LAVA

W4115 Group Project

Hongning Yuan hy2486

Jiacheng Liu jl4784

An Wang aw3001

Yimin Wei yw2907

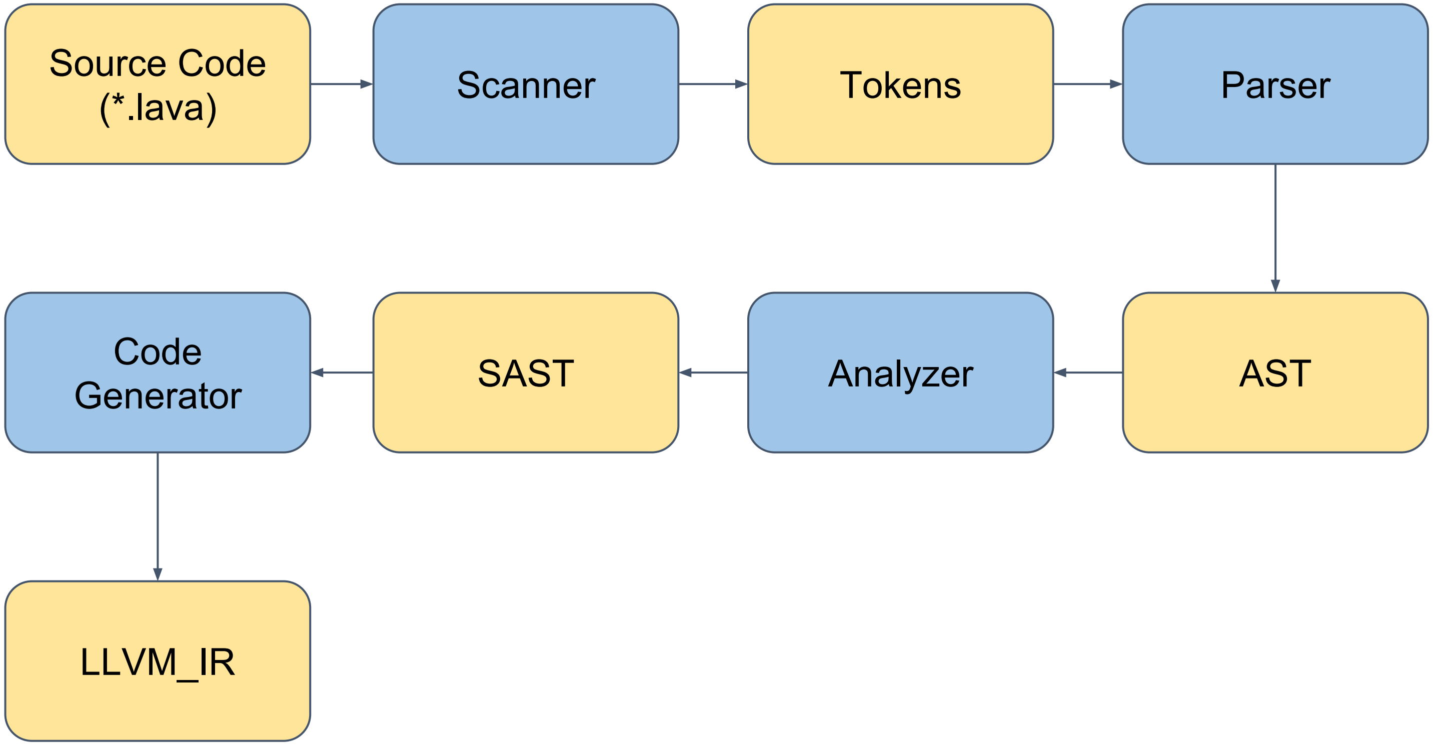
Introduction

- Lava - Volcano - The Java Island
- Originally known as a dialect of Java on JVM
- Turned out to be C++ on LLVm
- Object Oriented without Inheritance
- Casting between primitives

