**TEST PLAN**

All the test cases used to test Liva are put into a folder named tests. The test cases which consist of unit tests, integration tests are designed to test all the features of Liva, from both positive way and negative way. All these various testing methods are used to create a robust testing environment for Liva language.

Reason for chosing test cases

All the tests were added as new language features were added, therefore, the language features decided the test cases we picked out to a large extent, and most of these tests are aimed at testing every aspect of Liva.

Unit testing

Unit testing was used to check whether small pieces of our language could behave as defined. The tests can be divided into two types: tests are meant to pass, tests are meant to fail, thereby utilizing positive and negative testing. Negative testing ensures that invalid input is not accepted, i.e. Liva is able to properly reject invalid input, while positive testing on the other hand allows us to assess whether Liva is able to behave and work out the result as defined.

Integration testing

Once smaller tests were verified to pass, they would be integrated into larger programs, so as to ensure whether Liva can manage to behave properly in more complex program. These integrated tests are the basis of our final interesting program.

Automation

Automation of testing becomes more and more necessary as the project moves forward, and it is an extremely useful tool for developing our language. Automation allows us to make sure that new added language features would not obstruct other features which have passed the tests. We created an automated regression test suite largely borrowed from the MicroC Compiler. The test cases meant to pass are written out using the notation ‘test-.liva’ and the corresponding output as ‘test-.out’, while the corresponding output of test cases meat to fail as ‘test-.err’. The test script is executed with the ./testall.sh command which will then display a list of tests that pass, fail, or produce a printed output that differs from the desired printed output.

Representative source language programs

1.

test-inheritance2.liva

class calculator {

int add(int x, int y){

int z = x + y;

return(z);

}

}

class my\_calculator extends calculator{

}

class test {

void main(){

int x;

int y;

int z;

x = 66;

y = 98;

class my\_calculator obj = new my\_calculator();

z = obj.add(x,y);

print ("z=",z);

}

}

test-inheritance2.ll

; ModuleID = 'Liva'

%test = type <{ i32 }>

%my\_calculator = type <{ i32 }>

%calculator = type <{ i32 }>

@tmp = private unnamed\_addr constant [3 x i8] c"z=\00"

@tmp.1 = private unnamed\_addr constant [5 x i8] c"%s%d\00"

declare i32 @printf(i8\*, ...)

declare i8\* @malloc(i32)

define i64\* @lookup(i32 %c\_index, i32 %f\_index) {

entry:

%tmp = alloca i64\*\*, i32 3

%tmp1 = alloca i64\*, i32 0

%tmp2 = getelementptr i64\*\*, i64\*\*\* %tmp, i32 2

store i64\*\* %tmp1, i64\*\*\* %tmp2

%tmp3 = alloca i64\*

%tmp4 = getelementptr i64\*, i64\*\* %tmp3, i32 0

store i64\* bitcast (i32 (%my\_calculator\*, i32, i32)\* @my\_calculator.add to i64\*), i64\*\* %tmp4

%tmp5 = getelementptr i64\*\*, i64\*\*\* %tmp, i32 1

store i64\*\* %tmp3, i64\*\*\* %tmp5

%tmp6 = alloca i64\*

%tmp7 = getelementptr i64\*, i64\*\* %tmp6, i32 0

store i64\* bitcast (i32 (%calculator\*, i32, i32)\* @calculator.add to i64\*), i64\*\* %tmp7

%tmp8 = getelementptr i64\*\*, i64\*\*\* %tmp, i32 0

store i64\*\* %tmp6, i64\*\*\* %tmp8

%tmp9 = getelementptr i64\*\*, i64\*\*\* %tmp, i32 %c\_index

%tmp10 = load i64\*\*, i64\*\*\* %tmp9

%tmp11 = getelementptr i64\*, i64\*\* %tmp10, i32 %f\_index

%tmp12 = load i64\*, i64\*\* %tmp11

ret i64\* %tmp12

}

define %test\* @test.constructor() {

entry:

