (" ^ paraliststr ^ ") { " ^
    exprstr ^"}"


let handle_assign f a =
  match a with
  | Assign(x, y) ->
     (handle_assignable f (handle_value f) x) ^ " = " ^ (f y)

let rec handle_if_block f a =
  match a with
  | IfBlock(x, y) -> "if ( " ^ (f x) ^ " ) \n{\n" ^ (handle_block f
     y) ^ "\n}"
  | IfBlockSeq(x, y, z) ->
    (handle_if_block f x) ^ "else if ( " ^ (f y) ^ " ) \n{\n" ^ (
    handle_block f z) ^ "\n}"

let handle_if f a =
  match a with
  | IfElse(x, y) -> (handle_if_block f x) ^ " else {\n" ^ (
    handle_block f y) ^ "\n}"
  | IfOnly(x) -> handle_if_block f x
  | PostIfStatement(x, y) -> "if ("^ (f x) ^") { "^ (
    handle_statement f y) ^ "}\n"

let handle_while_source f a =
  match a with
  | WhileSource(x) -> f x

let handle_while f a =
  match a with
  | While(x,y) -> let whilesourcestr = handle_while_source f x in
                  let blockstr = handle_block f y in
                  makestr ["while ("; whilesourcestr; ")\n{\n "
    ;blockstr;"\n}"]

let handle_for_var a =
  match a with
  | ForVar(x) -> let idstr = handle_identifier x in
            idstr

let handle_for_start a =
  match a with
  | ForStart(x) ->  handle_for_var x


let handle_for_source f a =
  match a with
  | ForSource(x) -> f x


let handle_for_body f a =
  match a with
  | ForBody(x, y) -> let forstartstr = handle_for_start x in
                     let forsourcestr = handle_for_source f y in
                     makestr ["("; forstartstr;" in ";forsourcestr;
    ")"]

let handle_for f a =
  match a with
  | For(x, y) -> let forbodystr = handle_for_body f x in
                 let blockstr = handle_block f y in
                 makestr ["for ";forbodystr;"\n{\n "; blockstr; "\n
     }"]

let rec handle_expr a =
  match a with
  | ValueExpression(x) -> (handle_value handle_expr x)
  | InvocationExpression(x) -> (handle_invocation handle_expr (
    handle_value handle_expr) x)
  | CodeExpression(x) -> (handle_code handle_expr x)
  | OperationExpression(x) -> (handle_operation handle_expr x)
  | AssignExpression(x) -> (handle_assign handle_expr x)
  | IfExpression(x) -> handle_if handle_expr x
  | WhileExpression(x) -> handle_while handle_expr x
  | ForExpression(x) -> handle_for handle_expr x

let handle_top_level_expr a =
  match a with
  | ValueExpression(x) -> (handle_value handle_expr x) ^ ";"
  | InvocationExpression(x) -> (handle_invocation handle_expr (
    handle_value handle_expr) x) ^ ";"
  | CodeExpression(x) -> (handle_code handle_expr x)
  | OperationExpression(x) -> (handle_operation handle_expr x) ^ ";
    "
  | AssignExpression(x) -> (handle_assign handle_expr x) ^ ";"
  | IfExpression(x) -> handle_if handle_expr x
  | WhileExpression(x) -> handle_while handle_expr x
  | ForExpression(x) -> handle_for handle_expr x




let handle_root f body =
  handle_body f body

let rec explode = function
    "" -> []
  | s  -> (String.get s 0) ::
          explode (String.sub s 1 ((String.length s) - 1))

let rec implode = function
    []       -> ""
  | charlist -> (String.make 1 (List.hd charlist)) ^
                 (implode (List.tl charlist))

let rec remove x =
  match x with
  | a :: (b :: c)  ->
    if a == '}' && b == ';' then a :: (remove c)
    else a :: (remove (List.tl x))
  | _ -> x

(* Line handler.
 * We take AssignExpresion line into account, store it to
 * lga datastructure for code generating
 * f: function to handle expression in general
 * a: line
 * return a var_lga
 *)
let handle_top_line fe ft a =
  match a with
  | ExpressionLine(line) ->
     begin match line  with
           | AssignExpression (x) -> ft fe x
           | _ -> Ignore
     end
  | _ -> Ignore

