# IPL

## Image Processing Language?

Jianning Yue
Wookyun Kho
Young Jin Yoon

AGL : Animation  applet Generation Language

# Contents

- IPL?
- Advantages
- Syntax
- Development
- Examples
- Lessons Learned

# IPL?

- IPL is not Image Processing Language
  - Now this is an

**Animation applet**

**Generation**

**Language!**

# Advantages (1/2)

- IPL provide very flexible image handling
  - Provide fundamental operation for image as expression
    - Rotate  (@ operator)
    - Translate (‘ operator)
    - Scale (^ operator)
  - Provide animate() function to produce an animated Image
  - Provide coord type to handle coordinates

# Advantages (2/2)

- Easy to learn
  - C like syntax and scope
  - Easily-recognized operator
    - (^ is power operator from another language)
- Productive
  - Can be exported as an JAVA applet
    - smaller than GIF Animated Image

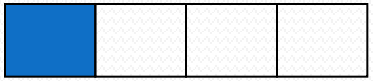
# Syntax : Types (1/3)

- Four types in IPL
  - number
  - image
  - coord
  - bool
- Optional declarator in IPL
  - [] for array definitions

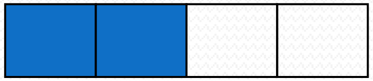
# Syntax : Types (2/3)

- **FLEXIBLE ARRAY HANDLING (1/2)**

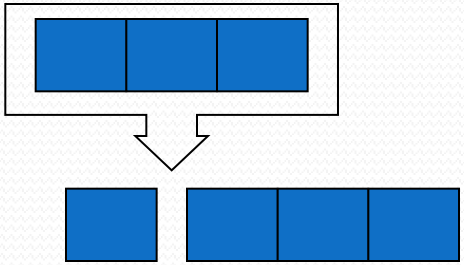
- For both
  - imgA[0]



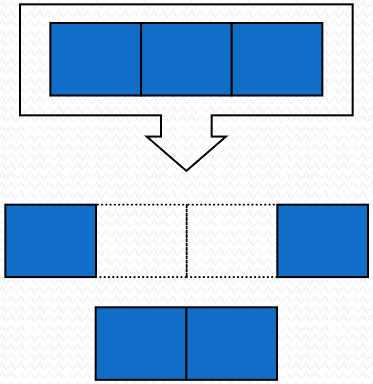
imgA[0~1]



- For lvalue
  - imgA[1+]

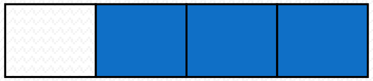


imgA[1~2+]

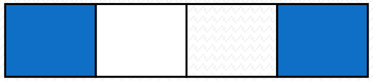


# Syntax : Types (3/3)

- **FLEXIBLE ARRAY HANDLING (2/2)**
  - For rvalue
    - imgA[0-]



    - imgA[1~2-]



# Syntax : Expr (1/4)

- Basic image operator
  - imgA = imgA @ numA; // rotate operator
  - imgA = imgA ^ numA; // scale operator
  - imgA = imgA ` coordA; // set operator
  - imgA = imgA : numA; // alpha operator
  - imgA = imgA $ imgB; // concat operator

# Syntax : Expr (2/4)

- Basic bool operator
  - booA = numA > numB;    // gt operator
  - booA = numA < numB;    // lt operator
  - booA = numA >= numB;    // ge operator
  - booA = numA <= numB;    // le operator
  - booA = numA == numB;    // eq operator
  - booA = numbooA != numbooB;    // neq
  - booA = !booA    // not operator

# Syntax : Expr (3/4)

- Basic arithmetic operator
  - numA = numA * numB;          // multiply
  - numA = numA / numB;          // division
  - numA = numA % numB;          // modulo
  - numA = numA + numB;          // plus
  - numA = numA - numB;          // minus
- For coord, there is no operation. However we can still handle this. How?