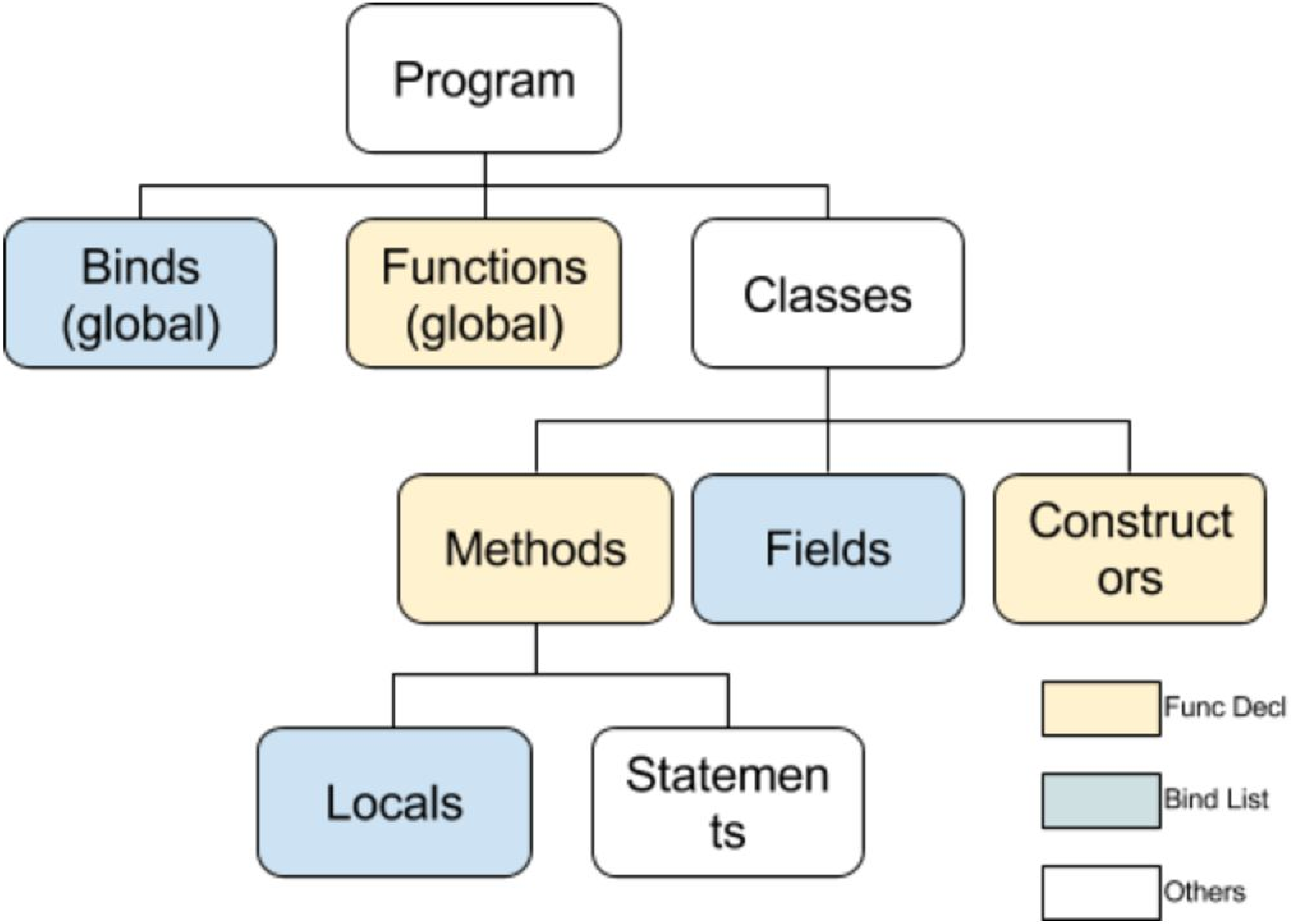
Compiler Overview

- Compiler files:
  - lava.ml: the entry pointer of the compiler
  - scanner.mll: scanning the input into tokens
  - ast.ml: the AST
  - sast.ml: the SAST
  - parser.mly: parsing tokens to a AST
  - analyzer.ml: semantic checking, generating the SAST
  - codegen.ml: code generation
- 2120 lines of code
- Automated testing in tests file