(* Body is a list of lines, which is the top level of LGA.
 * We only take AssignExpression line into account
 * f: function to handle expression in general
 * a: body
 * return a list of var_lga
 *)
let handle_top_body fe ft a =
  List.map (handle_top_line fe ft) a

(* top level obj handler
 * return top_id_list
 *)
let handle_top_obj f obj =
    match obj with
    | Object(o) ->
        let construct aobj =
        (* return a TopId(string, string) *)
          match aobj with
          | AssignObj(x, y) -> TopId(handle_obj_assignable x, (f y))
        in
        TopIdList(List.map construct o)

(* top level handler
 * f: expression handler
 * a: input line (Which is a AssignExpresion)
 * return a var_lga based on the type
 *)
let handle_top_level f a =
  match a with
  | Assign(left, right) ->
     let id =
       begin match left with
         | IdentifierAssignable(y) ->
            handle_identifier y
         | _ -> raise(Failure("Only allow identifier as variable"))
       end
     in
     begin match right with
             | ValueExpression(AssignableValue(ObjAssignable(obj)))
     ->
                 ObjVar_lga(TopObj(id, (handle_top_obj f obj)))
             | _ -> IdVar_lga(TopId(id, (f right)))
     end



(* return a list of var_lga *)
```

```
let handle_top_root f body =
  handle_top_body f handle_top_level body

let lga_of_file filename =
  let root = ast_of_file Parser.root Scanner.token filename in
  handle_top_root handle_top_level_expr root

let js_of_file filename =
  let root = ast_of_file Parser.root Scanner.token filename in
  let b = handle_root handle_expr root in
  implode (remove (explode b))
```

Listing 9.3: semantic.ml language

```
open Lga
open Ast
open Parser
open Semantic
open List
open String

type output = Js | Lga

let lga_translate_type a =
  match a with
  | "Rectangle" -> "two.makeRectangle"
  | "Circle" -> "two.makeCircle"
  | _ as field-> "two.make"^ (String.sub field 0 ((String.length
    field) - 1))

let lga_get_by_field field a =
  match a with
  | TopId(k, v) -> if k = field then true else false

let lga_find_field field l =
  try
    let target = find (lga_get_by_field field) l in
    match target with
    | TopId(x, y) ->
       try
       String.sub y 1 ((String.length y) - 3)
       with
       | Invalid_argument(x) -> "0, 0"
  with
  | Not_found -> "0, 0"

let rec lga_obj_type id a =
  match a with
  | h :: t ->
     begin match h with
           | TopId(key, value) ->
              begin match key with
                    | "type" ->
                       let pos = lga_find_field "pos" t in
                       let size = lga_find_field "size"  t in
                       let objtype = lga_translate_type value in
                       (id^" = "^objtype^"("^pos^", "^size^");\n")
```

```
                        ::(lga_obj_type id t)
                     | "pos" -> lga_obj_type id t
                     | "size" -> lga_obj_type id t
                     | _ as field -> (id^"."^field^" = "^value^"\n")
     :: lga_obj_type id t
                end
      end
    | _ -> []


let lga_top_id a =
  match a with
  | TopId(key, value) ->
      "var "^key^" = "^value

(* return [ "var ID;" "ID.key = value;" "var ID_KEY = function ..."
     ...]*)
let lga_top_obj a =
  match a with
  | TopObj(id, alist) ->
      concat ""
             (append ["var ";id;";\n"]
                     begin match alist with
                           | TopIdList(list) -> (lga_obj_type id
    list)
                     end)


let lga_generate_code a =
  match a with
  | ObjVar_lga(x) -> lga_top_obj x
  | IdVar_lga(x) -> lga_top_id x
  | _ -> ""

let gen_code filename t =
  match t with
  | Lga ->
      let lga = lga_of_file filename in
      String.concat "\n" (List.map lga_generate_code lga)
  | Js ->
      js_of_file filename

let temp_head =
  "var elem = document.getElementById('lga-div');
   var params = { width: 1080, height: 1024 };
   var two = new Two(params).appendTo(elem);
   var run = function(f) {}"