%this = alloca %test

%tmp = call i8\* @malloc(i32 ptrtoint (i1\*\* getelementptr (i1\*, i1\*\* null, i32 1) to i32))

%tmp1 = bitcast i8\* %tmp to %test\*

%tmp2 = load %test, %test\* %tmp1

store %test %tmp2, %test\* %this

%.key = getelementptr inbounds %test, %test\* %this, i32 0, i32 0

store i32 2, i32\* %.key

ret %test\* %this

}

define i32 @my\_calculator.add(%my\_calculator\* %this, i32 %x, i32 %y) {

entry:

%z = alloca i32

%addtmp = add i32 %x, %y

store i32 %addtmp, i32\* %z

%z1 = load i32, i32\* %z

ret i32 %z1

}

define %my\_calculator\* @my\_calculator.constructor() {

entry:

%this = alloca %my\_calculator

%tmp = call i8\* @malloc(i32 ptrtoint (i1\*\* getelementptr (i1\*, i1\*\* null, i32 1) to i32))

%tmp1 = bitcast i8\* %tmp to %my\_calculator\*

%tmp2 = load %my\_calculator, %my\_calculator\* %tmp1

store %my\_calculator %tmp2, %my\_calculator\* %this

%.key = getelementptr inbounds %my\_calculator, %my\_calculator\* %this, i32 0, i32 0

store i32 1, i32\* %.key

ret %my\_calculator\* %this

}

define i32 @calculator.add(%calculator\* %this, i32 %x, i32 %y) {

entry:

%z = alloca i32

%addtmp = add i32 %x, %y

store i32 %addtmp, i32\* %z

%z1 = load i32, i32\* %z

ret i32 %z1

}

define %calculator\* @calculator.constructor() {

entry:

%this = alloca %calculator

%tmp = call i8\* @malloc(i32 ptrtoint (i1\*\* getelementptr (i1\*, i1\*\* null, i32 1) to i32))

%tmp1 = bitcast i8\* %tmp to %calculator\*

%tmp2 = load %calculator, %calculator\* %tmp1

store %calculator %tmp2, %calculator\* %this

%.key = getelementptr inbounds %calculator, %calculator\* %this, i32 0, i32 0

store i32 0, i32\* %.key

ret %calculator\* %this

}

define i32 @main() {

entry:

%this = alloca %test

%tmp = call i8\* @malloc(i32 ptrtoint (i1\*\* getelementptr (i1\*, i1\*\* null, i32 1) to i32))

%tmp1 = bitcast i8\* %tmp to %test\*

%tmp2 = load %test, %test\* %tmp1

store %test %tmp2, %test\* %this

%.key = getelementptr inbounds %test, %test\* %this, i32 0, i32 0

store i32 2, i32\* %.key

%x = alloca i32

%y = alloca i32

%z = alloca i32

store i32 66, i32\* %x

store i32 98, i32\* %y

%obj = alloca %my\_calculator

%tmp3 = call %my\_calculator\* @my\_calculator.constructor()

%tmp4 = load %my\_calculator, %my\_calculator\* %tmp3

store %my\_calculator %tmp4, %my\_calculator\* %obj

%cindex = getelementptr inbounds %my\_calculator, %my\_calculator\* %obj, i32 0, i32 0

%cindex5 = load i32, i32\* %cindex

%fptr = call i64\* @lookup(i32 %cindex5, i32 0)

%my\_calculator.add = bitcast i64\* %fptr to i32 (%my\_calculator\*, i32, i32)\*

%x6 = load i32, i32\* %x

%y7 = load i32, i32\* %y

%tmp8 = call i32 %my\_calculator.add(%my\_calculator\* %obj, i32 %x6, i32 %y7)

store i32 %tmp8, i32\* %z

%z9 = load i32, i32\* %z

%tmp10 = call i32 (i8\*, ...) @printf(i8\* getelementptr inbounds ([5 x i8], [5 x i8]\* @tmp.1, i32 0, i32 0), i8\* getelementptr inbounds ([3 x i8], [3 x i8]\* @tmp, i32 0, i32 0), i32 %z9)

ret i32 0

}

2.