# Syntax : Expr (4/4)

- For coordination
  - cooA = (xof(cooA),numA);
  - cooB = (numA, yof(cooA));

- By providing xof() and yof(), we can still maintain flexibility without any complexibility!

# Syntax : Stmt

- Providing while, if statement just as almost same as C's statement definition.
    - Except using {} for single statement.

- You can define a function using defunc keywords.
    - defunc foo (number A, number B) number C

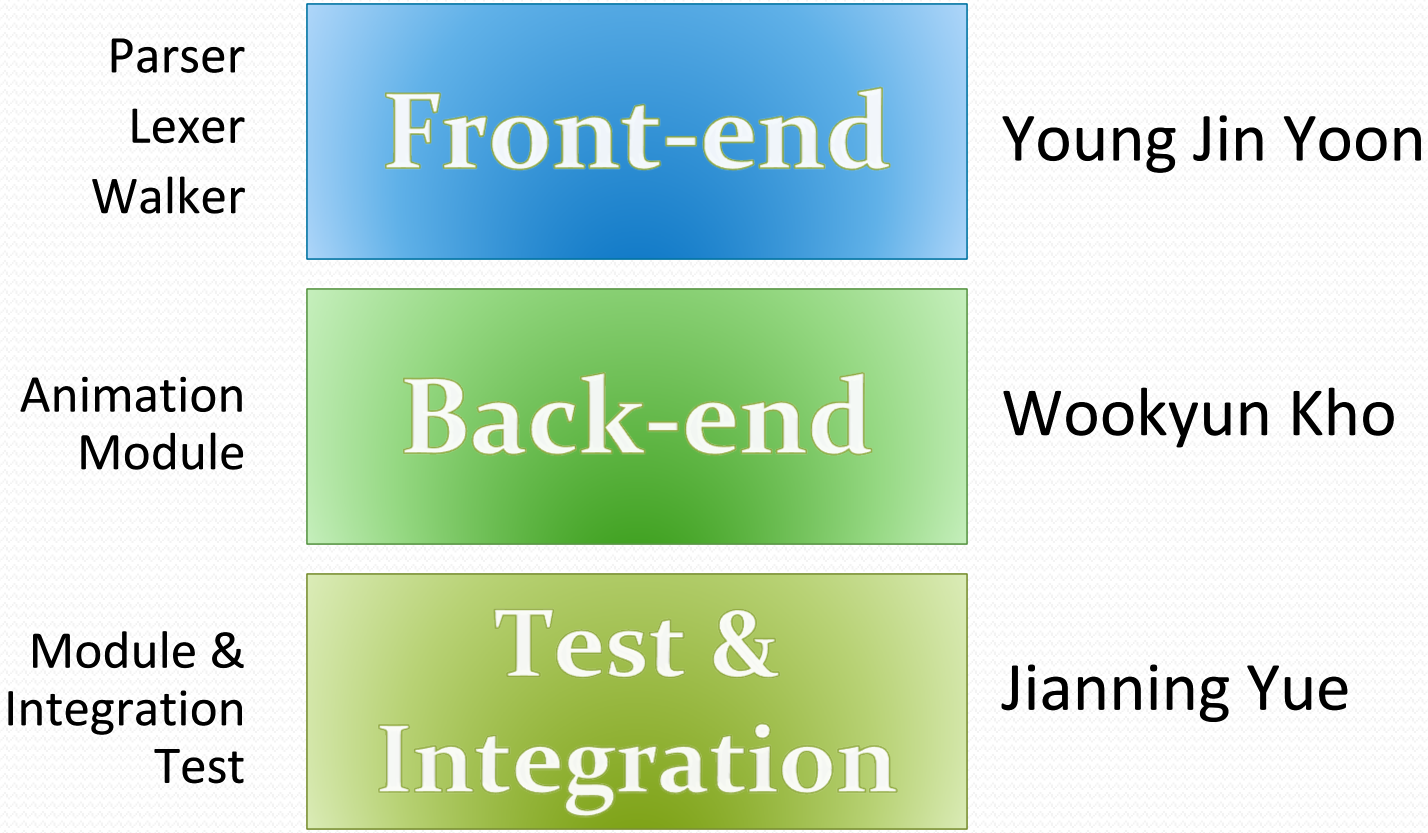
{ C = A + B; }

- Providing return, break, continue statements.