let temp_tail =
  "two.bind('update', run).play();"

let get_mode m =
  if m then Lga else Js

let mode = ref true
let in_file = ref "test.lga"
let out_file = ref "lga.js"
```

```ocaml
let get_in_name x = in_file := x

let spec = [
  ("-js", Arg.Clear mode, "Enable Javascript mode");
  ("-o", Arg.Set_string out_file, "Output file");
]
let usage = "lgac: [-js] [-o outputfile] inputfile"

let gen_output file t channel =
  match t with
  | Lga ->
     Printf.fprintf channel "%s\n%s\n%s\n" temp_head (gen_code file
     t) temp_tail
  | _ -> Printf.fprintf channel "%s\n" (gen_code file t)

let _ =
  Arg.parse spec get_in_name usage;
  let oc = open_out !out_file in
  gen_output !in_file (get_mode !mode) oc;
  close_out oc
```

Listing 9.4: codegen.ml

```ocaml
type op = Plus | Minus | Times | Divide | Eq | Neq | Mod | And | Or
     | Less | Leq | Greater | Geq | Not

type literal = Literal of string

type identifier = literal

type indexValue = literal

type index = Index of indexValue

type accessor =
    DotAccessor of identifier
  | IndexAccessor of index

type 'a argList = 'a list

type 'a array = 'a argList

type thisProperty = identifier

type objAssignable =
    IdentifierObjAssignable of identifier
  | ThisPropertyObjAssignable of thisProperty

type 'a assignObj = AssignObj of objAssignable * 'a

type 'a assignList = 'a assignObj list

type 'a obj = Object of 'a assignList

type 'a return = Return of 'a

type 'a statement =
    ReturnStatement of 'a return
  | LiteralStatement of literal

type 'a line =
    ExpressionLine of 'a
  | StatementLine of 'a statement

type 'a body = ('a line) list

type 'a parenthetical = Parenthetical of 'a body

type 'a arguments = 'a argList

type ('expr, 'value) invocation = Invocation of 'value * 'expr
    arguments

type ('expr, 'value) assignable =
    ArrayAssignable of 'expr array
  | ObjAssignable of 'expr obj
  | IdentifierAssignable of identifier
  | ThisPropertyAssignable of thisProperty
  | ValueAccessorAssignable of 'value * accessor
  | InvocationAccessorAssignable of ('expr, 'value) invocation *
    accessor

type 'a value =
    AssignableValue of ('a, 'a value) assignable
  | LiteralValue of literal
  | ParentheticalValue of 'a parenthetical

type 'a lop =
    ValueLop of 'a value
  | InvocationLop of ('a, 'a value) invocation

type 'a operation =
    Binop of 'a lop * op * 'a
  | Neg of 'a

type 'a block = 'a body

type paramList = identifier list

type 'a code =
    BlockCode of paramList * 'a block
  | ExpressionCode of paramList * 'a

type 'a assign = Assign of ('a, 'a value) assignable * 'a

type 'a ifBlock =
    IfBlock of 'a * 'a block
  | IfBlockSeq of 'a ifBlock * 'a * 'a block

type 'a iftype =
    IfElse of 'a ifBlock * 'a block
  | IfOnly of 'a ifBlock
  | PostIfStatement of 'a * 'a statement

```

```
type 'a whileSource = WhileSource of 'a

type 'a whiletype = While of 'a whileSource * 'a block

type 'a forSource = ForSource of 'a

type forVar = ForVar of identifier

type forStart = ForStart of forVar

type 'a forBody = ForBody of forStart * 'a forSource

type 'a fortype  = For of 'a forBody * 'a block

type expression =
    ValueExpression of expression value
  | InvocationExpression of (expression, expression value)
    invocation
  | CodeExpression of expression code
  | OperationExpression of expression operation
  | AssignExpression of expression assign
  | IfExpression of expression iftype
  | WhileExpression of expression whiletype
  | ForExpression of expression fortype

type root = expression body
```

Listing 9.5: ast.mli

```
type shape = Rect | Circle

(* id = id *)
type top_id = TopId of string * string

type top_id_list = TopIdList of top_id list
(* id = {
         [ string : string,
           string : string,
                  .
                  .
                  .
         ]
        }
 *)
type top_obj = TopObj of string * top_id_list