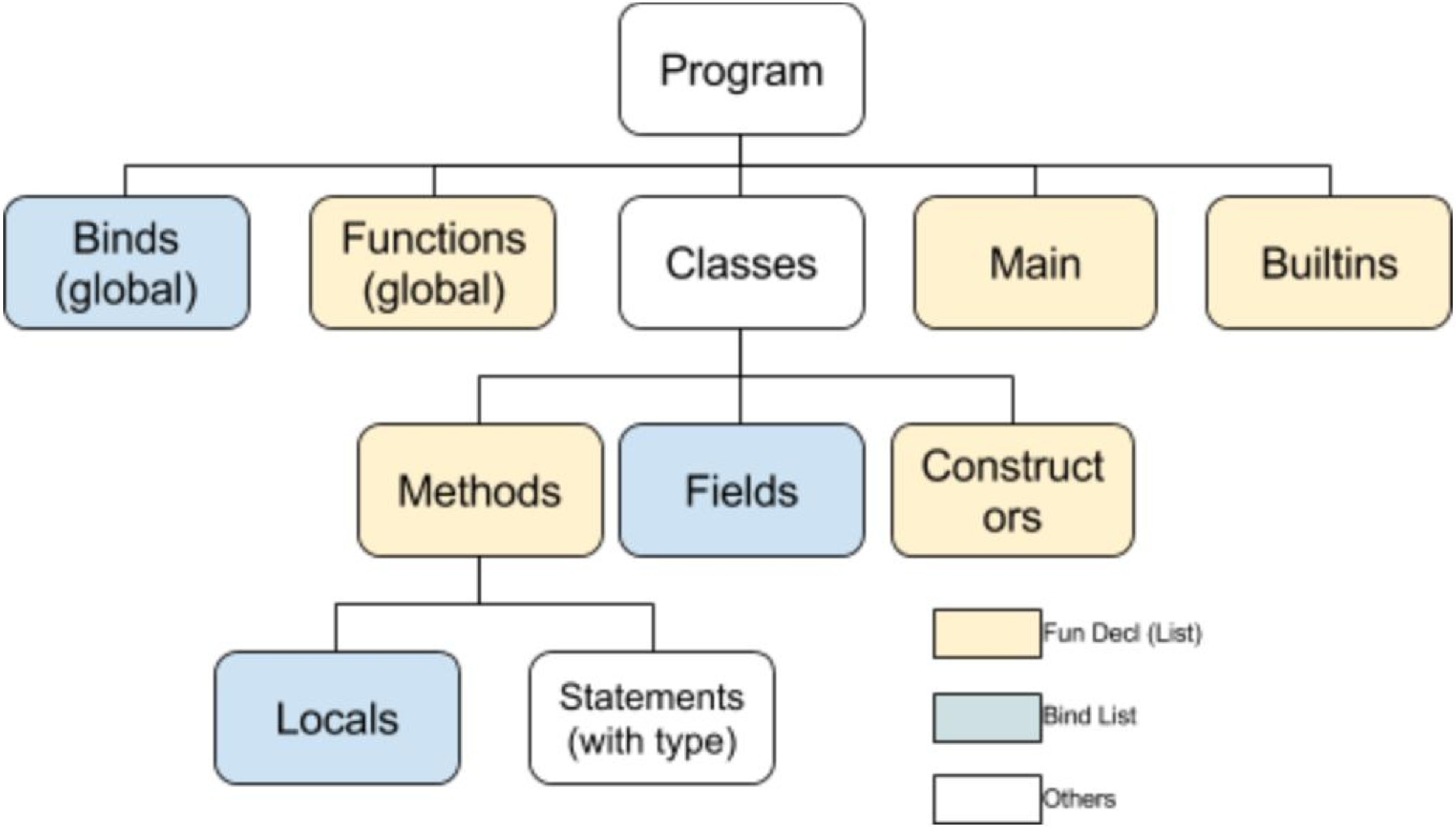
Architecture



Abstract Syntax Tree



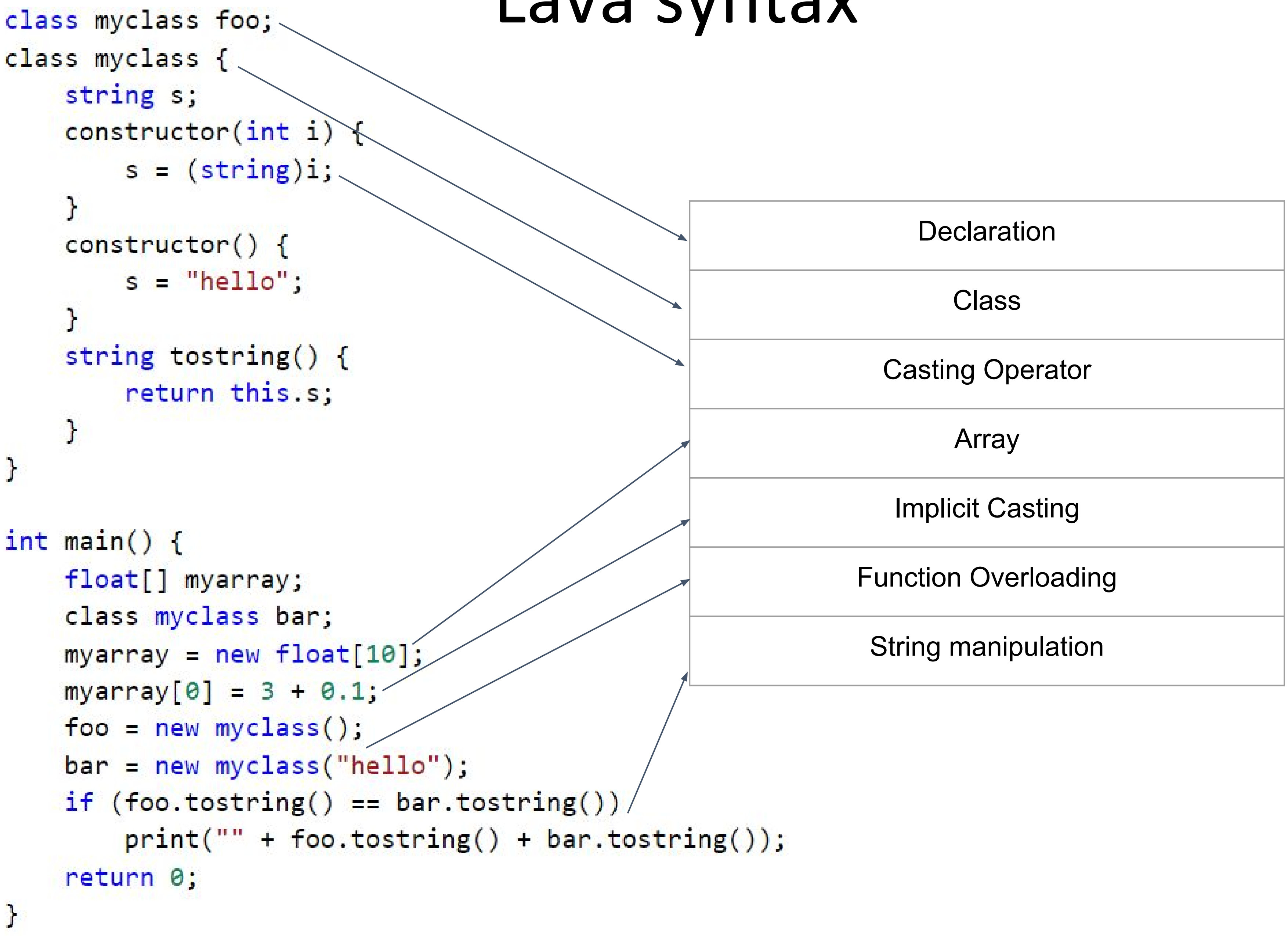
Abstract Syntax Tree (Semantically checked)



Features

- C++-like language (supposed to be a Java-like language)
- Everything implemented in microC
- Class
- Array
- String, Float, Casting operator
- String manipulation (concatenation, comparison, casting)
- Function overloading

Lava syntax



```
class myclass foo;
class myclass {
    string s;
    constructor(int i) {
        s = (string)i;
    }
    constructor() {
        s = "hello";
    }
    string tostring() {
        return this.s;
    }
}

int main() {
    float[] myarray;
    class myclass bar;
    myarray = new float[10];
    myarray[0] = 3 + 0.1;
    foo = new myclass();
    bar = new myclass("hello");
    if (foo.tostring() == bar.tostring())
        print("" + foo.tostring() + bar.tostring());
    return 0;
}
```