# Development

- Task Distribution
- Architecture Overview
- Implementation
- Test and Debug plan

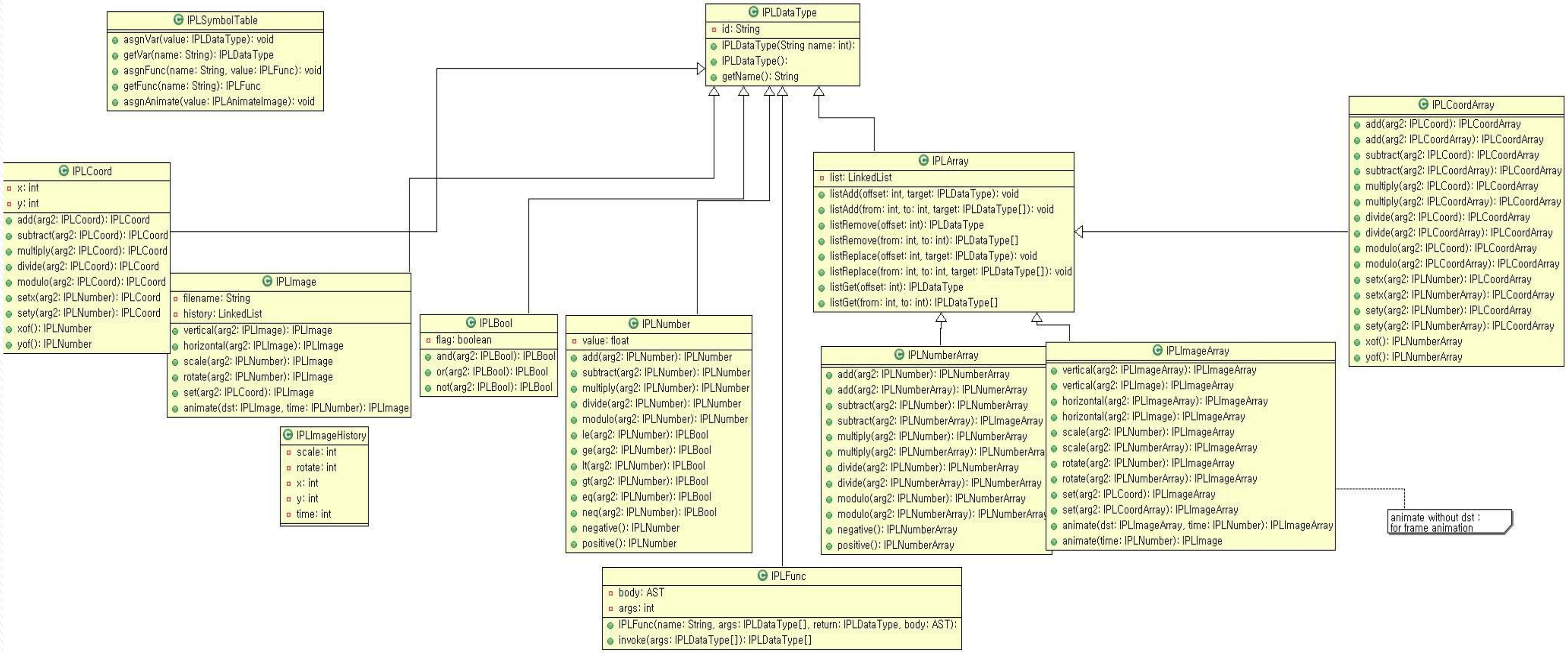
# Task distribution



| | | |
|---|---|---|
| Parser<br>Lexer<br>Walker | Front-end | Young Jin Yoon |
| Animation Module | Back-end | Wookyun Kho |
| Module & Integration Test | Test & Integration | Jianning Yue |