(* id = function (...) {...} *)
type var_lga =
    ObjVar_lga of top_obj
  | IdVar_lga of top_id
  | Ignore

type lga = var_lga list
```

Listing 9.6: lga.mli

```
open Lexing
open Parser
open List
open Utils

let string_list_of_input_file file =
  let lexbuf = Lexing.from_channel (open_in file) in
  let token_list = token_list_of_lexbuf lexbuf Scanner.token Parser
    .EOF in
  let stringify x = Printf.sprintf "%s" (string_of_token x) in
  List.map stringify token_list

let string_list_of_output_file file =
  let lines = ref [] in
  let ic = open_in file in
  try
    while true; do
      lines := input_line ic :: !lines
    done; []
  with End_of_file ->
    close_in ic;
    List.rev !lines

(* Compare the tokens list generated by input file, with the
    expected output (outfile) *)
let test_file infile outfile =
  let in_list = (string_list_of_input_file infile) in
  let out_list = (string_list_of_output_file outfile) in
  let equal = fun x -> (fst x) = (snd x) in
  if List.length in_list = List.length out_list then
  List.for_all equal (List.combine in_list out_list)
  else (print_endline "Token list length doesn't match."; false)

let _ =
  print_endline "\n*** SCANNER TEST ***\n";
  let indir = Sys.argv.(1) in
  let outdir = Sys.argv.(2) in
  test_dir test_file indir outdir
```

Listing 9.7: test_scanner

```
open Printf
open Ast
open Scanner
open Parser
open Utils
open Test_parser_handler

let check = ref true

let string_list_of_file file =
  let lines = ref [] in
  let ic = open_in file in
  try
   while true; do
     lines := input_line ic :: !lines
   done; []
  with End_of_file ->
   close_in ic;
   List.rev !lines

let string_list_of_input_file file =
  let ast = get_ast_list file in
    ast

let test_file ast expc =
  let equal = fun x -> (fst x) = (snd x) in
  if List.for_all equal (List.combine ast expc) then true
  else false

let file_filter = fun dir ->
  let infile_array = Sys.readdir dir in
  let raw_file_list = Array.to_list infile_array in
  let is_target = fun file ->
     if (String.sub file 0 4) = "test" then true
     else begin
      false end
  in
  List.filter is_target raw_file_list

let print_result = fun ast expc indir infile->
  let filename = String.concat "" [indir;infile] in
  if((test_file ast expc) = true) then Printf.sprintf "Test %s PASS
    !" filename
  else begin
    Printf.sprintf "Test %s FAIL!" filename;
  end

let test_dir indir outdir =
  let in_files = (file_filter indir) in
  let test_file_fun = fun infile  ->
    let ast = string_list_of_input_file (abs_input_testfile indir
    infile) and
        expc = string_list_of_file (abs_output_testfile_of_input
    outdir infile) in
  print_endline (print_result ast expc indir infile)
  in
  List.map test_file_fun in_files
  (*if check then print_endline "ALL PASS!"*)

let _ =
  let indir = Sys.argv.(1) in
  let outdir = Sys.argv.(2) in
  test_dir indir outdir
```

Listing 9.8: test_parser

```
open Printf

open Ast
open Scanner
open Parser
open Utils
```

```
type senario = ID

let rs = ref []

let handle_op a =
  rs := List.append !rs ["OP"]

let handle_literal a =
  match a with
  | Literal(x) -> rs := List.append !rs ["Literal"];
                  rs := List.append !rs [x]

let handle_identifier a =
  rs := List.append !rs ["Identifier"];
  handle_literal a


let handle_index_value a =
  rs := List.append !rs ["IndexValue"];
  handle_literal a

let handle_index a =
  match a with
  | Index(x) -> rs := List.append !rs ["Index"];
                handle_index_value x

let handle_accessor a =
  match a with
  | DotAccessor(x) -> rs := List.append !rs ["DotAccessor"];
                      handle_identifier x
  | IndexAccessor(x) -> rs:= List.append !rs ["IndexAccessor"];
                        handle_index x