test-while\_for\_nest.liva

class test {

void main(){

int i = 1;

int j;

while(i < 10)

{

j = 11;

for(j = 11; j < 13; j = j + 1)

print("i = ", i, " ", "j = ", j, "\n");

i = i + 1;

}

print("\n\n");

for(i = 1; i < 10; i = i + 1)

{

j = 11;

while(j < 13)

{

print("i = ", i, " ", "j = ", j, "\n");

j = j + 1;

}

}

}

}

test-while\_for\_nest.ll

; ModuleID = 'Liva'

%test = type <{ i32 }>

@tmp = private unnamed\_addr constant [5 x i8] c"i = \00"

@tmp.1 = private unnamed\_addr constant [2 x i8] c" \00"

@tmp.2 = private unnamed\_addr constant [5 x i8] c"j = \00"

@tmp.3 = private unnamed\_addr constant [2 x i8] c"\0A\00"

@tmp.4 = private unnamed\_addr constant [13 x i8] c"%s%d%s%s%d%s\00"

@tmp.5 = private unnamed\_addr constant [3 x i8] c"\0A\0A\00"

@tmp.6 = private unnamed\_addr constant [3 x i8] c"%s\00"

@tmp.7 = private unnamed\_addr constant [5 x i8] c"i = \00"

@tmp.8 = private unnamed\_addr constant [2 x i8] c" \00"

@tmp.9 = private unnamed\_addr constant [5 x i8] c"j = \00"

@tmp.10 = private unnamed\_addr constant [2 x i8] c"\0A\00"

@tmp.11 = private unnamed\_addr constant [13 x i8] c"%s%d%s%s%d%s\00"

declare i32 @printf(i8\*, ...)

declare i8\* @malloc(i32)

define i64\* @lookup(i32 %c\_index, i32 %f\_index) {

entry:

%tmp = alloca i64\*\*

%tmp1 = alloca i64\*, i32 0

%tmp2 = getelementptr i64\*\*, i64\*\*\* %tmp, i32 0

store i64\*\* %tmp1, i64\*\*\* %tmp2

ret i64\* null

}

define %test\* @test.constructor() {

entry:

%this = alloca %test

%tmp = call i8\* @malloc(i32 ptrtoint (i1\*\* getelementptr (i1\*, i1\*\* null, i32 1) to i32))

%tmp1 = bitcast i8\* %tmp to %test\*

%tmp2 = load %test, %test\* %tmp1

store %test %tmp2, %test\* %this

%.key = getelementptr inbounds %test, %test\* %this, i32 0, i32 0

store i32 0, i32\* %.key

ret %test\* %this

}

define i32 @main() {

entry:

%this = alloca %test

%tmp = call i8\* @malloc(i32 ptrtoint (i1\*\* getelementptr (i1\*, i1\*\* null, i32 1) to i32))

%tmp1 = bitcast i8\* %tmp to %test\*

%tmp2 = load %test, %test\* %tmp1

store %test %tmp2, %test\* %this

%.key = getelementptr inbounds %test, %test\* %this, i32 0, i32 0

store i32 0, i32\* %.key

%i = alloca i32

store i32 1, i32\* %i

%j = alloca i32

br label %cond

loop: ; preds = %cond

store i32 11, i32\* %j

store i32 11, i32\* %j

br label %cond5

loop3: ; preds = %cond5

%i7 = load i32, i32\* %i

%j8 = load i32, i32\* %j

%tmp9 = call i32 (i8\*, ...) @printf(i8\* getelementptr inbounds ([13 x i8], [13 x i8]\* @tmp.4, i32 0, i32 0), i8\* getelementptr inbounds ([5 x i8], [5 x i8]\* @tmp, i32 0, i32 0), i32 %i7, i8\* getelementptr inbounds ([2 x i8], [2 x i8]\* @tmp.1, i32 0, i32 0), i8\* getelementptr inbounds ([5 x i8], [5 x i8]\* @tmp.2, i32 0, i32 0), i32 %j8, i8\* getelementptr inbounds ([2 x i8], [2 x i8]\* @tmp.3, i32 0, i32 0))