# Architecture Overview

- Used UML Class Diagram



IPLSymbolTable
asgnVar(value: IPLDataType): void
getVar(name: String): IPLDataType
asgnFunc(name: String, value: IPLFunc): void
getFunc(name: String): IPLFunc
asgnAnimate(value: IPLAnimateImage): void

IPLDataType
id: String
IPLDataType(String name: int):
IPLDataType():
getName(): String

IPLCoord
x: int
y: int
add(arg2: IPLCoord): IPLCoord
subtract(arg2: IPLCoord): IPLCoord
multiply(arg2: IPLCoord): IPLCoord
divide(arg2: IPLCoord): IPLCoord
modulo(arg2: IPLCoord): IPLCoord
setx(arg2: IPLNumber): IPLCoord
sety(arg2: IPLNumber): IPLCoord
xof(): IPLNumber
yof(): IPLNumber

IPLImage
filename: String
history: LinkedList
vertical(arg2: IPLImage): IPLImage
horizontal(arg2: IPLImage): IPLImage
scale(arg2: IPLNumber): IPLImage
rotate(arg2: IPLNumber): IPLImage
set(arg2: IPLCoord): IPLImage
animate(dst: IPLImage, time: IPLNumber): IPLImage

IPLImageHistory
scale: int
rotate: int
x: int
y: int
time: int

IPLBool
flag: boolean
and(arg2: IPLBool): IPLBool
or(arg2: IPLBool): IPLBool
not(arg2: IPLBool): IPLBool

IPLNumber
value: float
add(arg2: IPLNumber): IPLNumber
subtract(arg2: IPLNumber): IPLNumber
multiply(arg2: IPLNumber): IPLNumber
divide(arg2: IPLNumber): IPLNumber
modulo(arg2: IPLNumber): IPLNumber
le(arg2: IPLNumber): IPLBool
ge(arg2: IPLNumber): IPLBool
lt(arg2: IPLNumber): IPLBool
gt(arg2: IPLNumber): IPLBool
eq(arg2: IPLNumber): IPLBool
neq(arg2: IPLNumber): IPLBool
negative(): IPLNumber
positive(): IPLNumber

IPLFunc
body: AST
args: int
IPLFunc(name: String, args: IPLDataType[], return: IPLDataType, body: AST):
invoke(args: IPLDataType[]): IPLDataType[]

IPLArray
list: LinkedList
listAdd(offset: int, target: IPLDataType): void
listAdd(from: int, to: int, target: IPLDataType[]): void
listRemove(offset: int): IPLDataType
listRemove(from: int, to: int): IPLDataType[]
listReplace(offset: int, target: IPLDataType): void
listReplace(from: int, to: int, target: IPLDataType[]): void
listGet(offset: int): IPLDataType
listGet(from: int, to: int): IPLDataType[]

IPLNumberArray
add(arg2: IPLNumber): IPLNumberArray
add(arg2: IPLNumberArray): IPLNumerArray
subtract(arg2: IPLNumber): IPLNumberArray
subtract(arg2: IPLNumberArray): IPLImageArray
multiply(arg2: IPLNumber): IPLNumberArray
multiply(arg2: IPLNumberArray): IPLNumberArra
divide(arg2: IPLNumber): IPLNumberArray
divide(arg2: IPLNumberArray): IPLNumberArray
modulo(arg2: IPLNumber): IPLNumberArray
modulo(arg2: IPLNumberArray): IPLNumberArray
negative(): IPLNumberArray
positive(): IPLNumberArray