let handle_arg_list f a =
  rs := List.append !rs ["ArgList"];
  List.iter f a

let handle_array f a =
  rs := List.append !rs ["Array"];
  handle_arg_list f a

let handle_this_property a =
  rs := List.append !rs ["ThisProperty"];
  handle_identifier a

let handle_obj_assignable a =
  match a with
  | IdentifierObjAssignable(x) -> rs := List.append !rs ["
    IdentifierObjAssignable"];
                                  handle_identifier x
  | ThisPropertyObjAssignable(x) -> rs := List.append !rs ["
    ThisPropertyObjAssignable"];
                                    handle_this_property x

let handle_assign_obj f a =
  match a with
  | AssignObj(x, y) -> rs := List.append !rs ["AssignObj"];
```

```
                        handle_obj_assignable x;
                        f y

let handle_assign_list f a =
  rs := List.append !rs ["AssignList"];
  List.iter (handle_assign_obj f) a

let handle_obj f a =
  match a with
  | Object(x) -> rs := List.append !rs ["Object"];
                  handle_assign_list f x

let handle_return f a =
  match a with
  | Return(x) -> rs := List.append !rs ["Return"];
                  f x

let handle_statement f a =
  match a with
  | ReturnStatement(x) -> rs := List.append !rs ["ReturnStatement"
    ];
                           handle_return f x
  | LiteralStatement(x) -> rs := List.append !rs ["LiteralStatement
    "];
                            handle_literal x

let handle_line f a =
  match a with
  | ExpressionLine(x) -> rs := List.append !rs ["ExpressionLine"];
                          f x
  | StatementLine(x) -> rs := List.append !rs ["StatementLine"];
                        handle_statement f x

let handle_body f a =
  rs := List.append !rs ["Body"];
  List.iter (handle_line f) a

let handle_parenthetical f a =
  match a with
  | Parenthetical(x) -> rs := List.append !rs ["Parenthetical"];
                        handle_body f x

let handle_arguments f a =
  rs := List.append !rs ["Arguments"];
  handle_arg_list f a

let handle_invocation fe fv a =
  match a with
  | Invocation(x, y) -> rs := List.append !rs ["Invocation"];
                        fv x;
                        handle_arguments fe y

let handle_assignable fe fv a =
  match a with
  | ArrayAssignable(x) -> rs := List.append !rs ["ArrayAssignable"
    ];
                           handle_array fe x
  | ObjAssignable(x) -> rs := List.append !rs ["ObjAssignable"];
                        handle_obj fe x
  | IdentifierAssignable(x) -> rs := List.append !rs ["
    IdentifierAssignable"];
                               handle_identifier x
  | ThisPropertyAssignable(x) -> rs := List.append !rs ["
    ThisPropertyAssignable"];
                                 handle_this_property x
  | ValueAccessorAssignable(x, y) -> rs := List.append !rs ["
    ValueAccessorAssignable"];
                                     fv x;
                                     handle_accessor y
  | InvocationAccessorAssignable(x, y) -> rs := List.append !rs ["
    InvocationAccessorAssignable"];
                                          handle_invocation fe fv x
    ;
                                          handle_accessor y


let rec handle_value f a =
  match a with
  | AssignableValue(x) -> rs := List.append !rs ["AssignableValue"
    ];
                          handle_assignable f (handle_value f) x
  | LiteralValue(x) -> rs := List.append !rs ["LiteralValue"];
                       handle_literal x
  | ParentheticalValue(x) -> rs := List.append !rs ["
    ParentheticalValue"];
                             handle_parenthetical f x

let handle_lop f a =
  match a with
  | ValueLop(x) -> rs := List.append !rs ["ValueLop"];
                   handle_value f x
  | InvocationLop(x) -> rs := List.append !rs ["InvocationLop"];
                        handle_invocation f (handle_value f) x

let handle_operation f a =
  match a with
  | Binop(x, y, z) -> rs := List.append !rs ["Binop"];
                      handle_lop f x;
                      handle_op y;
                      f z
  | Neg(x) -> rs := List.append !rs ["Neg"];
              f x

let handle_block f a =
  rs := List.append !rs ["Block"];
  handle_body f a

let handle_param_list a =
  rs := List.append !rs ["ParamList"];
  List.iter handle_identifier a

let handle_code f a =
  match a with
  | Code(x, y) -> rs := List.append !rs ["Code"];
```

