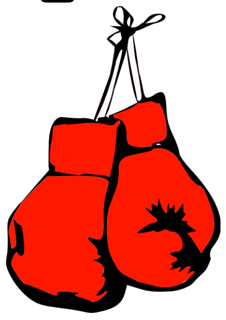
rocky2



"I think a something-to-Pd compiler might be really interesting and useful - I've never seen anyone try that."

- Miller Puckette, 2016

Gazo: How's about investing in condominiums? It's safe.

Rocky Balboa: Condominiums?

Gazo: Yeah, condominiums.

Rocky Balboa: I never use 'em.

- Rocky 2, 1979

"Let's leave."

- Thunderbolt & Sung, 2016

sung	(uy2106)	Extreme Programmer
thunderbolt	(awk2123)	Extreme Programmer

Table of Contents

Section I: Outro

Discussion with Prof. Miller Puckette, creator of Pure Data (pd), reveals the following:

"most probably people will want to mix rocky2 (for stuff like iterative algorithms) with straight-up Pd for interactivity design. So even though you're not likely to need the GUI, it would be most useful as a source of 'abstractions' in Pd."

pd is a visual programming language built on and around data flow for music simulation and signal processing. rocky2 is a way to program in pd minus the visual GUI. every rocky2 file compiles to a pd patch.

pd is notoriously poorly documented, and as a result, the authors of rocky2 had to reverse engineer the pd file format to figure out how to translate rocky2 to pd. At this time, it is important to acknowledge the extreme programming techniques employed by the authors in designing and building rocky2.

It is interesting to note that rocky2 can write a multitude of "senseless" pd patches; rocky2 programs that compile nicely into pd code, but don't do anything "useful". We think that this may lead to some very interesting possibilities for artists and musicians working with randomness.

rocky2 is wabisabi & minimalistic. If anything, rocky2 will help pd users make patches more quickly. The authors are hopeful the pd community will devise new ways to use rocky2 and its aesthetic in ways the authors never imagined.

Cheers and yours in all things rocky2,

thunderbolt & sung

Section II: Language Tutorial

Using the rcc compiler:
First, do
chmod +x rcc.sh
This gives rcc.sh the correct permissions so you can run it as an executable.
Three ways to use the rcc compiler (it behaves just like a gcc compiler…):

1. ./rcc.sh
   Code in rocky2 right in the terminal. This is not the most user-friendly interface, but it exists in case you are interested in using it.
   ctrl+d gives the output pd file:

```
$ ./rcc.sh
**** Welcome to Rocky2 Compiler!!! ****
canvas main(100, 100,100,100,10){
        object("hello", 50, 50);
}
#N canvas 100 100 100 100 10;
#X obj 50 50 hello;
```

2. ./rcc.sh example_code/helloworld
   Compile the file helloworld in the subdirectory example_code to example_code/helloworld.pd
3. ./rcc.sh example_code/helloworld samecode_differentfile
   Compile example_code/helloworld file and save the output file as samecode_differentfile
   Note that you can save the compiled output to the same directory as the source file or elsewhere.

Running the source code files to be able to compile rocky2 code without rcc.sh:
$ make build
To compile all source code files.
$ make clean
To remove binaries.

Section III: Language Manual

Scoping: you can only do one level of abstraction in rocky2, unless you use kfi. No subsubpatches, only subpatches. But external patching takes care of any potential issues. In pd environment, you can manually add subsubpatches as desired (and connect this and other components using rocky2) without any problem. Or you can just use kfi for this too...

(subpatch), graph are keywords in pd
pd is another keyword, to denote an internal subpatch that's usable (not a graph…)
Graph subpatching with (subpatch) keyword:

```
#N canvas 0 22 450 300 (subpatch) 0;
#X array excitation 194810 float 0;
#X coords 0 1 194809 -1 100 70 1;
#X restore 334 183 graph;
```

YOU WILL WRITE ABOUT KFI IN HERE...ITS NECESSARY. Explain that it is the user's discretion to use it...based on their knowledge of pd file format. Include any needed pd information at bottom of LRM for reference.

Lessons learnt from patching:

Kfi doesnt deal with variables properly. You cant pass in variables.

Kfi ignores whitespace.

X coords doesnt go at the end of the file

With the object function, whitespace is ignored before the first nonwhitespace character is read, but not after the last nonwhitespace is read.

X and A are reserved $\Rightarrow$ predefined macro variables X and A. Cant declare it nor assign as vairable. DONT SAY WHY → secret appendix for kfi.
Every pd patch needs a main canvas
Object reference number count starts from 0, not 1.

Comma doesnt work because u need an escape sequence for it. We parse commas too, so need /, if you want to display a comma in a Pd file as a message / text, whatever.
Same thing for $ sign. You need \$ to see a $ in pd environment. Use this for passable variables.

You can only define variables inside functions. You have to declare in a separate line (outside canvases for global, inside canvas for local).

Using kfi isn't exactly one-to-one. There's a discrepancy between rocky2 code that uses kfi and pd corresponding code.

We can by extension of how we implemented objects also implement (by including as string arguments the necessary arguments…): Bang, toggle, nbox, vsl, hsl, vradio, hradio, vu, cnv, external subpatch...internal subpatch, kfi can do anything..

Automatic restore after u declare new canvas.

array(12, 13, 141, 151);
#A 12 13 141 151;\r\n

KFI is an easter egg

You shouldn't use the kfi easter egg feature if you are a KFI ;)

Kfi and object() are both ways to declare functions.

When u use kfi(). Use connectS, not connectW.

We have also implemented arrays now via kfi function.

Canvas main (checks if this is present and formtted correctly first of all.

Type checks, declaration function, variable declaration, scope of variables, scope of objects, scope of connections.

Conditional statements, loop statements. Repetition of declaring variables → and also object repition to test connectW and connectS.

Make test = is the test suite. (AUTOMATED).

Lexical Conventions and Parsing

/* */	multi-line comment
=	assignment
+ - * / %	arithmetic operators (plus, minus, times, divide, modulo, respectively)
;	statement ending
{}, ()	function block, scope
<,>, ==, !=, =<, =>	comparators
int, string	primitive data types
void, bool	backend infrastructure data types, unusable / non-declarable
,	separation marker for loops and function parameters
canvas main	main function implements universal canvas
canvas [name]	function declaration for function [name]
object	objectize string values or functions
for, while	loop
if	conditional

Just a note: the ~ (tilde) is a commonly used character in .Pd files. It is used to define specialized audio-rate DSP functional objects, like pink~, osc~, fft~, etc.

Types and Literals

Primitive Types

integer:

A literal such as 50 is a 64-bit signed integer.

We declare integers like:

```
int [name];
```

For example:

```
int a;
```

To assign a value, we must do that in a separate line and within the main / other function block.
For example:

```
int a;
canvas main () {
	a = 1;
}
```

string:

ASCII character sequences enclosed by double quotations and where special characters are handled via a backslash. A literal such as “gug” is a string.

The following escape sequences are supported:

\n newline	\r carriage return
\t horizontal tab	\v vertical tab
\\ backslash	\$ dolla dolla bill yall

We declare strings like:

```
string [name];
```

For example:

```
string puredata;
```

Assigning strings values works the same way assigning integer values does. rocky2 is not strongly typed. Keep this in mind.

Operators and Expressions

Assignment and Variables

= is used to assign values of an expression to an identifier, returning a single assignment. Keep in mind this operation is not associative, so

```
int a = int b = 3;
```

is incorrect.

```
int a;
int b;
a = b = 2;
```

This is possible

```
int a, b;
```

This isn't possible

Arithmetic Operators

These operators are used mostly for "for" loops. Precedence (highest to lowest, separated by comma):

- ! (unary, unary boolean)
* / (binary)
+ - (binary)

All operators are associative left wise. Recall rocky2 includes only integers, so modulus returns only integer values.

Logical and Relational Operators

These operators help in making conditional statements. Precedence (highest to lowest, separated by comma):

>,<
==, !=

=<, =>

Wiring Operator

Pd is about data flow. We want to make it easy for users of rocky2 to be able to connect various objects in a patch and define the flow they want more precisely. With that in mind we have designed two connection functions in rocky2.

Weak connection function

- Used to connect objects by name instead of object identification number.

```
connectW (obj1, outlet, obj2, inlet);
```

So obj1 and obj2 are of type string.

Strong connection function

- Used to connect objects using the object identification number which removes any sort of ambiguity regarding what to connect, since sometimes you have multiple instances of the same object in Pd.

```
connectS (obj_num1, outlet, obj_num2, inlet);
```

So obj_num1, obj_num2 are of type int.

In Pd the object is given an index / count so it has a certain ordering when appearing in the text version of the .Pd file. Similarly, outlets and inlets of an object are counted from 0 to however many there are on a particular object.

Wiring information is taken care of at the end of a universal canvas / .Pd patch. rocky2 first lays out the objects in a user-specified order, then handles connections, or wiring. The user need not worry about wiring things and then defining what it is that needs to be wired later.

Therefore, for example, say there's the following sample code:

```
string a
a  = “pink~”;
/* all declaration(assigning) and object calls must be at the end */
```

```
int b;
int c;
canvas main (100, 100, 100, 100, 10) {
        b = 5;
        c = 10;
        object(a, b, c);
        object("output~", 10, 10);
        /*weak connection via object names and not object id num*/
        connectW(a, 0, "output~", 0);
}
```

Note: when connecting Pd objects, it is important to have an understanding of what that object might look like so that it's easy for you to connect them appropriately (how many outlets, inlets, etc).