IPLImageArray
vertical(arg2: IPLImageArray): IPLImageArray
vertical(arg2: IPLImage): IPLImageArray
horizontal(arg2: IPLImageArray): IPLImageArray
horizontal(arg2: IPLImage): IPLImageArray
scale(arg2: IPLNumber): IPLImageArray
scale(arg2: IPLNumberArray): IPLImageArray
rotate(arg2: IPLNumber): IPLImageArray
rotate(arg2: IPLNumberArray): IPLImageArray
set(arg2: IPLCoord): IPLImageArray
set(arg2: IPLCoordArray): IPLImageArray
animate(dst: IPLImageArray, time: IPLNumber): IPLImageArray
animate(time: IPLNumber): IPLImage

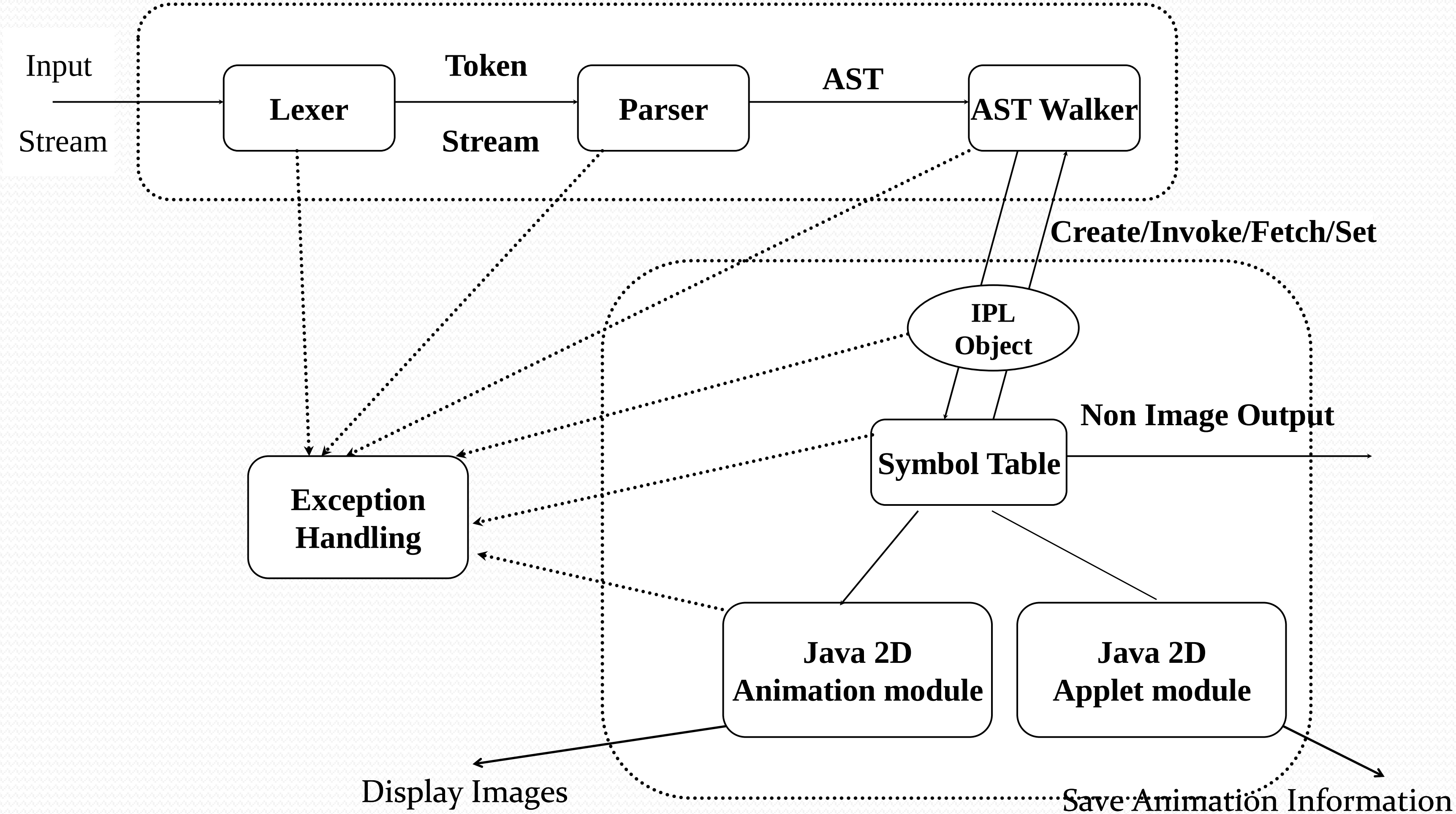
animate without dst :
for frame animation