```
                   handle_param_list x;
                   handle_block f y

let handle_assign f a =
  match a with
  | Assign(x, y) -> rs := List.append !rs ["Assign"];
                     handle_assignable f (handle_value f) x;
                     f y

let rec handle_if_block f a =
  match a with
  | IfBlock(x, y) ->  rs := List.append !rs ["IfBlock"];
                       f x;
                   handle_block f y
  | IfBlockSeq(x, y, z) -> rs := List.append !rs ["IfBlockSeq"];
                            handle_if_block f x;
                            f y;
                            handle_block f z

let handle_if f a =
  match a with
  | IfElse(x, y) -> rs := List.append !rs ["IfElse"];
                     handle_if_block f x;
                  handle_block f y
  | IfOnly(x) -> rs := List.append !rs ["IfOnly"];
                  handle_if_block f x

let handle_while_source f a =
  match a with
  | WhileSource(x) -> rs := List.append !rs ["WhileSource"];
                       f x

let handle_while f a =
  match a with
  | While(x,y) -> rs := List.append !rs ["While"];
                   handle_while_source f x;
                   handle_block f y

let handle_for_source f a =
  match a with
  | ForSource(x) -> f x

let handle_for_var a =
  match a with
  | ForVar(x) -> rs := List.append !rs ["ForVar"];
                  handle_identifier x

let handle_for_start a =
  match a with
  | ForStart(x) ->  rs := List.append !rs ["ForStart"];
                     handle_for_var x

let handle_for_body f a =
  match a with
  | ForBody(x, y) -> rs := List.append !rs ["ForBody"];
                      handle_for_start x;
                      handle_for_source f y
```

```

let handle_for f a =
  match a with
  | For(x, y) -> rs := List.append !rs ["For"];
                 handle_for_body f x;
                 handle_block f y

let rec handle_expr a =
  match a with
  | ValueExpression(x) -> rs := List.append !rs ["ValueExpression"
    ];
                          handle_value handle_expr x
  | InvocationExpression(x) -> rs := List.append !rs ["
    InvocationExpression"];
                               handle_invocation handle_expr (
    handle_value handle_expr) x
  | CodeExpression(x) -> rs := List.append !rs ["CodeExpression"];
                         handle_code handle_expr x
  | OperationExpression(x) -> rs := List.append !rs ["
    OperationExpression"];
                              handle_operation handle_expr x
  | AssignExpression(x) -> rs := List.append !rs ["AssignExpression
    "];
                           handle_assign handle_expr x
  | IfExpression(x) -> rs := List.append !rs ["IfExpression"];
                       handle_if handle_expr x
  | WhileExpression(x) -> rs := List.append !rs ["WhileExpression"
    ];
                          handle_while handle_expr x
  | ForExpression(x) -> rs := List.append !rs ["ForExpression"];
                        handle_for handle_expr x

let handle_root f body =
  rs := [];
  handle_body f body

let get_ast_list filename =
  let root = ast_of_file Parser.root Scanner.token filename in
  let b = handle_root handle_expr root in
  !rs
```

Listing 9.9: test_parserhandler

```
(** The Util module is a grouping of commonly-needed functions. *)

open Ast
open Parser
open Printf
open List
open Lexing
let log x = print_endline x

(** Match tokens to strings reflecting their names. *)
let string_of_token : Parser.token -> string = function
  | IDENTIFIER(x) ->
     Printf.sprintf "IDENTIFIER<%s>" x
  | NUM(x) ->
```