Conditional Statements

Conditional statements are formatted similarly to C's if blocks. There is no "else" in rocky2. "if" is the only conditional in rocky2. There is else(even else if is possible; this is automatically done by syntax of our language of if and else statements)

```
if (insert boolean expression here) {}
```

For example (within a function block):

```
a = 0;
if (a < 5) {
        a = a + 1;
}
```

Loops

Looping is probably the most obviously useful feature of rocky2 with respect to Pd. It is extremely easy to patch lots of repeating things together using rocky2 loops.

All loops (for and while) are formatted C-like. For example:

```
while (i < 500) {
       object("mtof", i, 10);
```

```
        i = i + 50;
}

for  (j = 0; j < 8; j = j + 1) {
        connectS(j, 0, k, 0);
        j = j + 1;
        k = k + 1;
}
```

It is important to remember that in the for loop, j = j+1 is needed and j++ and ++j are not supported.

Function Declaration

Default Patching

You may define whatever is necessary for a simple patch in canvas main () {} function block. It is important to introduce all variables necessary outside the scope of the main function block, but one must make sure to define the actual variable values within the main function block itself. Please see above examples / provided sample code to witness this.

Internal Subpatch

To make use of Pd's internal subpatch capability, we define the following format:

```
canvas [function name] (int xPos, int yPos, int xSize, int ySize, string
name_for_internal_subpatch, int font_size) {
         /*statements*/
}
```

To call this function (i.e. internal subpatch) once you have defined it use:

```
function_name (object_name, x_pos, y_pos);
```

External subpatch

To make use of external subpatches that may be needed, simply call them by string value name in your rocky2 program, but also make sure that any previously defined Pd subpatches are located in the same folder as your rocky2 → Pd implementation because otherwise your patch will not be able to use the external subpatch that it calls - Pd environment simply won't recognize the external file if it's not located in the same directory as the main canvas.

For example, to call playsound~.Pd in helloworld, do:

object("playsound~", 0, 0);

where playsound~.Pd exists in the same directory as the main canvas file helloworld

Program Structure and Scope

It's important to know that the current version of rocky2 supports main canvas patching, internal and external subpatching, object handling, and connections, but no objects of a particular type were implemented. Namely messages, numbers, floatatoms, etc. were not incorporated for this release.

File Extension

Pd file extension is .Pd. rocky2 file extension doesn't matter, but we recommend you use dp because it's funny and cool. Like deadpool.

File Format

No particular file format. You can wire things first and define what needs to be wired later even. rocky2 will handle all your Pd evils nicely.

Scoping

- {} block for scope of statements
- () block for scope of functions and for loop setup
- everything inside {} or sometimes () for special cases is considered in the same scope

Structure of Program

Very simply put:

Declare variables, start function definition, define variables, define function as needed, end function definition.

For example:

```
int a;
canvas main () {
        a = 1;
}
```

Built-in Functions / Standard Library

rocky2 incorporates a main function like so:

```
canvas main(int xPos, int yPos, int xSize, int ySize, int font size ) {

        /*statements*/
}
```

The purpose of this main function is to make it easier to bunch together the objects defined in this patch, within a built-in subpatch, and also the external subpatches that exist within the same directory as the rocky2 file you wish to execute.

The execution procedure to compile helloworld (for example) in rocky2 is as follows:

```
make build

cat helloworld | ./rocky2 > helloworld.Pd
```

rocky2 need not take any command line inputs, since ultimately the Pd standalone environment will handle the .Pd file that rocky2 compiles to, compiling it and allowing for user interaction. It is important to stress here that rocky2 at the moment does not allow users to directly create sounds / analyze signals (push and toggle the buttons that Pd lets you play with). For this, one still needs Pd standalone environment.

The Pure Data website http://puredata.info/ has a lot of information regarding various objects.

For the best use of rocky2, the authors suggest downloading the extended Pd package instead of the vanilla version, which lacks the open-source community's contributions to Pd over the years. That way, the possibilities and the application of rocky2 for purposes of enjoyment will be greater enjoyed. http://puredata.info/downloads/Pd-L2Ork

http://puredata.info/downloads/pd-extended
The first link leads to a more stable release of a particular extended Pd.

Section IV: Project Plan

Extreme programming

Our project was decided to be an experiment in extreme programming very early on.

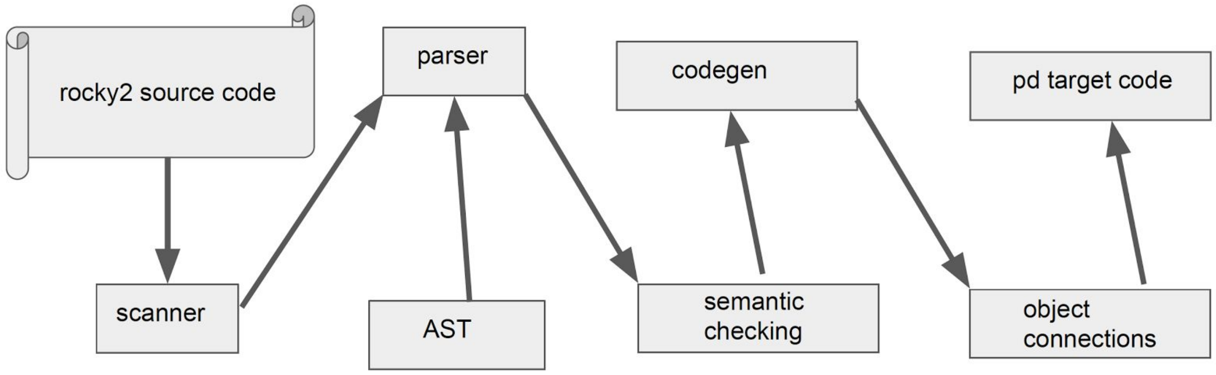
Thus, both authors shared all roles, all at once. From semantics design to implementing the architecture, both authors did everything together. This is their most successful (complete) project to date -- and the authors cannot wait to share rocky2 with the pd world.

February -- March:
Design
Learn OCaml

March--May
Build
Test

The authors finished quite early and had time to further hone in the language to be more robust and more intuitive, which was nice.

Section V: Architectural Design



As discussed in the section 4, project plan, everything was implemented via extreme programming techniques by both authors at once.

Section VI: Test Plan

Testing mostly for rocky2 bugs, The Hello, World! program is also rather limited and nonsensical for what Pd users might consider a good use of Pd.

The test plan was to test mostly for rocky2 bugs, not trying to look at sensibility of the program created, just testing all sorts of rocky2 capabilities. This holds true also for the test rocky2 file in the main directory, as well as the Hello, World! Program in the example_code subdirectory.

As discussed in the section 4, project plan, everything was implemented via extreme programming techniques by both authors at the same time.

Negative tests → senseless even to the point that they dont open in pd. the compiler will show the compilation errors
 -- > connectW (weak) function..negative test yields need for connectS (strong) function.

Positive tests → don't need to have any "purpose" per se, but open in the pd environment without an issue

The example code demonstrates rocky2 code that generates "sensible" "purposeful" pd code.

The following are the test files used. Included are positive test cases, negative test cases, a sample test rocky2 file (separate from the test suite)

test0.r

```
/* thunderbolt & sung */



/*
        basic main
        expected: #N canvas 0 0 0 0 0;
        actual: #N canvas 0 0 0 0 0;
*/
canvas main() {
}
```

test1.r

```
/* thunderbolt & sung */
```

```
/*
	basic main with initializing arguments
	expected: #N canvas 1 2 3 4 5;
	actual: #N canvas 1 2 3 4 5;
*/
canvas main(1,2,3,4,5) {
}
```

test2.r

```
/* thunderbolt & sung */

/*
	negative test: basic main with insufficient arguments
	expected: build-fail: parsing error
	actual: build-fail: parsing error
*/
canvas main(1,2,3,4) {
}
```

test3.r

```
/* thunderbolt & sung */

/*
	test declaraing global variables int & string
	expected: #N canvas 1 2 3 4 5;
	actual: #N canvas 1 2 3 4 5;
*/
int a;
string b;
canvas main(1,2,3,4,5) {
}
```

test4.r

```
/* thunderbolt & sung */

/*
	negative test: declaring and initializing at the same time for
				global variables int & string
```

```
        expected: build-fail: parsing error;
        actual: build-fail: parsing error;
*/
int a = 1;
string b = 2;
canvas main(1,2,3,4,5) {
}
```

test5.r

```
/* thunderbolt & sung */

/*
        test initializing global variables int & string
        expected: #N canvas 1 2 3 4 5;
        actual: #N canvas 1 2 3 4 5;
*/
int a;
string b;
canvas main(1,2,3,4,5) {
        a = 0;
        b = "hello world!";
}
```

test6.r

```
/* thunderbolt & sung */

/*
        negative test: initializing global variables int with string value
                        and string with int value
        expected: illegal assignment;
        actual: illegal assignment;
*/
int a;
string b;
canvas main(1,2,3,4,5) {
        a = "hello world!";     /* illegal assignment int = string in a = "hello world!" */
        b = 0;                  /* illegal assignment string = int in b = 0 */
}
```

test7.r

```
/* thunderbolt & sung */
/*
	test calling built-in function object
	expected:
		#N canvas 1 2 3 4 5;
		#X obj 0 0 foo;
	actual:
		#N canvas 1 2 3 4 5;
		#X obj 0 0 foo;
*/
int a;
string b;
canvas main(1,2,3,4,5) {
	a = 0;
	b = "hello world!";

	object("foo", 0, 0);
}
```

test8.r

```
/* thunderbolt & sung */
/*
	test substituting variables when calling a function
	expected:
		#N canvas 1 2 3 4 5;
		#X obj 0 0 hello world!;
	actual:
		#N canvas 1 2 3 4 5;
		#X obj 0 0 hello world!;
*/
int a;
string b;
canvas main(1,2,3,4,5) {
	a = 0;
	b = "hello world!";

	object(b, a, a);
}
```