IPLCoordArray
add(arg2: IPLCoord): IPLCoordArray
add(arg2: IPLCoordArray): IPLCoordArray
subtract(arg2: IPLCoord): IPLCoordArray
subtract(arg2: IPLCoordArray): IPLCoordArray
multiply(arg2: IPLCoord): IPLCoordArray
multiply(arg2: IPLCoordArray): IPLCoordArray
divide(arg2: IPLCoord): IPLCoordArray
divide(arg2: IPLCoordArray): IPLCoordArray
modulo(arg2: IPLCoord): IPLCoordArray
modulo(arg2: IPLCoordArray): IPLCoordArray
setx(arg2: IPLNumber): IPLCoordArray
setx(arg2: IPLNumberArray): IPLCoordArray
sety(arg2: IPLNumber): IPLCoordArray
sety(arg2: IPLNumberArray): IPLCoordArray
xof(): IPLNumberArray
yof(): IPLNumberArray

# Implementation (1/3)

- ANTLR
  - Parser
  - Lexer
  - Walker
- Animation Module
  - Animation Displaying Engine
  - Animation Applet Code

# Implementation (2/3)



# Implementation (3/3)

- Animation Applet
  - If you do “export”,  you have to specify the filename.
  - Ex) export to “IPLoutput.ipl”

```
<applet code=IPLApplet.class  width=1024 height=600>
<param name="fps" value="20">
<param name="ipl" value="IPLoutput.ipl">
</applet>
```