```
      Printf.sprintf "NUM<%s>" x
  | STATEMENT(x) ->
      Printf.sprintf "STATEMENT<%s>" x
  | BOOL(x) ->
      Printf.sprintf "BOOL<%s>" x
  | STRING(x) ->
      Printf.sprintf "STRING<%s>" x
  | ASSIGN ->
      "ASSIGN"
  | TERMINATOR ->
      "TERMINATOR"
  | RETURN ->
      "RETURN"
  | FUN ->
      "FUN"
  | ARROW ->
      "ARROW"
  | LBK ->
      "LBK"
  | RBK ->
      "RBK"
  | LPAREN ->
      "LPAREN"
  | RPAREN ->
      "RPAREN"
  | LBRACE ->
      "LBRACE"
  | RBRACE ->
      "RBRACE"
  | PLUS ->
      "PLUS"
  | MINUS ->
      "MINUS"
  | TIMES ->
      "TIMES"
  | DIVIDE ->
      "DIVIDE"
  | EQ ->
      "EQ"
  | NEQ ->
      "NEQ"
  | MOD ->
      "MOD"
  | AND ->
      "AND"
  | OR ->
      "OR"
  | LT ->
      "LT"
  | LEQ ->
      "LEQ"
  | GT ->
      "GT"
  | GEQ ->
      "GEQ"
  | NOT ->
      "NOT"
  | COLON ->
      "COLON"
  | COMMA ->
      "COMMA"
  | DOT ->
      "DOT"
  | NULL(x) ->
      "NULL"
  | THIS ->
      "THIS"
  | IF ->
      "IF"
  | WHEN ->
      "WHEN"
  | ELSE ->
      "ELSE"
  | WHILE ->
      "WHILE"
  | FOR ->
      "FOR"
  | FORIN ->
      "FORIN"
  | INDENT ->
      "INDENT"
  | OUTDENT_COUNT(x) ->
      Printf.sprintf "OUTDENT<%d>" x
  | OUTDENT ->
      "OUTDENT"
  | EOF ->
      "EOF"

(** Returns the number of outdent(s) that occur at the
    beginning of one line according to the number of
    white spaces and a stack. The stack sequentially holds
    the size of pre-occurred indents that haven't been
    matched yet by outdents. the function is called when
    the new line has at least one outdent. The size of
    indent has to match some smaller level and the function
    returns a value equal or larger than 1, otherwise
    returns -1. *)
let outdent_count = fun len stack ->
  let rec helper inc =
    if (Stack.top stack) > len then
      begin
        Stack.pop stack;
        helper (inc + 1)
      end
    else if (Stack.top stack) < len then -1
    else inc
  in helper 0

(** Returns all the test files' filenames in the directory dir. *)
let raw_testfile_list_of_dir = fun dir ->
  let infile_array = Sys.readdir dir in
  let raw_file_list = Array.to_list infile_array in
  let is_test_file file =
    if (String.sub file 0 4) = "test" then true else false in
  List.filter is_test_file raw_file_list

(** Returns input files' path by concatenating directory name indir
     and filename. *)
let abs_input_testfile = fun indir filename->
  String.concat "" [indir;filename]

(** Returns expected output files' path by concatenating directory
    name outdir and filename. *)
let abs_output_testfile_of_input = fun outdir filename->
  String.concat "" [outdir;filename;".out"]

(** Print the PASS/FAIL result of a test file. Testfun is the
    function that returns testing results for input file infile and
     its expected output file outfile. *)
let get_test_result_string = fun testfun infile outfile ->
  let result = (testfun infile outfile) in
  if result then Printf.sprintf "Test %s PASS!" infile
  else Printf.sprintf "Test %s FAIL!" infile

(** Print the PASS/FAIL results for all input test files in a
    directory indir with their expected output files in directory
    outdir. *)
let test_dir test_fun indir outdir =
    let raw_list = raw_testfile_list_of_dir indir in
    let print = fun file ->
      let abs_in = abs_input_testfile indir file in
      let abs_out = abs_output_testfile_of_input outdir file in
      print_endline (get_test_result_string test_fun abs_in abs_out
    )
  in
  List.map print raw_list