test9.r

```
/* thunderbolt & sung */
/*
	negative test: testing invalid arguments
	expected:
		build-fail: illegal argument error
	actual:
		build-fail: illegal argument error
*/
int a;
string b;
canvas main(1,2,3,4,5) {
	a = 0;
	b = "hello world!";

	object("foo~", "these", "should be int values");
}
```

test10.r

```
/* thunderbolt & sung */
/*
	negative test: testing invalid number of arguments
	expected:
		build-fail: expecting 3 arguments
	actual:
		build-fail: expecting 3 arguments
*/
int a;
string b;
canvas main(1,2,3,4,5) {
	a = 0;
	b = "hello world!";

	object("foo~", 0, 0, "object must have only 3 arguments");
}
```

test11.r

```
/* thunderbolt & sung */
```

```
/*
	test conditional statements
	expected:
		#N canvas 1 2 3 4 5;
		#X obj 0 0 true: a == 0;
		#X obj 0 0 false: a == 0;
		#X obj 0 0 else if: a == 2;
		#X obj 0 0 false: a == 0 && a == 2;

	actual:
		#N canvas 1 2 3 4 5;
		#X obj 0 0 true: a == 0;
		#X obj 0 0 false: a == 0;
		#X obj 0 0 else if: a == 2;
		#X obj 0 0 false: a == 0 && a == 2;

*/
int a;
canvas main(1,2,3,4,5) {
	a = 0;
	if (a == 0) {
		object("astrue: a == 0", 0, 0);
		object("asfalse: a == 0", 0, 0);
	} else {
		object("false: a == 0", 0, 0);
	}

	a = 1;
	if (a == 0) {
		object("true: a == 0", 0, 0);
	} else {
		object("false: a == 0", 0, 0);
	}

	a = 2;   /* testing else if */
	if (a == 0) {
		object("true: a == 0", 0, 0);
	} else if (a == 2) {
		object("else if: a == 2", 0, 0);
	} else {
		object("false: a == 0 && a == 2", 0, 0);
	}

	a = 3;
	if (a == 0) {
		object("true: a == 0", 0, 0);
	} else if (a == 2) {
		object("else if: a == 2", 0, 0);
	} else {
		object("false: a == 0 && a == 2", 0, 0);
	}
}
```

test12.r

```
/* thunderbolt & sung */
/*
	negative test: testing calling function with an argument
			with uninitialized variable
	expected:
		build-fail: exception Not_Found
	actual:
		build-fail: exception Not_Found
*/
int a;
string b;
canvas main(1,2,3,4,5) {
	b = "Error!!!";

	object(b, a, 0);
}
```

test13.r

```
/* thunderbolt & sung */
/*
	testing arithematic calculations
	expected:
		#N canvas 1 2 3 4 5;
		#X obj 0 5 a = 0, a + 5 = 5;
```

```
			#X obj 0 -5 a = 0, a - 5 = -5;
			#X obj 1 5 a = 1, a * 5 = 5;
			#X obj 2 1 a = 2, a / 2 = 1;

		actual:
			#N canvas 1 2 3 4 5;
			#X obj 0 5 a = 0, a + 5 = 5;
			#X obj 0 -5 a = 0, a - 5 = -5;
			#X obj 1 5 a = 1, a * 5 = 5;
			#X obj 2 1 a = 2, a / 2 = 1;
*/
int a;
string b;
canvas main(1,2,3,4,5) {
	a = 0;
	b = "a = 0, a + 5 = 5";

	object(b, a, a + 5);
	object("a = 0, (a - 5) = -5", a, (a - 5));

	a = a + 1;
	object("a = a + 1 = 1, a * 5 = 5", a, a * 5);

	a = a + 1;
	object("a = a + 1 = 2, a / 2 = 1", a, a / 2);
}
```

test14.r

```
/* thunderbolt & sung */
/*
	testing declararing local variables
	expected:
		#N canvas 1 2 3 4 5;

	actual:
		#N canvas 1 2 3 4 5;
*/
canvas main(1,2,3,4,5) {
	int a;
	string b;
}
```

test15.r

```
/* thunderbolt & sung */
/*
	negative test: testing calling function with an argument
			with uninitialized local variable
	expected:
		build-fail: exception Not_Found
	actual:
		build-fail: exception Not_Found
*/
canvas main(1,2,3,4,5) {
	int a;
	string b;

	a = 3;

	object(b, a, 0);
}
```

test16.r

```
/* thunderbolt & sung */
/*
	testing using both local and global variables at the same time
	expected:
		#N canvas 1 2 3 4 5;
		#X obj 3 0 Hello World!;
	actual:
		#N canvas 1 2 3 4 5;
		#X obj 3 0 Hello World!;
*/
int a;
string b;
canvas main(1,2,3,4,5) {
/*
	it is possible to have duplicated
	name between global and local variables, but
	it isn't among local variables or global variables
*/
	int a;
	string b;
	int c;

/*
	when there are global and local variables
	with the same name, a local variable has
	the priority to be assigned.
*/
	a = 3;
	b = "Hello World!";
	c = 0;

	object(b, a, c);
}
```

test17.r

```
/* thunderbolt & sung */
/*
	testing subpatch(function declaration)
	expected:
		#N canvas 1 2 3 4 5;
		#X obj 3 0 Hello World!;
	actual:
		#N canvas 1 2 3 4 5;
		#X obj 3 0 Hello World!;
*/
int a;
string b;
canvas main(1,2,3,4,5) {
/*
	assigned to global variables
*/
	a = 3;
	b = "Hello World!";

	object(b, a, 0);
}

canvas subpatch_test() {

}
```

test18.r

```
/* thunderbolt & sung */
/*
	testing subpatch call(user-built function)
	expected:
		#N canvas 1 2 3 4 5;
		#X obj 3 0 Hello World!;
		#N canvas 0 0 0 0 0 test subpatch;
		#X restore 0 0 pd test subpatch;

	actual:
		#N canvas 1 2 3 4 5;
		#X obj 3 0 Hello World!;
		#N canvas 0 0 0 0 0 test subpatch;
		#X restore 0 0 pd test subpatch;
*/
int a;
string b;
canvas main(1,2,3,4,5) {
/*
	assigned to global variables
*/
	a = 3;
	b = "Hello World!";

	object(b, a, 0);
	subpatch_test("test subpatch", 0, 0);
}

canvas subpatch_test() {

}
```

test19.r

```
/* thunderbolt & sung */
/*
	testing object function call in subpatch(user-built function)
	expected:
		#N canvas 1 2 3 4 5;
		#X obj 3 0 Hello World!;
		#N canvas 0 0 0 0 0 test subpatch;
		#X obj 0 0 test object in subpatch;
		#X restore 0 0 pd test subpatch;

	actual:
		#N canvas 1 2 3 4 5;
		#X obj 3 0 Hello World!;
		#N canvas 0 0 0 0 0 test subpatch;
		#X obj 0 0 test object in subpatch;
		#X restore 0 0 pd test subpatch;
*/
int a;
string b;
canvas main(1,2,3,4,5) {
/*
	assigned to global variables
*/
	a = 3;
	b = "Hello World!";

	object(b, a, 0);
	subpatch_test("test subpatch", 0, 0);
}

canvas subpatch_test() {
	object("test object in subpatch", 0, 0);
}
```

test20.r

```
/* thunderbolt & sung */
/*
	testing subpatch with arguments and declaraing local variables
	expected:
		#N canvas 1 2 3 4 5;
		#X obj 3 0 Hello World!;
		#N canvas 0 0 0 0 0 test subpatch;
		#X obj 0 0 test object in subpatch;
		#X restore 0 0 pd test subpatch;

	actual:
		#N canvas 1 2 3 4 5;
		#X obj 3 0 Hello World!;
		#N canvas 0 0 0 0 0 test subpatch;
		#X obj 0 0 test object in subpatch;
		#X restore 0 0 pd test subpatch;
*/
int a;
string b;
/*
	main canvas has 5 arguments
*/
canvas main(1,2,3,4,5) {
/*
	assigned to global variables
*/
	a = 3;
	b = "Hello World!";

	object(b, a, 0);
	subpatch_test("test subpatch", 0, 0);
}

/*
	subpatch canvas has 4 arguments
*/
canvas subpatch_test(2, 3, 4, 5) {
	int a;
	string b;

	object("test object in subpatch", 0, 0);
}
```

test21.r

```
/* thunderbolt & sung */
/*
	negative test: subpatch with argument with variables
	expected:
		build-fail: Parsing Error

	actual:
		build-fail: Parsing Error
*/
int a;
string b;
/*
	main canvas has 5 arguments
*/
canvas main(1,2,3,4,5) {
/*
	assigned to global variables
*/
	a = 3;
	b = "Hello World!";

	object(b, a, 0);
	subpatch_test("test subpatch", 0, 0);
}

/*
	subpatch canvas has 4 arguments
	only literal is applied to arguments
*/
canvas subpatch_test(a, 3, 4, 5) {
	int a;
	string b;

	object("test object in subpatch", 0, 0);
}
```

test22.r

```
/* thunderbolt & sung */
/*
        testing using global variables in subpatch
                #N canvas 1 2 3 4 5;
                #X obj 3 0 Hello World!;
                #N canvas 0 0 0 0 0 test subpatch;
                #X obj 3 0 test global variables a and b, a = 3;
                #X restore 0 0 pd test subpatch;

        actual:
                #N canvas 1 2 3 4 5;
                #X obj 3 0 Hello World!;
                #N canvas 0 0 0 0 0 test subpatch;
                #X obj 3 0 test global variables a and b, a = 3;
                #X restore 0 0 pd test subpatch;
*/
int a;
string b;
/*
        main canvas has 5 arguments
*/
canvas main(1,2,3,4,5) {
/*
        assigned to global variables
*/
        a = 3;
        b = "Hello World!";

        object(b, a, 0);
        subpatch_test("test subpatch", 0, 0);
}

/*
        subpatch canvas has 4 arguments
*/
canvas subpatch_test(2, 3, 4, 5) {
        int a;
        string c;
```

```
	b = "test global variables a and b, a = 3";

/*
	There is local variable a, but in this case, global variable
	a has the priority since local variable a isn't initialized
*/
	object(b, a, 0);
}
```