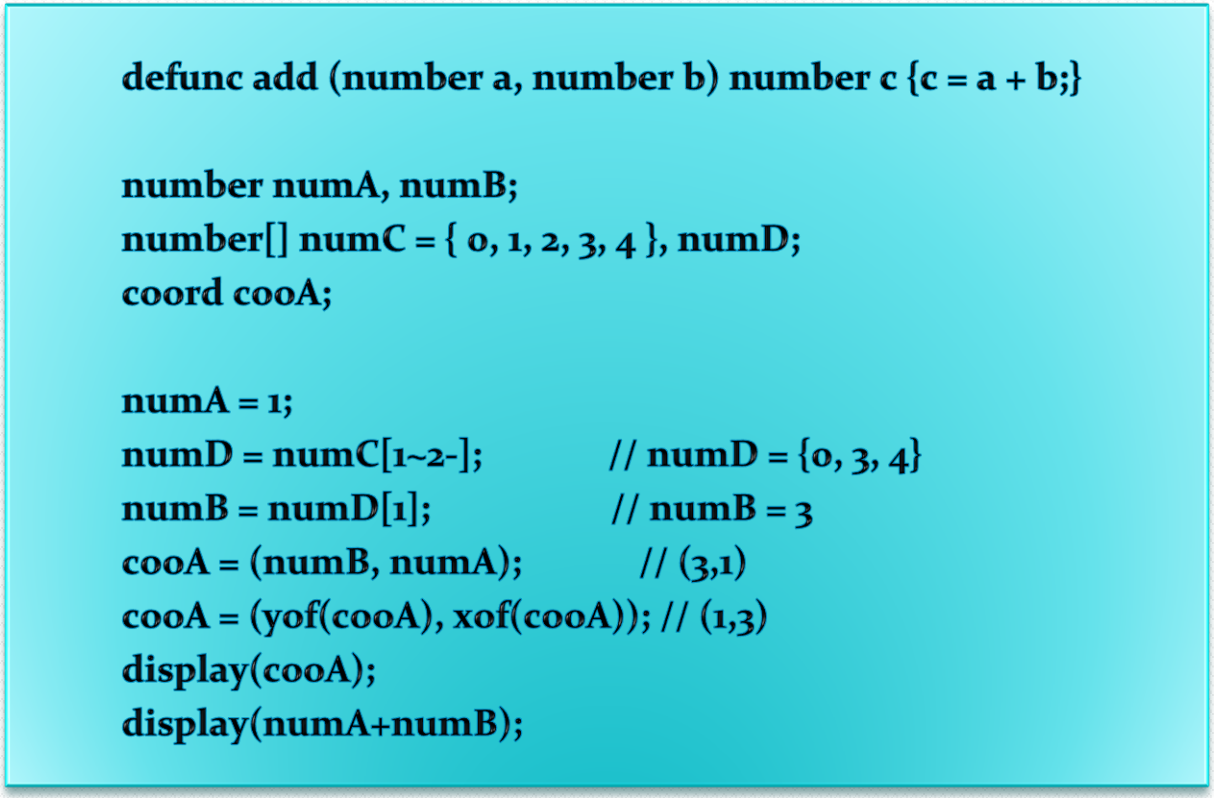
# Test and Debug Plan

- Test plan
  - Control statement
  - Function call
  - Static scope
  - Static image display
  - Image rotation
  - Image scale
  - Image rotate
  - Image set
  - Image alpha
  - Image animation
  - Image Array animation
  - Combined Image animation

- Debug Plan
  - Make debug flag and debug() for debugging
  - Using assert()
  - Using eclipse IDE
    - Good for debugging

# Examples (1/4)

- Basic Arithmetic, Coordination



```
defunc add (number a, number b) number c {c = a + b;}

number numA, numB;
number[] numC = { 0, 1, 2, 3, 4 }, numD;
coord cooA;

numA = 1;
numD = numC[1~2-];          // numD = {0, 3, 4}
numB = numD[1];             // numB = 3
cooA = (numB, numA);          // (3,1)
cooA = (yof(cooA), xof(cooA)); // (1,3)
display(cooA);
display(numA+numB);
```

Results:

(1,3)

4.0

# Examples (2/4)

- Static image

```
image imgA, imgB, imgC, imgD;

imgA = "sshield.jpg"` (100,100);
imgB = imgA` (800,100) @ 90;
imgC = imgA` (800,500) @ 180 :-100;
imgD = imgA` (100,500) @ 270 : -50;

display(imgA $ imgB $ imgC $ imgD);
```