br label %step4

step4: ; preds = %loop3

%j10 = load i32, i32\* %j

%addtmp = add i32 %j10, 1

store i32 %addtmp, i32\* %j

br label %cond5

cond5: ; preds = %step4, %loop

%j11 = load i32, i32\* %j

%lesstmp = icmp slt i32 %j11, 13

br i1 %lesstmp, label %loop3, label %afterloop6

afterloop6: ; preds = %cond5

%i12 = load i32, i32\* %i

%addtmp13 = add i32 %i12, 1

store i32 %addtmp13, i32\* %i

br label %step

step: ; preds = %afterloop6

br label %cond

cond: ; preds = %step, %entry

%i14 = load i32, i32\* %i

%lesstmp15 = icmp slt i32 %i14, 10

br i1 %lesstmp15, label %loop, label %afterloop

afterloop: ; preds = %cond

%tmp16 = call i32 (i8\*, ...) @printf(i8\* getelementptr inbounds ([3 x i8], [3 x i8]\* @tmp.6, i32 0, i32 0), i8\* getelementptr inbounds ([3 x i8], [3 x i8]\* @tmp.5, i32 0, i32 0))

store i32 1, i32\* %i

br label %cond19

loop17: ; preds = %cond19

store i32 11, i32\* %j

br label %cond23

loop21: ; preds = %cond23

%i25 = load i32, i32\* %i

%j26 = load i32, i32\* %j

%tmp27 = call i32 (i8\*, ...) @printf(i8\* getelementptr inbounds ([13 x i8], [13 x i8]\* @tmp.11, i32 0, i32 0), i8\* getelementptr inbounds ([5 x i8], [5 x i8]\* @tmp.7, i32 0, i32 0), i32 %i25, i8\* getelementptr inbounds ([2 x i8], [2 x i8]\* @tmp.8, i32 0, i32 0), i8\* getelementptr inbounds ([5 x i8], [5 x i8]\* @tmp.9, i32 0, i32 0), i32 %j26, i8\* getelementptr inbounds ([2 x i8], [2 x i8]\* @tmp.10, i32 0, i32 0))

%j28 = load i32, i32\* %j

%addtmp29 = add i32 %j28, 1

store i32 %addtmp29, i32\* %j

br label %step22

step22: ; preds = %loop21

br label %cond23

cond23: ; preds = %step22, %loop17

%j30 = load i32, i32\* %j

%lesstmp31 = icmp slt i32 %j30, 13

br i1 %lesstmp31, label %loop21, label %afterloop24

afterloop24: ; preds = %cond23

br label %step18

step18: ; preds = %afterloop24

%i32 = load i32, i32\* %i

%addtmp33 = add i32 %i32, 1

store i32 %addtmp33, i32\* %i

br label %cond19

cond19: ; preds = %step18, %afterloop

%i34 = load i32, i32\* %i

%lesstmp35 = icmp slt i32 %i34, 10

br i1 %lesstmp35, label %loop17, label %afterloop20

afterloop20: ; preds = %cond19

ret i32 0

}

Test suites

fail-add.liva:

class arith {

void main()

{

int i;

i = "1" + 1;

print(i);

}

}

------------------------------------------------------------------------------------------------------

fail-array\_access.liva:

class test {

void main() {

char b = 'a';

float[] a = new float[10];

print(a[b]);

}

}

------------------------------------------------------------------------------------------------------

fail-array\_access2.liva:

class test {

void main() {

float[] a = new float[10];

print(a[1][1]);

}

}

------------------------------------------------------------------------------------------------------

fail-array\_init.liva:

class test {

void main() {

float[] a = new float[10.0];

}

}

------------------------------------------------------------------------------------------------------

fail-diff.liva:

class arith {

void main()

{

int i;

i = "1" - 1;

print(i);