test23.r

```
/* thunderbolt & sung */
/*
	testing global and local variables with the same name
		#N canvas 1 2 3 4 5;
		#X obj 3 0 Hello World!;
		#N canvas 0 0 0 0 0 test subpatch;
		#X obj 3 0 test global variables a and b, a = 3;
		#X obj 5 0 local variable a = 5, global variable a = 3;
		#X restore 0 0 pd test subpatch;
		#X obj 3 0 a = 3, a != 5;

	actual:
		#N canvas 1 2 3 4 5;
		#X obj 3 0 Hello World!;
		#N canvas 0 0 0 0 0 test subpatch;
		#X obj 3 0 test global variables a and b, a = 3;
		#X obj 5 0 local variable a = 5, global variable a = 3;
		#X restore 0 0 pd test subpatch;
		#X obj 3 0 a = 3, a != 5;
*/
int a;
string b;
/*
	main canvas has 5 arguments
*/
canvas main(1,2,3,4,5) {
/*
	assigned to global variables
*/
```

```
        a = 3;
        b = "Hello World!";

        object(b, a, 0);
        subpatch_test("test subpatch", 0, 0);
/*
        in here variable a still remain as 3
        since it's referring to global variable
*/
        object("a = 3, a != 5", a, 0);
}

/*
        subpatch canvas has 4 arguments
*/
canvas subpatch_test(2, 3, 4, 5) {
        int a;
        string c;

        b = "test global variables a and b, a = 3";

/*
        There is local variable a, but in this case, global variable
        a has the priority since local variable a isn't initialized
*/
        object(b, a, 0);

/*
        initializing local variable a and c
*/
        a = 5;
        c = "local variable a = 5, global variable a = 3";

/*
        local variable has the priority, so this a = 5
*/
        object(c, a, 0);
}
```

test24.r

```
/* thunderbolt & sung */
/*
	testing connectW (connect weak), connecting with object name
		#N canvas 1 2 3 4 5;
		#X obj 3 0 foo~;
		#X obj 3 0 bar~;
		#X connect 0 0 1 0;

	actual:
		#N canvas 1 2 3 4 5;
		#X obj 3 0 foo~;
		#X obj 3 0 bar~;
		#X connect 0 0 1 0;
*/
canvas main(1,2,3,4,5) {
	string o1;
	string o2;
	int x;
	int y;

	o1 = "foo~";
	o2 = "bar~";
	x = 10;
	y = 20;

/*
	each objects is numbered in order of declaration
*/
	object(o1, x, y);			/* object 0 */
	object(o2, x + 30, y + 20);		/* object 1 */

	connectW(o1, 0, o2, 0); /* connect from outlet 0 of "foo~" to inlet 0 of "bar~" */
}
```

test25.r

```
/* thunderbolt & sung */
/*
	negative test: use connectW with the same name of object
```

```
                                        this does not cause build-fail, but cause
                                        logical error
                        #N canvas 1 2 3 4 5;
                        #X obj 3 0 foo~;
                        #X obj 3 0 bar~;
                        #X connect 0 0 0 0;

        actual:
                        #N canvas 1 2 3 4 5;
                        #X obj 3 0 foo~;
                        #X obj 3 0 bar~;
                        #X connect 0 0 0 0;
*/
canvas main(1,2,3,4,5) {
        string o1;
        string o2;
        int x;
        int y;

        o1 = "foo~";
        o2 = "bar~";
        x = 10;
        y = 20;

/*
        each objects is numbered in order of declaration
*/
        object(o1, x, y);                       /* object 0 */
        object(o1, x + 30, y + 20);             /* object 1 */

        connectW(o1, 0, o1, 0);
        /*
                both object is named with "foo~", compiler doesn't know which object
                resulting the first declared object of objects with the same name
        */
}
```

test26.r

```
/* thunderbolt & sung */
```

```
/*
	testing connectS (connect strong), connecting with object number
		#N canvas 1 2 3 4 5;
		#X obj 3 0 foo~;
		#X obj 3 0 bar~;
		#X connect 0 0 1 0;

	actual:
		#N canvas 1 2 3 4 5;
		#X obj 3 0 foo~;
		#X obj 3 0 bar~;
		#X connect 0 0 1 0;
*/
canvas main(1,2,3,4,5) {
	string o1;
	string o2;
	int x;
	int y;

	o1 = "foo~";
	o2 = "bar~";
	x = 10;
	y = 20;

/*
	each objects is numbered in order of declaration
*/
	object(o1, x, y);			/* object 0 */
	object(o2, x + 30, y + 20);		/* object 1 */

	connectS(0, 0, 1, 0); /* connect from outlet 0 of "foo~" to inlet 0 of "bar~" */
}
```

test27.r

```
/* thunderbolt & sung */
/*
	testing connectW, connecting object with subpatch
		#N canvas 1 2 3 4 5;
		#X obj 3 0 foo~;
		#X obj 3 0 bar~;
		#X connect 0 0 1 0;

	actual:
		#N canvas 1 2 3 4 5;
		#X obj 3 0 foo~;
		#X obj 3 0 bar~;
		#X connect 0 0 1 0;
*/
canvas main(1,2,3,4,5) {
	string o1;
	string o2;
	int x;
	int y;

	o1 = "foo~";
	o2 = "bar~";
	x = 10;
	y = 20;

/*
	each objects is numbered in order of declaration
*/
	object(o1, x, y);			/* object 0 */
	sub("subpatch", x - 5, y - 5); /* object 1 */

	connectW(o1, 0, "subpatch", 1);
}

canvas sub() {
	object("jar~", 0, 0);	/* object 2 */
}
```

test28.r

```
/* thunderbolt & sung */
/*
	negative test: connectW, connecting object with object in different canvas
		build-fail
	actual:
		build-fail
*/
canvas main(1,2,3,4,5) {
	string o1;
	string o2;
	int x;
	int y;

	o1 = "foo~";
	o2 = "bar~";
	x = 10;
	y = 20;

/*
	each objects is numbered in order of declaration
*/
	object(o1, x, y);				/* object 0 */
	sub("subpatch", x - 5, y - 5); /* object 1 */

	connectW(o1, 0, "jar~", 1);	/* cannot connect to object in different canvas */
}

canvas sub() {
	object("jar~", 0, 0);	/* object 2 */
}
```

test29.r

```
/* thunderbolt & sung */
/*
	testing loop statements, while
		#N canvas 1 2 3 4 5;
		#X obj 20 40 repeated objects;
		#X obj 30 60 repeated objects;
		#X obj 40 80 repeated objects;
		#X obj 50 100 repeated objects;
		#X obj 60 120 repeated objects;
	actual:
		#N canvas 1 2 3 4 5;
		#X obj 20 40 repeated objects;
		#X obj 30 60 repeated objects;
		#X obj 40 80 repeated objects;
		#X obj 50 100 repeated objects;
		#X obj 60 120 repeated objects;
*/
canvas main(1,2,3,4,5) {
	string o1;
	int x;
	int y;
	int i;

	o1 = "repeated objects";
	x = 10;
	y = 20;
	i = 0;

	while (i < 5) {
		object(o1, x, y);
		x = x + 10;
		y = y + 20;
		i = i + 1;
	}


}
```

test30.r

```
/* thunderbolt & sung */
/*
	testing loop statements, for
		#N canvas 1 2 3 4 5;
		#X obj 20 40 repeated objects;
		#X obj 30 60 repeated objects;
		#X obj 40 80 repeated objects;
		#X obj 50 100 repeated objects;
		#X obj 60 120 repeated objects;
	actual:
		#N canvas 1 2 3 4 5;
		#X obj 20 40 repeated objects;
		#X obj 30 60 repeated objects;
		#X obj 40 80 repeated objects;
		#X obj 50 100 repeated objects;
		#X obj 60 120 repeated objects;
*/
canvas main(1,2,3,4,5) {
	string o1;
	int x;
	int y;
	int i;

	o1 = "repeated objects";
	x = 10;
	y = 20;

	for (i = 0; i < 5; i = i + 1) {
		object(o1, x, y);
		x = x + 10;
		y = y + 20;
	}


}
```

test31.r

```
/* thunderbolt & sung */
/*
	test: loop statements, combination of while and for
		#N canvas 1 2 3 4 5;
		#X obj 20 40 repeated objects;
		#X obj 30 60 repeated objects;
		#X obj 40 80 repeated objects;
		#X obj 50 100 repeated objects;
		#X obj 60 120 repeated objects;
	actual:
		#N canvas 1 2 3 4 5;
		#X obj 10 20 repeated objects;
		#X obj 20 40 repeated objects;
		#X obj 30 60 repeated objects;
		#X obj 40 80 repeated objects;
		#X obj 50 100 repeated objects;
*/
canvas main(1,2,3,4,5) {
	string o1;
	int x;
	int y;
	int i;
	int j;

	o1 = "repeated objects~ 250";
	x = 10;
	y = 20;

	for (i = 0; i < 2; i = i + 1) {
		j = 0;
		while (j < 5) {
			object(o1, x, y);
			x = x + 10;
			y = y + 20;
			j = j + 1;
		}
	}


}
```

test32.r

```
/* thunderbolt & sung */
/*
	To fix the problem of test31.r, it is recommended to initialize all the variables
	before using multi-loop statements
		#N canvas 1 2 3 4 5;
```

```
                #X obj 10 20 repeated objects;
                #X obj 20 40 repeated objects;
                #X obj 30 60 repeated objects;
                #X obj 40 80 repeated objects;
                #X obj 50 100 repeated objects;

        actual:
                #N canvas 1 2 3 4 5;
                #X obj 10 20 repeated objects;
                #X obj 20 40 repeated objects;
                #X obj 30 60 repeated objects;
                #X obj 40 80 repeated objects;
                #X obj 50 100 repeated objects;
*/
canvas main(1,2,3,4,5) {
        string o1;
        int x;
        int y;
        int i;
        int j;

        o1 = "repeated objects";
        x = 10;
        y = 20;
        j = 0;

        for (i = 0; i < 2; i = i + 1) {
                while (j < 5) {
                        object(o1, x, y);
                        x = x + 10;
                        y = y + 20;
                        j = j + 1;
                }
                j = 0;
        }


}
```

Section VII: Lessons Learnt

Thunderbolt’s teachings on inclement weather

It is counter intuitive that the order you put things on the patch is the order they appear in the pd custom format text file (which you can open using the pd environment). The “higher” up they are on the patch isn’t the order they appear in the pd custom format text file.
Number objects as you put them on the patch so you can use the connectS function which is so much more robust than the connectW function (connect objects using their reference numbers, each object has a unique one rather than their names…)

Say youre trying to connect a bunch of stuff with the same name using a for loop for example.