Declaration

Class

Casting Operator

Array

Implicit Casting

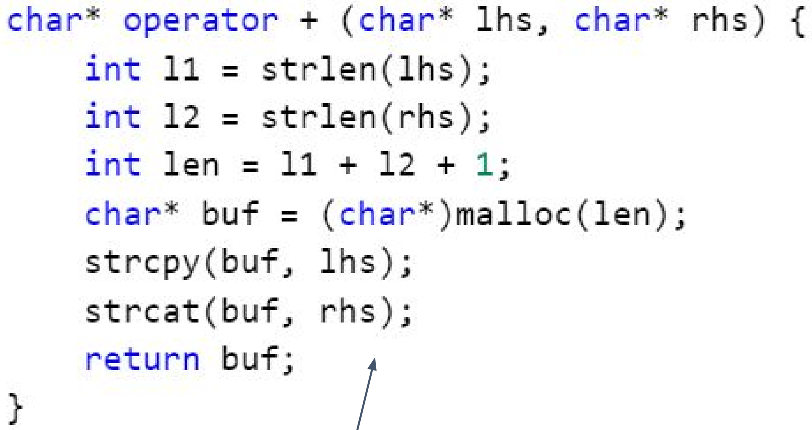
Function Overloading

String manipulation

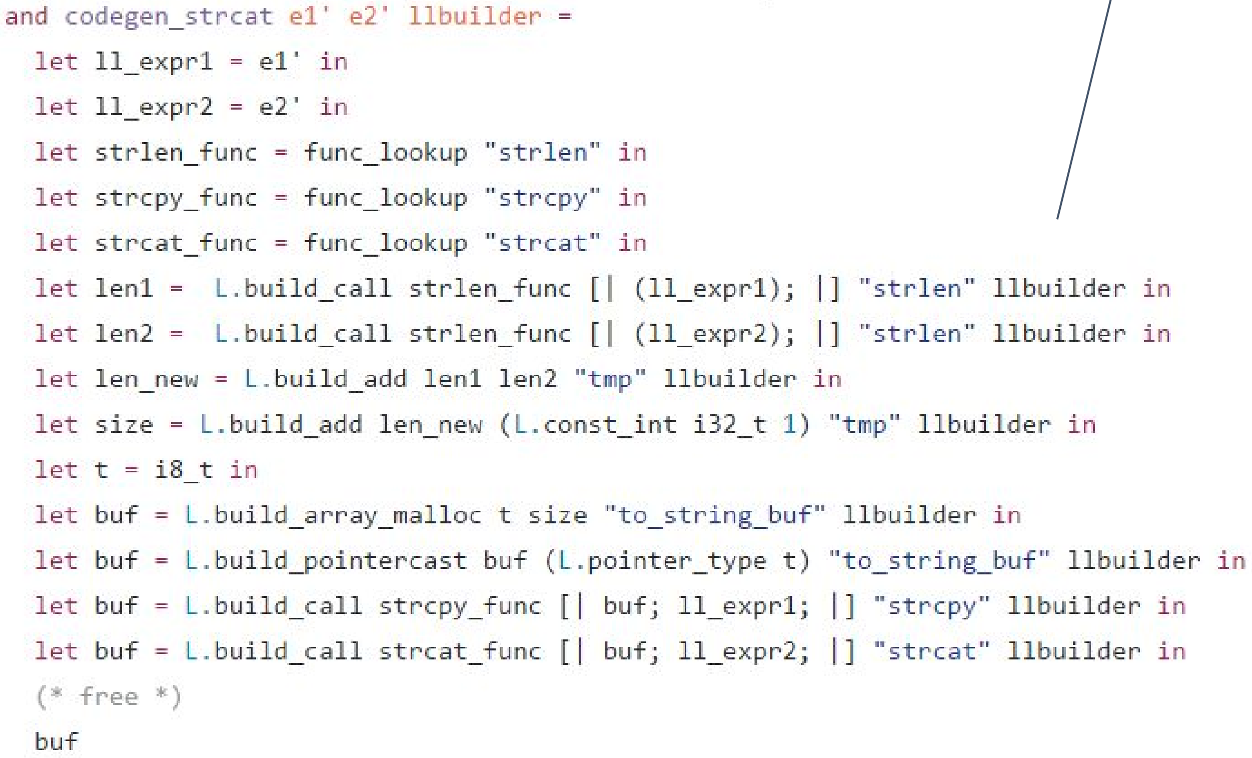
Features Highlight

- String manipulation
  - + operator (concatenation)
  - == operator (comparison)
  - casting operator

  (int to string, string to int)