(** Return a new list with int token OUTDENT_COUNT in the list
    expanded into cooresponding number of OUTDENT token(s), and
    TERMINATOR token(s) added to every line break. *)
let rec expand_token_list = fun list ->
  let rec expand_token = fun token ->
    match token with
    | INDENT -> [token]
    | OUTDENT_COUNT(x) ->
       let rec populate token =
         match token with
         | OUTDENT_COUNT(x) ->
            if x = 1 then [OUTDENT]
            else OUTDENT :: (populate (OUTDENT_COUNT(x-1)))
         | _ -> [token] in
       List.append (populate token) [TERMINATOR]
    | _ -> [token] in
  let rec add_terminator = fun list ->
    match list with
    | OUTDENT :: (TERMINATOR :: (ELSE :: t)) -> OUTDENT :: (ELSE ::
     (add_terminator t))
    | [] -> []
    | _ -> (hd list) :: (add_terminator (tl list))
  in
  add_terminator (List.flatten (List.map expand_token list))
```

```

(** Use tokenizer to process lexbuf and return the generated token
    list. *)
let token_list_of_lexbuf lexbuf tokenizer stopsign =
  let rec helper lexbuf list =
    let token = tokenizer lexbuf in
    if token = stopsign then (List.append list [token])
    else token :: (helper lexbuf list)
  in expand_token_list (helper lexbuf [])


let ast_of_token_list myparser tokenlist =
  let list = ref tokenlist in
  let tokenize lexbuf =
    match !list with
    | [] -> Parser.EOF
    | h :: t -> log (string_of_token h); list := t; h
  in
  myparser tokenize (Lexing.from_string "")

(** Parse a file with myparser and tokenizer and return a list
    which represents the ast. *)
let ast_of_file myparser tokenizer filename =
  let lexbuf = Lexing.from_channel (open_in filename) in
  let token_list = ref (token_list_of_lexbuf lexbuf tokenizer
    Parser.EOF) in
  let tokenize lexbuf =
    match !token_list with
    | [] -> Parser.EOF
    | h :: t -> token_list := t; h
  in
  myparser tokenize (Lexing.from_string "")

(** Increment line number information of lexbuf. *)
let incr_linenum lexbuf =
  let pos = lexbuf.lex_curr_p in
  lexbuf.lex_curr_p <- { pos with
    pos_lnum = pos.pos_lnum + 1;
    pos_bol = pos.pos_cnum;
  }
```

Listing 9.10: utils

```
open Lexing
open Parser
open List
open Utils

let string_list_of_input_file file =
  let lexbuf = Lexing.from_channel (open_in file) in
  let token_list = token_list_of_lexbuf lexbuf Scanner.token Parser
    .EOF in
  let stringify x = Printf.sprintf "%s" (string_of_token x) in
  List.map stringify token_list

let _ =
  let filename = Sys.argv.(1) in
  let l = string_list_of_input_file filename in
  List.iter print_endline l
```

Listing 9.11: token

```
CC = ocamlc -rectypes
YACC = ocamlyacc
LEX = ocamllex

PARSER = parser
SCANNER = scanner
AST = ast
SEMANTIC = semantic
LGA = lga
UTILS = utils
CODEGEN = codegen

all: codegeno
  $(CC) $(UTILS).cmo $(PARSER).cmo $(SCANNER).cmo $(SEMANTIC).cmo $
    (CODEGEN).cmo -o ../../$(LGA)c
utilso: parsero
  $(CC) -c $(UTILS).ml
scannero: scanner utilso
  $(CC) -c $(SCANNER).ml
scanner:
  $(LEX) $(SCANNER).mll
parsero: parseri
  $(CC) -c $(PARSER).ml
parseri: parser asti
  $(CC) -c $(PARSER).mli
parser:
  $(YACC) $(PARSER).mly
asti:
  $(CC) -c $(AST).mli
semantico: asti lgai parsero scannero
  $(CC) -c $(SEMANTIC).ml
lgai:
  $(CC) -c $(LGA).mli
codegeno: semantico
  $(CC) -c $(CODEGEN).ml
token: scannero parsero
  $(CC) -c token.ml
  $(CC) $(UTILS).cmo $(PARSER).cmo $(SCANNER).cmo token.cmo -o
    token

clean:
  rm -rf *.cmo *.cmi $(LGA) $(SCANNER).mli $(SCANNER).ml $(PARSER).
    mli $(PARSER).ml token
```

Listing 9.12: Makefile