When you’re building a big, beautiful patch, I suggest constantly compiling to Pd / viewing the patch as its being built, because it’s hard to envision what the code might generate visually in the pd environment.

You still need Pd to turn stuff of / on select. Rocky2 just builds the patch for you, so you can focus on playing the patch :)

Discuss plans to release rocky2 to pd community.

Sung’s words of wisdom regarding the extreme

I learnt a ton about using lambda calculus. Also, working with friends is awesome.

Section VIII: Appendix A

This appendix contains all the source files for rocky2. They are presented below in terms of alphabetical order.

ast.ml

```
(* thunderbolt & sung *)

(* Abstract Syntax Tree and functions for printing it *)

type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq |
          And | Or

type uop = Neg | Not

type typ = Int | Bool | Void | String

type bind = typ * string

type expr =
    Literal of int
  | BoolLit of bool
  | Id of string
  | Str of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr
  | Call of string * expr list
  | Noexpr

type stmt =
    Block of stmt list
  | Expr of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt

type func_decl = {
    fname : string;
    formals_opt : int list;
    locals : bind list;
    body : stmt list;
  }

type program = bind list * func_decl list

(* Pretty-printing functions *)
let string_of_typ = function
            Int -> "int"
           | Bool -> "bool"
           | Void -> "void"
           | String -> "string"

let string_of_op = function
            Add -> "+"
           | Sub -> "-"
           | Mult -> "*"
           | Div -> "/"
           | Equal -> "=="
           | Neq -> "!="
           | Less -> "<"
           | Leq -> "<="
           | Greater -> ">"
           | Geq -> ">="
           | And -> "&&"
           | Or -> "||"

let string_of_uop = function
            Neg -> "-"
           | Not -> "!"

let rec string_of_expr = function
            Literal(l) -> string_of_int l
           | BoolLit(true) -> "true"
           | BoolLit(false) -> "false"
           | Id(s) -> s
           | Str(s) -> s
```

```
          | Binop(e1, o, e2) ->
                    string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
          | Unop(o, e) -> string_of_uop o ^ string_of_expr e
          | Assign(v, e) -> v ^ " = " ^ string_of_expr e
          | Call(f, el) ->
                    f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
          | Noexpr -> ""

let rec string_of_stmt = function
   Block(stmts) ->
     "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
     string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
     "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
     string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
```