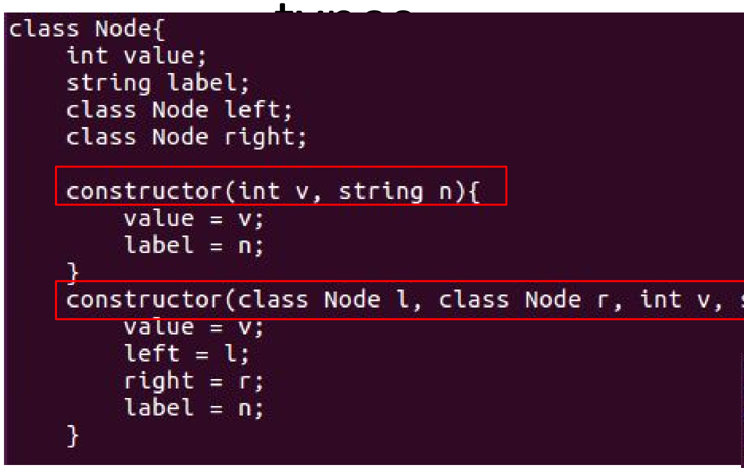
```c
char* operator + (char* lhs, char* rhs) {
    int l1 = strlen(lhs);
    int l2 = strlen(rhs);
    int len = l1 + l2 + 1;
    char* buf = (char*)malloc(len);
    strcpy(buf, lhs);
    strcat(buf, rhs);
    return buf;
}
```



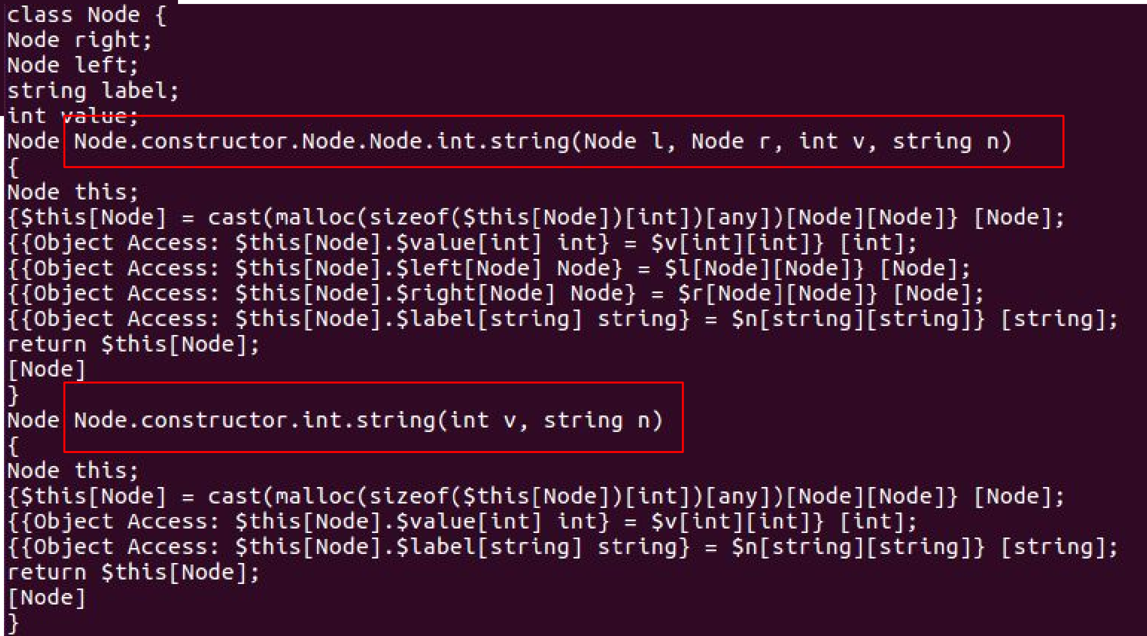
```ocaml
and codegen_strcat e1' e2' llbuilder =
  let ll_expr1 = e1' in
  let ll_expr2 = e2' in
  let strlen_func = func_lookup "strlen" in
  let strcpy_func = func_lookup "strcpy" in
  let strcat_func = func_lookup "strcat" in
  let len1 =  L.build_call strlen_func [| (ll_expr1); |] "strlen" llbuilder in
  let len2 =  L.build_call strlen_func [| (ll_expr2); |] "strlen" llbuilder in
  let len_new = L.build_add len1 len2 "tmp" llbuilder in
  let size = L.build_add len_new (L.const_int i32_t 1) "tmp" llbuilder in
  let t = i8_t in
  let buf = L.build_array_malloc t size "to_string_buf" llbuilder in
  let buf = L.build_pointercast buf (L.pointer_type t) "to_string_buf" llbuilder in
  let buf = L.build_call strcpy_func [| buf; ll_expr1; |] "strcpy" llbuilder in
  let buf = L.build_call strcat_func [| buf; ll_expr2; |] "strcat" llbuilder in
  (* free *)
  buf
```