}

}

------------------------------------------------------------------------------------------------------

fail-div.liva:

class arith {

void main()

{

int i;

i = "4" / 2;

print(i);

}

}

------------------------------------------------------------------------------------------------------

fail-equal1.liva:

class test {

void main(){

float i = 1.0;

if (i == 1.0) print(42);

else print(8);

}

}

------------------------------------------------------------------------------------------------------

fail-equal2.liva:

class test {

void main(){

float i = "123";

if (i == true) print(42);

else print(8);

}

}

------------------------------------------------------------------------------------------------------

fail-for1.liva:

class test {

void main(){

int i;

for (i = 0 ; i = 10 ; i = i + 1) {

print(i);

}

}

}

------------------------------------------------------------------------------------------------------

fail-function.liva:

class myclass{

int calc (int x, int y){

int z;

z = x + y;

return (z);

}

}

class test {

void main(){

int x = 9;

int y = 6;

int z;

class myclass obj = new myclass();

z = obj.ca\_lc(x, y);

print ("z=",z);

}

}

------------------------------------------------------------------------------------------------------

fail-function2.liva:

class myclass{

int calc (int x, int y){

int z;

z = x + y;

return (z);

}

}

class test {

void main(){

int x = 9;

int y = 6;

int z;

class myclass obj = new myclass();

z = obj.calc(x, x, y);

print ("z=",z);

}

}

------------------------------------------------------------------------------------------------------

fail-function3.liva:

class myclass{

int calc (int x, int y){

int z;

z = x + y;

return (z);

}

}

class test {

void main(){

int x = 9;

float y = 6.0;

float z;

class myclass obj = new myclass();

z = obj.calc(x, y);

print ("z=",z);

}

}

------------------------------------------------------------------------------------------------------

fail-hello.liva:

class test {

print ("Hello World!");

void main(){

}

}

------------------------------------------------------------------------------------------------------

fail-hello2.liva:

class test {

void main(){

int a ;

int b ;

a=1.1;

b=3;

print ("multiple ", "params!", "\n", a, "\n" ,b, "\n");

}

}

------------------------------------------------------------------------------------------------------

fail-if1.liva:

class test {

void main(){

if ("123") print(42);

}

}

------------------------------------------------------------------------------------------------------

fail-mod.liva:

class arith {

void main()

{

int i;

i = "4" % 3;

print(i);

}

}

------------------------------------------------------------------------------------------------------

fail-mul.liva:

class arith {

void main()

{

int i;

i = "15" \* 5;

print(i);

}

}

------------------------------------------------------------------------------------------------------

fail-not.liva:

class test {

void main(){

int i = 1;

boolean j;

j = !(i + 1);

}

}

------------------------------------------------------------------------------------------------------

fail-obj\_access.liva:

class test {

void main() {

int a;

a.amethod;

}

}

------------------------------------------------------------------------------------------------------

fail-obj\_access2.liva:

class test {

void main() {

int a;

(1+1).amethod;

}

}

------------------------------------------------------------------------------------------------------

fail-obj\_access3.liva:

class myclass{

int a;

constructor(int x){

this.a = x;

}

}

class test {

void main(){

class myclass obj = new myclass(10);

print ("b=",obj.b);

}

}

------------------------------------------------------------------------------------------------------

fail-sub.liva:

class test {

void main(){

int j;

j = -(true);

print(j);

}

}

------------------------------------------------------------------------------------------------------

fail-while1.liva:

class test {

void main() {

int i;

i = 5;

while (i = 1) {

print(i);

i = i - 1;

}

print(42);

}

}

------------------------------------------------------------------------------------------------------

test-add.liva:

class arith {

void main()

{

int i;

i = 1 + 1;

print(i);

}

}

------------------------------------------------------------------------------------------------------

test-and.liva:

class test {

void main(){

int i = 1;

int j = 3;

if (i == 1 & j == 3)

{

print(i, " ", j, "\n");

}

}

}

------------------------------------------------------------------------------------------------------

test-arith.liva:

class arith {

void main()