codegen.ml

```
(* thunderbolt & sung *)

open Ast
open String
module StringHash = Hashtbl.Make(struct
          type t = string
          let equal x y = x = y
          let hash = Hashtbl.hash
          end)
module StringMap = Map.Make(String)

let translate (globals, functions) =
 let count = Array.make 1 0 in
 let global_var = StringHash.create 100 in
 let decl_global = StringHash.create 100 in
 let func = StringMap.empty in

 (**** Declare Global Variables ****)
```

```
List.iter (fun (m, n) -> StringHash.add decl_global ((string_of_typ m) ^ ", " ^ n) "") globals;

(* function declaration *)
let function_decls = List.fold_left (fun m fd -> StringMap.add fd.fname fd m)
                                func functions

in

let main = StringMap.find "main" function_decls in

(*let ptStr = ptStr ^ "#N canvas " ^ g ^ ";\r\n"*)
let add s e = s ^ " " ^ (string_of_int e)

in let ptStr = "#N canvas" ^ (List.fold_left add "" main.formals_opt) ^ ";\r\n"

in

(* count amount of objects *)
let rec object_count funct =
        (* local variable declaration *)
        let decl_local = StringHash.create 100 in
        let local_var = StringHash.create 100 in

        (StringHash.add global_var "string, X" "#X");
        (StringHash.add global_var "string, A" "#A");
        (StringHash.add global_var "string, N" "#N");
        List.iter (fun (m, n) ->
                        StringHash.add decl_local ((string_of_typ m) ^ ", " ^ n) "") funct.locals;
let rec expr_count = function
          Literal n -> string_of_int n
        | BoolLit(true) -> "true"
        | BoolLit(false) -> "false"
        | Id s ->
                (if (StringHash.mem local_var ("int, " ^ s)) then
                        (StringHash.find local_var ("int, " ^ s)) else
                        (if (StringHash.mem local_var ("string, " ^ s)) then
                                (StringHash.find local_var ("string, " ^ s)) else
                                (if (StringHash.mem global_var ("int, " ^ s)) then
                                        (StringHash.find global_var ("int, " ^ s)) else
                                        (if (StringHash.mem global_var ("string, " ^ s)) then
                                                (StringHash.find global_var ("string, " ^ s)) else
                                                (StringHash.find local_var (""))
                        ))))
                | Str s -> let len = (String.length s) - 2 in
                                String.sub s 1 len
                | Binop(e1, op, e2) -> let v1 = expr_count e1 and v2 = expr_count e2 in
                        (match op with
                                 Add -> string_of_int ((int_of_string v1) + (int_of_string v2))
                                | Sub -> string_of_int ((int_of_string v1) - (int_of_string v2))
                                | Mult -> string_of_int ((int_of_string v1) * (int_of_string v2))
                                | Div -> string_of_int ((int_of_string v1) / (int_of_string v2))
                                | Equal -> if (int_of_string v1) = (int_of_string v2)
                                                 then "true" else "false"
                                | Neq -> if (int_of_string v1) != (int_of_string v2)
                                                then "true" else "false"
                                | Less -> if (int_of_string v1) < (int_of_string v2)
                                                then "true" else "false"
                                | Leq -> if (int_of_string v1) <= (int_of_string v2)
                                                then "true" else "false"
                                | Greater -> if (int_of_string v1) > (int_of_string v2)
                                                then "true" else "false"
                                | Geq -> if (int_of_string v1) >= (int_of_string v2)
                                                then "true" else "false"
                                | And ->
                                        if v1 = "true" then
                                                (if v2 = "true" then "true" else "false")
                                        else "false"
                                | Or -> (if v1 = "false" then
                                                (if v2 = "false" then "false" else "ture")
                                                        else "true"))
                | Assign(s, e) -> let v = expr_count e in
                        (if (StringHash.mem decl_local ("int, " ^ s)) then
                                (StringHash.add local_var ("int, " ^ s) v) else
                                (if (StringHash.mem decl_local ("string, " ^ s)) then
                                        (StringHash.add local_var ("string, " ^ s) v) else
                                        (if (StringHash.mem decl_global ("int, " ^ s)) then
                                                (StringHash.add global_var ("int, " ^ s) v) else
                                                (if (StringHash.mem decl_global ("string, " ^ s)) then
                                                        (StringHash.add global_var ("string, " ^ s) v) else
                                                        (StringHash.add local_var "" "") )))) ; ""
                | Unop(op, e) -> let v = expr_count e in
                        (match op with
                                 Neg -> string_of_int (0 - (int_of_string v))
                                | Not -> if v = "true" then "false" else "true")
                | Noexpr -> ""
                | Call(fname, actuals) ->
                        if fname != "connectW" || fname != "connectS" then
                                (count.(0) <- (count.(0) + 1);
                                ignore(if (StringMap.mem fname function_decls) then
                                let internal = StringMap.find fname function_decls in
                                count.(0) <- (count.(0) + 1); (object_count internal) else ""); "")
                        else ""

         and

         stmt_count =
                let rec revStmt = function
                Block st -> List.rev st
                | Expr e -> ignore(expr_count e); []
                | If(p, b1, b2) -> (if (expr_count p) = "true" then (revStmt b1) else (revStmt b2))
                | For(e1, e2, e3, st) -> revStmt st
                | While(p, s) -> revStmt s
                in function
                Block st -> let rec check_block_count = function
                                        Block sl :: ss ->  (check_block_count sl) ^ (check_block_count
        ss)
                                        | s :: ss -> (stmt_count s) ^ (check_block_count ss)
                                        | [] -> ""
                                        in check_block_count st
                | Expr e -> expr_count e
                | If(p, b1, b2) -> (if (expr_count p) = "true" then stmt_count (Block (revStmt b1))
                                        else stmt_count (Block (revStmt b2)))
                | For(e1, e2, e3, st) -> let sts = expr_count e1 in
                                                let rec loop sen =
                                                        if (expr_count e2) = "true" then
                                let tem = sen ^ stmt_count (Block (revStmt st)) ^ (expr_count e3) in
                                                               loop tem
```

```
                                        else sen
                                   in loop sts
          | While(p, s) -> let rec loop sen =
                              if (expr_count p) = "true" then
                                   let tem = sen ^ stmt_count (Block (revStmt s)) in
                                   loop tem
                              else sen
                         in loop ""

in
ignore(stmt_count (Block funct.body)); string_of_int count.(0)

in
let obj = Array.make (int_of_string (object_count main)) "" in
let global_vars = StringHash.create 100 in
let i = Array.make 1 0 in

(* saving object strings in array *)
let rec create_local funct =
          (* local variable declaration *)
          let decl_local = StringHash.create 100 in
          let local_var = StringHash.create 100 in

          (StringHash.add global_vars "string, X" "#X");
          (StringHash.add global_vars "string, A" "#A");
          (StringHash.add global_vars "string, N" "#N");
          List.iter (fun (m, n) ->
                    StringHash.add decl_local ((string_of_typ m) ^ ", " ^ n) "") funct.locals;
let rec expr = function
           Literal n -> string_of_int n
          | BoolLit(true) -> "true"
          | BoolLit(false) -> "false"
          | Id s ->
               (if (StringHash.mem local_var ("int, " ^ s)) then
                    (StringHash.find local_var ("int, " ^ s)) else
                    (if (StringHash.mem local_var ("string, " ^ s)) then
                         (StringHash.find local_var ("string, " ^ s)) else
                         (if (StringHash.mem global_vars ("int, " ^ s)) then
                              (StringHash.find global_vars ("int, " ^ s)) else
```

```
                                        (if (StringHash.mem global_vars ("string, " ^ s)) then
                                                (StringHash.find global_vars ("string, " ^ s)) else
                                (StringHash.find local_var (""))
                ))))
        | Str s -> let len = (String.length s) - 2 in
                        String.sub s 1 len
        | Binop(e1, op, e2) -> let v1 = expr e1 and v2 = expr e2 in
                (match op with
                         Add -> string_of_int ((int_of_string v1) + (int_of_string v2))
                        | Sub -> string_of_int ((int_of_string v1) - (int_of_string v2))
                        | Mult -> string_of_int ((int_of_string v1) * (int_of_string v2))
                        | Div -> string_of_int ((int_of_string v1) / (int_of_string v2))
                        | Equal -> if (int_of_string v1) = (int_of_string v2)
                                        then "true" else "false"
                        | Neq -> if (int_of_string v1) != (int_of_string v2)
                                        then "true" else "false"
                        | Less -> if (int_of_string v1) < (int_of_string v2)
                                        then "true" else "false"
                        | Leq -> if (int_of_string v1) <= (int_of_string v2)
                                        then "true" else "false"
                        | Greater -> if (int_of_string v1) > (int_of_string v2)
                                        then "true" else "false"
                        | Geq -> if (int_of_string v1) >= (int_of_string v2)
                                        then "true" else "false"
                        | And ->
                                if v1 = "true" then
                                        (if v2 = "true" then "true" else "false")
                                else "false"
                        | Or -> (if v1 = "false" then
                                        (if v2 = "false" then "false" else "ture")
                                                else "true"))
        | Assign(s, e) -> let v = expr e in
                (if (StringHash.mem decl_local ("int, " ^ s)) then
                        (StringHash.add local_var ("int, " ^ s) v) else
                        (if (StringHash.mem decl_local ("string, " ^ s)) then
                                (StringHash.add local_var ("string, " ^ s) v) else
                                (if (StringHash.mem decl_global ("int, " ^ s)) then
                                        (StringHash.add global_vars ("int, " ^ s) v) else
                                        (if (StringHash.mem decl_global ("string, " ^ s)) then
                                        (StringHash.add global_vars ("string, " ^ s) v) else
                                        (StringHash.add local_var "" "") )))) ; ""
        | Unop(op, e) -> let v = expr e in
                (match op with
                        Neg -> string_of_int (0 - (int_of_string v))
                    | Not -> if v = "true" then "false" else "true")
        | Noexpr -> ""
        | Call(fname, actuals) ->
                if fname != "connectW" || fname != "connectS" then
                let k = i.(0) in
                i.(0) <- (i.(0) + 1);
                (if fname = "object" then
                    let a1 = List.hd actuals and a2 = List.tl actuals in
                    let a2 = List.hd a2 and a3 = List.tl a2 in
                    let a3 = List.hd a3 in
                    let a1 = expr a1 and a2 = expr a2 and a3 = expr a3 in
                    obj.(k) <- ("#X obj " ^ a2 ^ " " ^ a3 ^ " " ^ a1 ^ ";\r\n"); obj.(k) else
                    (if (StringMap.mem fname function_decls) then
                        let a1 = List.hd actuals and a2 = List.tl actuals in
                        let a2 = List.hd a2 and a3 = List.tl a2 in
                        let a3 = List.hd a3 in
                        let a1 = expr a1 and a2 = expr a2 and a3 = expr a3 in
                        let internal = StringMap.find fname function_decls in
                        obj.(k) <- "#N canvas" ^ (List.fold_left add ""
internal.formals_opt)
                            ^ " " ^ a1 ^ ";\r\n"; ignore(create_local internal);
                        let k = i.(0) in
                        i.(0) <- (i.(0) + 1);
                        obj.(k) <- "#X restore " ^ a2 ^ " " ^ a3 ^ " " ^
                            "pd " ^ a1 ^ ";\r\n"; obj.(k)
                    else
                        (if fname = "kfi" then
                        let a1 = List.hd actuals and a2 = List.tl actuals in
                        let a1 = expr a1 and a2 = List.hd a2 in
                        let a2 = expr a2 in
                        obj.(k) <- (a1 ^ " " ^ a2 ^ ";\r\n"); obj.(k) else "")))
                else ""

and
```

```
stmt =
        let rec revStmt = function
        Block st -> List.rev st
        | Expr e -> ignore(expr e); []
        | If(p, b1, b2) -> (if (expr p) = "true" then (revStmt b1) else (revStmt b2))
        | For(e1, e2, e3, st) -> revStmt st
        | While(p, s) -> revStmt s
        in function
        Block st -> let rec check_block = function
                                Block sl :: ss ->  (check_block sl) ^ (check_block ss)
                               | s :: ss -> (stmt s) ^ (check_block ss)
                               | [] -> ""
                               in check_block st
        | Expr e -> expr e
        | If(p, b1, b2) -> (if (expr p) = "true" then stmt (Block (revStmt b1))
                               else stmt (Block (revStmt b2)))
        | For(e1, e2, e3, st) -> let rec loop temp =
                                               if (expr e2) = "true" then
                                                       (ignore(stmt (Block (revStmt st)));
                                                       ignore(expr e3); loop "")
                                               else temp
                                       in loop (expr e1)
        | While(p, s) -> let rec loop temp =
                               if (expr p) = "true" then
                                       (ignore(stmt (Block (revStmt s)));
                                       loop temp)
                               else temp
                         in loop ""

in
(stmt (Block funct.body))

in ignore(create_local main);

let test = Array.to_list obj in
let add_str s e = s ^ "" ^ e in

ptStr ^ (List.fold_left add_str "" test)
```