IN LVMINE TVO
VIDEBIMVS LVMEN

sshield.jpg

# Examples (3/4)

- Animated image

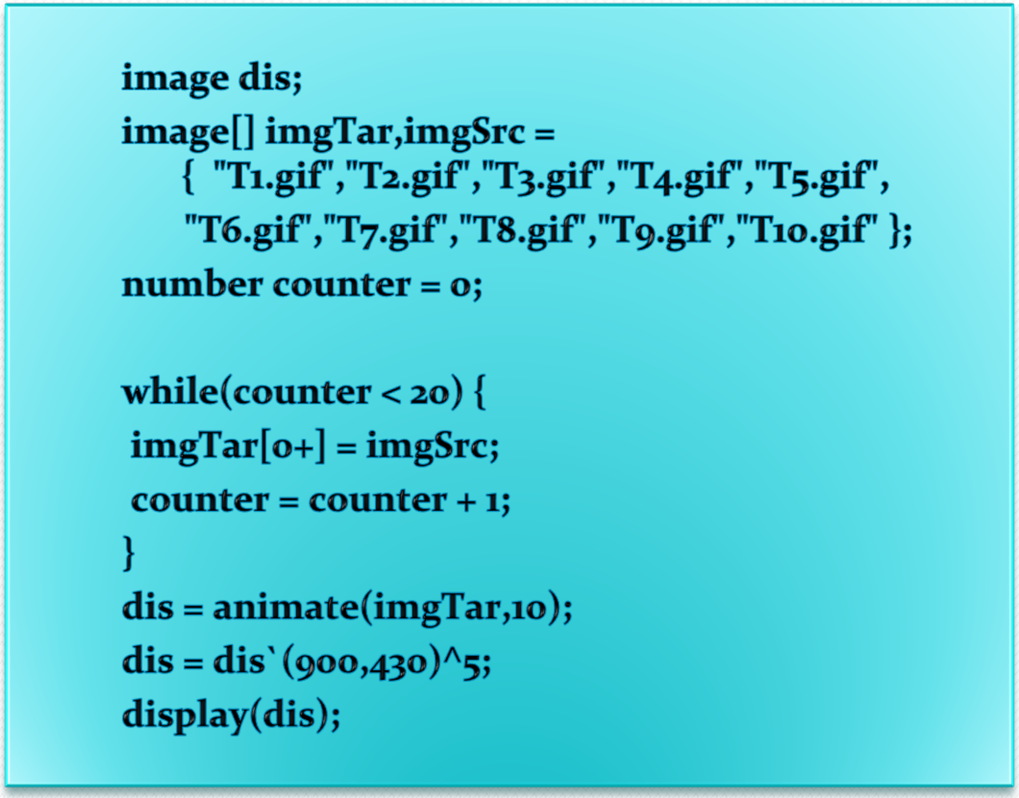
```
defunc rotate_animation(image src, number time, number
      rotate_amount) image target
{
 target = animate(src @ rotate_amount * time, time);
}
image imgA = "strawberry.jpg";
coord cooA;
number time = 8, rotate = 360;

cooA = (500,300);
imgA = imgA`cooA;
imgA = rotate_animation(imgA, time, rotate);
display(imgA);
```

strawberry.jpg

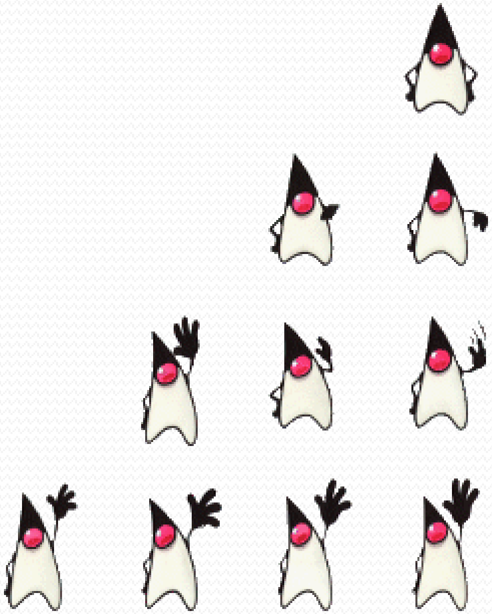
# Examples (4/4)

- GIF animation



```
image dis;
image[] imgTar,imgSrc =
      {  "T1.gif","T2.gif","T3.gif","T4.gif","T5.gif",
      "T6.gif","T7.gif","T8.gif","T9.gif","T10.gif" };
number counter = 0;

while(counter < 20) {
 imgTar[0+] = imgSrc;
 counter = counter + 1;
}
dis = animate(imgTar,10);
dis = dis`(900,430)^5;
display(dis);
```



# Lessons learned

- Things learned from Software Engineering actually works!
- Still, Team management.
  - Especially for Time management
  - Hard to find implement together!
- Need more fair distribution to learn
  - To learn something, everybody should do every procedure together that we have.
- Clarify how compiler works!

# Q & A?

Thank you for listening our presentation