Features Highlight

- Function Overloading
  - Same in name, different in params
  - In analyzer, rename the function to embed parameter types



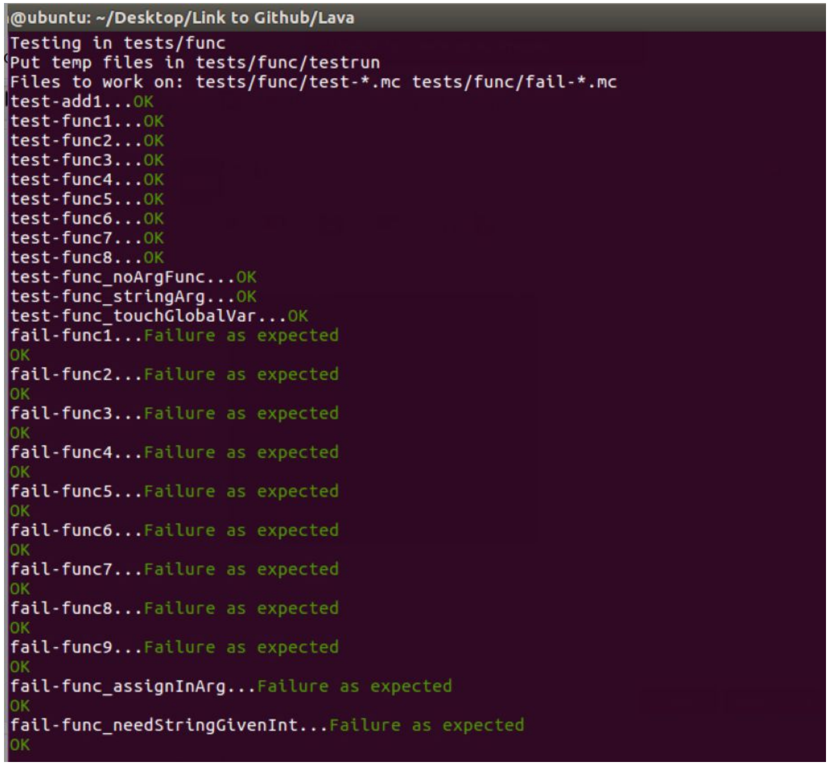
```
class Node{
    int value;
    string label;
    class Node left;
    class Node right;

    constructor(int v, string n){
        value = v;
        label = n;
    }
    constructor(class Node l, class Node r, int v, string n){
        value = v;
        left = l;
        right = r;
        label = n;
    }
```