connect.ml

```
(* thunderbolt & sung *)

open Ast
open String
module StringHash = Hashtbl.Make(struct
        type t = string
        let equal x y = x = y
        let hash = Hashtbl.hash
        end)
module StringMap = Map.Make(String)

let connect (globals, functions) =
 let count = Array.make 1 0 in
 let count_conn = Array.make 1 0 in
 let global_var = StringHash.create 100 in
 let decl_global = StringHash.create 100 in
 let func = StringMap.empty in


 (**** Declare Global Variables ****)
 (StringHash.add global_var "string, X" "#X");
 (StringHash.add global_var "string, A" "#A");
 (StringHash.add global_var "string, N" "#N");
 List.iter (fun (m, n) -> StringHash.add decl_global ((string_of_typ m) ^ ", " ^ n) "") globals;

 (* function declaration *)
 let function_decls = List.fold_left (fun m fd -> StringMap.add fd.fname fd m)
                                  func functions

 in

 let main = StringMap.find "main" function_decls in

 (* count amount of objects *)
 let rec object_count funct =
        (* local variable declaration *)
        let decl_local = StringHash.create 100 in
        let local_var = StringHash.create 100 in

        (StringHash.add global_var "string, X" "#X");
        (StringHash.add global_var "string, A" "#A");
        (StringHash.add global_var "string, N" "#N");
        List.iter (fun (m, n) ->
                        StringHash.add decl_local ((string_of_typ m) ^ ", " ^ n) "") funct.locals;
let rec expr_count = function
          Literal n -> string_of_int n
        | BoolLit(true) -> "true"
        | BoolLit(false) -> "false"
        | Id s ->
                (if (StringHash.mem local_var ("int, " ^ s)) then
                        (StringHash.find local_var ("int, " ^ s)) else
                        (if (StringHash.mem local_var ("string, " ^ s)) then
                                (StringHash.find local_var ("string, " ^ s)) else
                                (if (StringHash.mem global_var ("int, " ^ s)) then
                                        (StringHash.find global_var ("int, " ^ s)) else
                                        (if (StringHash.mem global_var ("string, " ^ s)) then
                                                (StringHash.find global_var ("string, " ^ s)) else
                                                (StringHash.find local_var (""))
                ))))
        | Str s -> let len = (String.length s) - 2 in
                        String.sub s 1 len
        | Binop(e1, op, e2) -> let v1 = expr_count e1 and v2 = expr_count e2 in
                (match op with
                          Add -> string_of_int ((int_of_string v1) + (int_of_string v2))
                        | Sub -> string_of_int ((int_of_string v1) - (int_of_string v2))
                        | Mult -> string_of_int ((int_of_string v1) * (int_of_string v2))
                        | Div -> string_of_int ((int_of_string v1) / (int_of_string v2))
                        | Equal -> if (int_of_string v1) = (int_of_string v2)
                                        then "true" else "false"
                        | Neq -> if (int_of_string v1) != (int_of_string v2)
                                        then "true" else "false"
                        | Less -> if (int_of_string v1) < (int_of_string v2)
                                        then "true" else "false"
                        | Leq -> if (int_of_string v1) <= (int_of_string v2)
                                        then "true" else "false"
                        | Greater -> if (int_of_string v1) > (int_of_string v2)
```

```
                                then "true" else "false"
                    | Geq -> if (int_of_string v1) >= (int_of_string v2)
                                then "true" else "false"
                    | And ->
                            if v1 = "true" then
                                (if v2 = "true" then "true" else "false")
                            else "false"
                    | Or -> (if v1 = "false" then
                                (if v2 = "false" then "false" else "ture")
                                    else "true"))
        | Assign(s, e) -> let v = expr_count e in
                (if (StringHash.mem decl_local ("int, " ^ s)) then
                    (StringHash.add local_var ("int, " ^ s) v) else
                    (if (StringHash.mem decl_local ("string, " ^ s)) then
                        (StringHash.add local_var ("string, " ^ s) v) else
                        (if (StringHash.mem decl_global ("int, " ^ s)) then
                            (StringHash.add global_var ("int, " ^ s) v) else
                            (if (StringHash.mem decl_global ("string, " ^ s)) then
                                (StringHash.add global_var ("string, " ^ s) v) else
                                (StringHash.add local_var "" "") )))) ; ""
        | Unop(op, e) -> let v = expr_count e in
                (match op with
                     Neg -> string_of_int (0 - (int_of_string v))
                    | Not -> if v = "true" then "false" else "true")
        | Noexpr -> ""
        | Call(fname, actuals) ->
                if fname = "connectW" || fname = "connectS" then
                    (count_conn.(0) <- (count_conn.(0) + 1); "")
                else
                    (count.(0) <- (count.(0) + 1);
                    ignore(if (StringMap.mem fname function_decls) then
                        let internal = StringMap.find fname function_decls in
                        (object_count internal) else ""); "")


and

stmt_count =
        let rec revStmt = function
```

```
          Block st -> List.rev st
          | Expr e -> ignore(expr_count e); []
          | If(p, b1, b2) -> (if (expr_count p) = "true" then (revStmt b1) else (revStmt b2))
          | For(e1, e2, e3, st) -> revStmt st
          | While(p, s) -> revStmt s
          in function
          Block st -> let rec check_block_count = function
                                   Block sl :: ss ->  (check_block_count sl) ^ (check_block_count
ss)
                                  | s :: ss -> (stmt_count s) ^ (check_block_count ss)
                                  | [] -> ""
                                  in check_block_count st
          | Expr e -> expr_count e
          | If(p, b1, b2) -> (if (expr_count p) = "true" then (stmt_count b1) else (stmt_count b2))
          | For(e1, e2, e3, st) -> let sts = expr_count e1 in
                                          let rec loop sen =
                                                  if (expr_count e2) = "true" then
                              let tem = sen ^ stmt_count (Block (revStmt st)) ^ (expr_count e3) in
                                                          loop tem
                                                  else sen
                                          in loop sts
          | While(p, s) -> let rec loop sen =
                                  if (expr_count p) = "true" then
                                          let tem = sen ^ stmt_count (Block (revStmt s)) in
                                          loop tem
                                  else sen
                                in loop ""

 in
 ignore(stmt_count (Block funct.body)); string_of_int count.(0) ^ "," ^ string_of_int
count_conn.(0)

 in
 let count_values = object_count main in

 let splitP = String.index count_values ',' in
 let lent = String.length count_values - 1 in

 let objc = String.sub count_values 0 splitP in
let connc = String.sub count_values (splitP + 1) (lent - splitP) in

let obj = Array.make (int_of_string objc) "" in
let conn = Array.make (int_of_string connc) "" in
let i = Array.make 1 0 in
let j = Array.make 1 0 in

(* saving object strings in array *)
let rec create_local funct =
        let rec find_str key lists n =
                let len = List.length lists in
                (if len < 2 then
                        (if len = 1 then
                                let com = List.hd lists in
                                if key = com then n
                                else raise (Failure ("There is no such object: " ^ key ^ "\n"))
                        else raise (Failure ("There is no such object: " ^ key ^ "\n")))
                else
                        (let coms = List.hd lists and rem = List.tl lists in
                        if key = coms then n else (find_str key rem (n+1))))
        in

        (* local variable declaration *)
        let local_obj = StringHash.create 100 in
        let decl_local = StringHash.create 100 in
        let local_var = StringHash.create 100 in

        (StringHash.add global_var "string, X" "#X");
        (StringHash.add global_var "string, A" "#A");
        (StringHash.add global_var "string, N" "#N");
        List.iter (fun (m, n) ->
                        StringHash.add decl_local ((string_of_typ m) ^ ", " ^ n) "") funct.locals;

let rec expr = function
          Literal n -> string_of_int n
        | BoolLit(true) -> "true"
        | BoolLit(false) -> "false"
        | Id s ->
                (if (StringHash.mem local_var ("int, " ^ s)) then
                        (StringHash.find local_var ("int, " ^ s)) else
                        (if (StringHash.mem local_var ("string, " ^ s)) then
                                (StringHash.find local_var ("string, " ^ s)) else
                                (if (StringHash.mem global_var ("int, " ^ s)) then
                                        (StringHash.find global_var ("int, " ^ s)) else
                                        (if (StringHash.mem global_var ("string, " ^ s)) then
                                                (StringHash.find global_var ("string, " ^ s)) else
                                                (StringHash.find local_var (""))
                ))))
        | Str s -> let len = (String.length s) - 2 in
                        String.sub s 1 len
        | Binop(e1, op, e2) -> let v1 = expr e1 and v2 = expr e2 in
                (match op with
                          Add -> string_of_int ((int_of_string v1) + (int_of_string v2))
                        | Sub -> string_of_int ((int_of_string v1) - (int_of_string v2))
                        | Mult -> string_of_int ((int_of_string v1) * (int_of_string v2))
                        | Div -> string_of_int ((int_of_string v1) / (int_of_string v2))
                        | Equal -> if (int_of_string v1) = (int_of_string v2)
                                        then "true" else "false"
                        | Neq -> if (int_of_string v1) != (int_of_string v2)
                                        then "true" else "false"
                        | Less -> if (int_of_string v1) < (int_of_string v2)
                                        then "true" else "false"
                        | Leq -> if (int_of_string v1) <= (int_of_string v2)
                                        then "true" else "false"
                        | Greater -> if (int_of_string v1) > (int_of_string v2)
                                        then "true" else "false"
                        | Geq -> if (int_of_string v1) >= (int_of_string v2)
                                        then "true" else "false"
                        | And ->
                                if v1 = "true" then
                                        (if v2 = "true" then "true" else "false")
                                else "false"
                        | Or -> (if v1 = "false" then
                                        (if v2 = "false" then "false" else "ture")
                                                else "true"))
        | Assign(s, e) -> let v = expr e in
                (if (StringHash.mem decl_local ("int, " ^ s)) then
                        (StringHash.add local_var ("int, " ^ s) v) else
                        (if (StringHash.mem decl_local ("string, " ^ s)) then
                                (StringHash.add local_var ("string, " ^ s) v) else
                                (if (StringHash.mem decl_global ("int, " ^ s)) then
                                        (StringHash.add global_var ("int, " ^ s) v) else
                                        (if (StringHash.mem decl_global ("string, " ^ s)) then
                                                (StringHash.add global_var ("string, " ^ s) v) else
                                                (StringHash.add local_var "" "") )))) ; ""
        | Unop(op, e) -> let v = expr e in
                (match op with
                          Neg -> string_of_int (0 - (int_of_string v))
                        | Not -> if v = "true" then "false" else "true")
        | Noexpr -> ""
        | Call(fname, actuals) ->
                (if fname = "object" then
                        let k = i.(0) in
                        i.(0) <- (i.(0) + 1);
                        let a1 = List.hd actuals in
                        let a1 = expr a1 in
                        (StringHash.add local_obj a1 ""); obj.(k) <- (a1); obj.(k)
                else
                 if fname = "connectW" then
                        let l = j.(0) in
                        j.(0) <- (j.(0) + 1);
                        let a1 = List.hd actuals and a2 = List.tl actuals in
                        let a2 = List.hd a2 and a3 = List.tl a2 in
                        let a3 = List.hd a3 and a4 = List.tl a3 in
                        let a4 = List.hd a4 in
                        let a1 = expr a1 and a2 = expr a2 and a3 = expr a3 and a4 = expr a4 in
                        let obj_list = Array.to_list obj in
                        let obj1 = string_of_int (find_str a1 obj_list 0)
                        and obj2 = string_of_int (find_str a3 obj_list 0) in
                        (if (StringHash.mem local_obj a1) && (StringHash.mem local_obj a3)
    then
                        (conn.(l) <- ("#X connect " ^ obj1 ^ " " ^ a2 ^ " " ^ obj2 ^ " " ^ a4^
    ";\r\n"))
                        else (raise (Failure ("There is no such object either: " ^ a1 ^ " or " ^ a3))));
                        conn.(l)
                 else if fname = "connectS" then
                        let l = j.(0) in
```

```
                              j.(0) <- (j.(0) + 1);
                              let a1 = List.hd actuals and a2 = List.tl actuals in
                              let a2 = List.hd a2 and a3 = List.tl a2 in
                              let a3 = List.hd a3 and a4 = List.tl a3 in
                              let a4 = List.hd a4 in
                              let a1 = expr a1 and a2 = expr a2 and a3 = expr a3 and a4 = expr a4 in
                              let obj1 = obj.((int_of_string a1))
                              and obj2 = obj.((int_of_string a3)) in
                              (if (StringHash.mem local_obj obj1) && (StringHash.mem local_obj
obj2) then
                              (conn.(l) <- ("#X connect " ^ a1 ^ " " ^ a2 ^ " " ^ a3 ^ " " ^ a4^ ";\r\n"))
                              else (raise (Failure ("There is no such object either: " ^ a1 ^ " or " ^ a3))));
                              conn.(l)
                        else
                        (if (StringMap.mem fname function_decls) then
                              let k = i.(0) in
                              i.(0) <- (i.(0) + 1);
                              let a1 = List.hd actuals in
                              let a1 = expr a1 in
                              let internal = StringMap.find fname function_decls in
                              (StringHash.add local_obj a1 "");
                              obj.(k) <- (a1); ignore(create_local internal);
                              obj.(k)
                        else
                              let k = i.(0) in
                              i.(0) <- (i.(0) + 1);
                              let a1 = List.hd actuals in
                              let a1 = expr a1 in
                              (StringHash.add local_obj a1 ""); obj.(k) <- (a1); obj.(k)))
and

stmt =
      let rec revStmt = function
      Block st -> List.rev st
      | Expr e -> ignore(expr e); []
      | If(p, b1, b2) -> (if (expr p) = "true" then (revStmt b1) else (revStmt b2))
      | For(e1, e2, e3, st) -> revStmt st
      | While(p, s) -> revStmt s
      in function
```

```
        Block st -> let rec check_block = function
                                   Block sl :: ss ->  (check_block sl) ^ (check_block ss)
                                 | s :: ss -> (stmt s) ^ (check_block ss)
                                 | [] -> ""
                                 in check_block st
        | Expr e -> expr e
        | If(p, b1, b2) -> (if (expr p) = "true" then (stmt b1) else (stmt b2))
        | For(e1, e2, e3, st) -> let rec loop temp =
                                                 if (expr e2) = "true" then
                                                          (ignore(stmt (Block (revStmt st)));
                                                          ignore(expr e3); loop "")
                                                 else temp
                                        in loop (expr e1)
        | While(p, s) -> let rec loop temp =
                                 if (expr p) = "true" then
                                        (ignore(stmt (Block (revStmt s)));
                                        loop temp)
                                 else temp
                          in loop ""

 in
 (stmt (Block funct.body))

 in ignore(create_local main);

 let test = Array.to_list conn in
 let add_str s e = s ^ e in

 (List.fold_left add_str "" test)
```