{

int i;

i = 1 + 3 \* 4 % 7 - 4 / 2;

print(i);

}

}

------------------------------------------------------------------------------------------------------

test-array.liva:

class test {

void main() {

float[] a = new float[10];

int[] b = new int[10];

int i;

a[0] = 1.0;

b[0] = 1;

for(i = 1; i < 10; i = i + 1)

{

a[i] = a[i - 1] + 1.0;

b[i] = b[i - 1] + 1;

}

for(i = 0; i < 10; i = i + 1)

print("a[",i,"]"," = ",a[i]," , ", "b[",i,"]"," = ",b[i], "\n");

}

}

------------------------------------------------------------------------------------------------------

test-array\_object.liva:

class calculator{

int g;

int addition(int x, int y){

this.g =9;

int z;

z = x + y;

return(z);

}

}

class test {

void main() {

class calculator c = new calculator();

class calculator[] a = new class calculator[10];

a[0] = c;

print(a[0].addition(1,1));

}

}

------------------------------------------------------------------------------------------------------

test-comments.liva:

class test {

void main(){

float i = 1.111;

/\*HAHAHAHA

/\* print(i);\*/

BOOOO!%$#$^%^&^%g)\_\_\*%^#@...

\*/

print(i);

}

}

------------------------------------------------------------------------------------------------------

test-constructor.liva:

class myclass{

int a;

constructor(int x){

this.a = x;

}

}

class test {

void main(){

class myclass obj = new myclass(10);

print ("a=",obj.a);

}

}

------------------------------------------------------------------------------------------------------

test-diff.liva:

class arith {

void main()

{

float i;

i = 1.3 - 1.0;

print(i);

}

}

------------------------------------------------------------------------------------------------------

test-div.liva:

class arith {

void main()

{

int i;

i = 4 / 2;

print(i);

}

}

------------------------------------------------------------------------------------------------------

test-equal.liva:

class test {

void main(){

int i = 1;

if (i == 1) print(42);

else print(8);

}

}

------------------------------------------------------------------------------------------------------

test-fib.liva:

class test {

int fib (int x){

int z;

if (x <2) z=1;

else z= this.fib(x-1) + this.fib(x-2);

return (z);

}

void main(){

int x;

int y;

int z;

int m;

x = 5;

y = 6;

z = this.fib (x);

print (z);

}

}

------------------------------------------------------------------------------------------------------

test-for1.liva:

class test {

void main(){

int i;

for (i = 0 ; i < 10 ; i = i + 1) {

print(i);

}

}

}

------------------------------------------------------------------------------------------------------

test-for\_nest.liva:

class test {

void main(){

int i;

int j;

for(i = 0; i < 10; i = i + 1)

for(j = 11; j < 13; j = j + 1)

print("i = ", i, " ", "j = ", j, "\n");

}

}

------------------------------------------------------------------------------------------------------

test-function.liva:

class myclass{

int calc (int x, int y){

int z;

z = x + y;

return (z);

}

}

class test {

void main(){

int x;

int y;

int z;

x = 9;

y = 6;

class myclass obj = new myclass();

z = obj.calc(x, y);

print ("z=",z);

}

}

------------------------------------------------------------------------------------------------------

test-gcd.liva:

class gcd {

void main(){

int x;

int y;

int z;

x = 66;

y = 98;

while(x != y){

if(x > y){

x = x - y;

}

else{

y = y - x;

}

}

print ("gcd=",x);

}

}

------------------------------------------------------------------------------------------------------

test-geq.liva:

class test {

void main(){

int i = 1;

int j = 1;

if (i >= j) print("yes");

else print("no");

}

}

------------------------------------------------------------------------------------------------------

test-gt.liva:

class test {

void main(){

int i = 4;

int j = 1;

if (i > j) print(4);

else print(8);

}

}

------------------------------------------------------------------------------------------------------

test-hello.liva:

class test {

void main(){

print ("Hello World!");

}

}

------------------------------------------------------------------------------------------------------

test-hello2.liva:

class test {

void main(){

int a ;

int b ;

a=1;

b=3;

print ("multiple ", "params!", "\n", a, "\n" ,b, "\n");