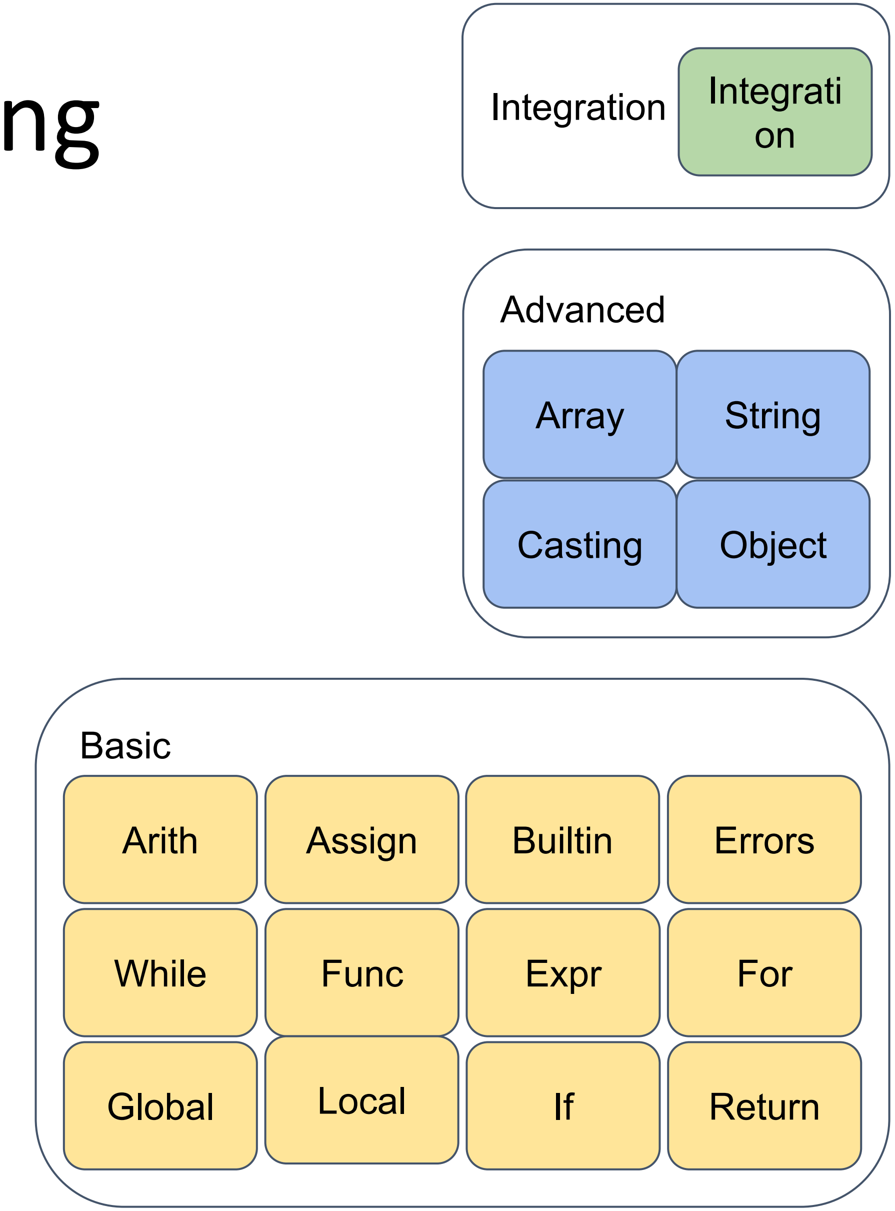
```
class Node {
Node right;
Node left;
string label;
int value;
Node Node.constructor.Node.Node.int.string(Node l, Node r, int v, string n)
{
Node this;
{$this[Node] = cast(malloc(sizeof($this[Node])[int])[any])[Node][Node]} [Node];
{{Object Access: $this[Node].$value[int] int} = $v[int][int]} [int];
{{Object Access: $this[Node].$left[Node] Node} = $l[Node][Node]} [Node];
{{Object Access: $this[Node].$right[Node] Node} = $r[Node][Node]} [Node];
{{Object Access: $this[Node].$label[string] string} = $n[string][string]} [string];
return $this[Node];
[Node]
}
Node Node.constructor.int.string(int v, string n)
{
Node this;
{$this[Node] = cast(malloc(sizeof($this[Node])[int])[any])[Node][Node]} [Node];
{{Object Access: $this[Node].$value[int] int} = $v[int][int]} [int];
{{Object Access: $this[Node].$label[string] string} = $n[string][string]} [string];
return $this[Node];
[Node]
}
```

Testing

- Automated testing
  - Test driven
  - Test to pass & test to fail
  - Regression testing
  - More than 150 test cases



```
@ubuntu: ~/Desktop/Link to Github/Lava
Testing in tests/func
Put temp files in tests/func/testrun
Files to work on: tests/func/test-*.mc tests/func/fail-*.mc
test-add1...OK
test-func1...OK
test-func2...OK
test-func3...OK
test-func4...OK
test-func5...OK
test-func6...OK
test-func7...OK
test-func8...OK
test-func_noArgFunc...OK
test-func_stringArg...OK
test-func_touchGlobalVar...OK
fail-func1...Failure as expected
OK
fail-func2...Failure as expected
OK
fail-func3...Failure as expected
OK
fail-func4...Failure as expected
OK
fail-func5...Failure as expected
OK
fail-func6...Failure as expected
OK
fail-func7...Failure as expected
OK
fail-func8...Failure as expected
OK
fail-func9...Failure as expected
OK
fail-func_assignInArg...Failure as expected
OK
fail-func_needStringGivenInt...Failure as expected
OK
```



Integration

Integrati on

Advanced

Array	String
Casting	Object

Basic

Arith	Assign	Builtin	Errors
While	Func	Expr	For
Global	Local	If	Return

Demonstration