LGA

Qian, Hao, Dong & Xia

Motivation

- Fun -> stands for joy... maybe lambda
- Coolness ->

  Painless graphic and animation

Language for Graphic & Animation

concise syntax

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

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

Language for Graphic & Animation

Intuitive semantic

```
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
}
```

Language for Graphic & Animation

Full-compatible with Javascript

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

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

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

Scanner

OUTDENT/TERMINATOR Handling

- `%token  <int> OUTDENT_COUNT`
- `OUTDENT_COUNT(x) = [OUTDENT, …, OUTDENT]`
- `TERMINAL: also not directly come from lexing items`

Scanner

“Preprocessing” lexbuf

```
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 ParserEOF) in
  let fake_tokenizer lexbuf =
    match !token_list with
    | [] -> Parser.EOF
    | h :: t -> token_list := t; h
  in
  myparser fake_tokenize (Lexing.from_string"")
```

Parser

```
type  invocation = Invocatior of value * expr
...
type assignable =
    ValueAccessorAssignabl of value * accessor
  | InvocationAccessorAssignabl of invocatior * accessor
...
type value =
    AssignableValu of assignable
  | LiteralValu of literal
  | ParentheticalValu of parenthetical
...
type expression =
    ValueExpressior of value
  | InvocationExpressior of invocation
  | AssignExpressior of assign
 ...
```

Parser

```
type ('expr, 'value) invocation = Invocation of 'value * 'expr arguments
...
type ('expr, 'value) assignable =
    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 expression =
    ValueExpression of expression value
  | InvocationExpression of (expression, expression value) invocation
  | AssignExpression of expression assign
```

Semantics

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

let handle_assignable a =
  match a with
   ...
  | InvocationAccessorAssignable(x, y) -> (handle_invocation x) ^ (handle_accessor y)

let rec handle_value a =
  match a with
   ...
  | AssignableValue(x) -> handle_assignable (handle_value x)

let rec handle_expr a =
  match a with
   ...
  | InvocationExpression(x) -> handle_invocation x
```

Semantics

```
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
   ...
  | 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

let rec handle_expr a =
  match a with
   ...
  | InvocationExpression(x) -> handle_invocation handle_expr (handle_value handle_expr) x
```

Code generation

- LGA ->
  LGA_Runtime + Javascript + HTML5 graphic API

- also support fully compatible JS mode

```
lgac -js <INPUT> -o <OUTPUT>
```

Future work

- Syntax
  - Array Comprehension?

```
a = [i*2 for i in [1,2,3]]
```

  - Splats?

```
race = fun (winner, runners…) -> print runners
race(“Stephen”, “Hang”, “Pinddan”)
> Hang Pindan
```

- LGA
  - Core library in LGA

謝謝！

Demo time …