}

}

------------------------------------------------------------------------------------------------------

test-if1.liva:

class test {

void main(){

print (100);

if (true) print(42);

else print(8);

print(17);

}

}

------------------------------------------------------------------------------------------------------

test-if\_nest.liva:

class test {

void main(){

int i = 1;

int j = 3;

if (true)

{

if(i == 1)

{

if(i < j)

{

print(j);

}

}

}

}

}

------------------------------------------------------------------------------------------------------

test-inheritance.liva:

class myclass{

int a;

constructor(int x){

this.a = x;

}

}

class subclass extends myclass{

constructor(int x){

this.a = x;

}

}

class test {

void main(){

class subclass obj = new subclass(10);

print ("a=",obj.a);

}

}

------------------------------------------------------------------------------------------------------

test-inheritance2.liva:

class calculator {

int add(int x, int y){

int z = x + y;

return(z);

}

}

class my\_calculator extends calculator{

}

class test {

void main(){

int x;

int y;

int z;

x = 66;

y = 98;

class my\_calculator obj = new my\_calculator();

z = obj.add(x,y);

print ("z=",z);

}

}

------------------------------------------------------------------------------------------------------

test-leq.liva:

class test {

void main(){

int i = 1;

int j = 1;

if (i <= j) print(4);

else print(8);

}

}

------------------------------------------------------------------------------------------------------

test-lt.liva:

class test {

void main(){

int i = 3;

int j = 1;

if (i < j) print(4);

else print(8);

}

}

------------------------------------------------------------------------------------------------------

test-mod.liva:

class arith {

void main()

{

int i;

i = 4 % 3;

print(i);

}

}

------------------------------------------------------------------------------------------------------

test-mul.liva:

class arith {

void main()

{

int i;

i = 15 \* 5;

print(i);

}

}

------------------------------------------------------------------------------------------------------

test-nequal.liva:

class test {

void main(){

int i = 3;

if (i != 1) print(4);

else print(8);

}

}

------------------------------------------------------------------------------------------------------

test-not.liva:

class test {

void main(){

int i = 1;

int j = 3;

if (!(i != 1))

{

print("BOOO!");

}

}

}

------------------------------------------------------------------------------------------------------

test-obj.liva:

class myclass{

}

class test {

void main(){

class myclass obj = new myclass();

print ("obj\n");

}

}

------------------------------------------------------------------------------------------------------

test-or.liva:

class test {

void main(){

int i = 1;

int j = 3;

if (i != 1 | j == 3)

{

print(i, " ", j, "\n");

}

}

}

------------------------------------------------------------------------------------------------------

test-override.liva:

class myclass{

int a;

int calc (int x, int y){

int z;

z = x + y;

return (z);

}

}

class subclass extends myclass{

int calc (int x, int y){

int z;

z = x - y;

return (z);

}

}

class test {

void main(){

int x;

int y;

int z;

x = 9;

y = 6;

class subclass obj = new subclass();

z = obj.calc(x, y);

print ("z=",z);

}

}

------------------------------------------------------------------------------------------------------

test-sub.liva:

class test {

void main(){

int i = 3;

int j;

j = -(-(i) + i \* i);

print(j);

}

}

------------------------------------------------------------------------------------------------------

test-while1.liva:

class test {

void main() {

int i;

i = 5;

while (i > 0) {

print(i);

i = i - 1;

}

print(42);

}

}

------------------------------------------------------------------------------------------------------

test-while\_for\_nest.liva:

class test {

void main(){

int i = 1;

int j;

while(i < 10)

{

j = 11;

for(j = 11; j < 13; j = j + 1)

print("i = ", i, " ", "j = ", j, "\n");

i = i + 1;

}

print("\n\n");

for(i = 1; i < 10; i = i + 1)

{

j = 11;

while(j < 13)

{

print("i = ", i, " ", "j = ", j, "\n");

j = j + 1;

}

}

}

}