Makefile

```
# thunderbolt & sung

.PHONY : lpc
lpc :
#First construct dependencies for lpc.pd, then compile lpc.pd from lpc rocky2 code.
# lpc
        cat ./example_code/audiosel~ | ./rocky2 > ./example_code/audiosel~.pd
```

```
	cat ./example_code/loadsoundfile | ./rocky2 > ./example_code/loadsoundfile.pd
	cat ./example_code/playloop | ./rocky2 > ./example_code/playloop.pd
	cat ./example_code/myenv | ./rocky2 > ./example_code/myenv.pd
	cat ./example_code/lpc | ./rocky2 > ./example_code/lpc.pd

.PHONY : helloworld
#Simple Hello, World! program
helloworld :
	cat ./example_code/helloworld | ./rocky2 > ./example_code/helloworld.pd

.PHONY : test
test :
# Negative tests which cause build-fail aren't included
# negative tests:	test2.r, test4.r, test6.r, test9.r test10.r, test12.r
#			test15.r test21.r test28.r
# test28 connectW testing, connect to object in different canvas
	cat ./test_cases/test0.r | ./rocky2 > ./test_cases/test0.pd
	cat ./test_cases/test1.r | ./rocky2 > ./test_cases/test1.pd
	cat ./test_cases/test3.r | ./rocky2 > ./test_cases/test3.pd
	cat ./test_cases/test5.r | ./rocky2 > ./test_cases/test5.pd
	cat ./test_cases/test7.r | ./rocky2 > ./test_cases/test7.pd
	cat ./test_cases/test8.r | ./rocky2 > ./test_cases/test8.pd
	cat ./test_cases/test11.r | ./rocky2 > ./test_cases/test11.pd
	cat ./test_cases/test13.r | ./rocky2 > ./test_cases/test13.pd
	cat ./test_cases/test14.r | ./rocky2 > ./test_cases/test14.pd
	cat ./test_cases/test16.r | ./rocky2 > ./test_cases/test16.pd
	cat ./test_cases/test17.r | ./rocky2 > ./test_cases/test17.pd
	cat ./test_cases/test18.r | ./rocky2 > ./test_cases/test18.pd
	cat ./test_cases/test19.r | ./rocky2 > ./test_cases/test19.pd
	cat ./test_cases/test20.r | ./rocky2 > ./test_cases/test20.pd
	cat ./test_cases/test22.r | ./rocky2 > ./test_cases/test22.pd
	cat ./test_cases/test23.r | ./rocky2 > ./test_cases/test23.pd
	cat ./test_cases/test24.r | ./rocky2 > ./test_cases/test24.pd
	cat ./test_cases/test25.r | ./rocky2 > ./test_cases/test25.pd
	cat ./test_cases/test26.r | ./rocky2 > ./test_cases/test26.pd
	cat ./test_cases/test27.r | ./rocky2 > ./test_cases/test27.pd
	cat ./test_cases/test29.r | ./rocky2 > ./test_cases/test29.pd
	cat ./test_cases/test30.r | ./rocky2 > ./test_cases/test30.pd
	cat ./test_cases/test31.r | ./rocky2 > ./test_cases/test31.pd
	cat ./test_cases/test32.r | ./rocky2 > ./test_cases/test32.pd

.PHONY : build
build :
	ocamllex scanner.mll
	ocamlyacc parser.mly
	ocamlc -c ast.ml
	ocamlc -c parser.mli
	ocamlc -c scanner.ml
	ocamlc -c parser.ml
	ocamlc -c semant.ml
	ocamlc -c codegen.ml
	ocamlc -c connect.ml
	ocamlc -c rocky2.ml
	ocamlc -o rocky2 parser.cmo scanner.cmo ast.cmo semant.cmo codegen.cmo
connect.cmo rocky2.cmo


.PHONY : clean
clean :
	ocamlbuild -clean
	rm -rf scanner.ml parser.ml parser.mli
	rm -rf *.cmx *.cmi *.cmo *.cmx *.o
	rm -rf rocky2 *.pd
	rm -rf ./test_cases/*.pd
```

note.txt

```
cat test | ./rocky2 > test.pd
```

parser.mly

```
(* thunderbolt & sung *)

/* Ocamlyacc parser for MicroC */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA
```

```
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token IF ELSE FOR WHILE INT BOOL VOID STRING CANVAS MAIN
%token <int> LITERAL
%token <string> ID
%token <string> STR
%token EOF

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


%start program
%type <Ast.program> program

%%

program:
 decls EOF { $1 }

decls:
  /* nothing */ { [], [] }
 | decls vdecl { ($2 :: fst $1), snd $1 }
 | decls fdecl { fst $1, ($2 :: snd $1) }

fdecl:
  CANVAS ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
    { { fname = $2;
          formals_opt = $4;
          locals = $7;
          body = $8; } }
 | CANVAS MAIN LPAREN main_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
      { { fname = "main";
            formals_opt = $4;
            locals = $7;
            body = $8; } }

formals_opt:
   /* default */ { [0; 0; 0; 0] }
 |  LITERAL COMMA LITERAL COMMA LITERAL COMMA LITERAL { [$1; $3; $5; $7] }

main_opt:
   /* default */ { [0; 0; 0; 0; 0] }
 |  LITERAL COMMA LITERAL COMMA LITERAL COMMA LITERAL COMMA
LITERAL { [$1; $3; $5; $7; $9] }

typ:
   INT { Int }
  | BOOL { Bool }
  | VOID { Void }
  | STRING { String }

vdecl_list:
   /* nothing */    { [] }
  | vdecl_list vdecl { $2 :: $1 }

vdecl:
  typ ID SEMI { ($1, $2) }

stmt_list:
   /* nothing */  { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
   expr SEMI { Expr $1 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
     { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }

expr_opt:
    /* nothing */ { Noexpr }
  | expr          { $1 }

expr:
    LITERAL              { Literal($1) }
  | TRUE                 { BoolLit(true) }
  | FALSE                { BoolLit(false) }
  | ID            { Id($1) }
  | STR                  { Str($1) }
  | expr PLUS   expr     { Binop($1, Add,   $3) }
  | expr MINUS  expr     { Binop($1, Sub,   $3) }
  | expr TIMES  expr     { Binop($1, Mult,  $3) }
  | expr DIVIDE expr     { Binop($1, Div,   $3) }
  | expr EQ     expr     { Binop($1, Equal, $3) }
  | expr NEQ    expr     { Binop($1, Neq,   $3) }
  | expr LT     expr     { Binop($1, Less,  $3) }
  | expr LEQ    expr     { Binop($1, Leq,   $3) }
  | expr GT     expr     { Binop($1, Greater, $3) }
  | expr GEQ    expr     { Binop($1, Geq,   $3) }
  | expr AND    expr     { Binop($1, And,   $3) }
  | expr OR     expr     { Binop($1, Or,    $3) }
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr             { Unop(Not, $2) }
  | ID ASSIGN expr       { Assign($1, $3) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN    { $2 }

actuals_opt:
    /* nothing */ { [] }
  | actuals_list  { List.rev $1 }

actuals_list:
    expr                 { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }
```

rcc.sh

```
#!/bin/bash
count=$#
a=0
b=1
c=2
d="s"
if [ $count -eq $a ]
then
	echo "**** Welcome to Rocky2 Compiler!!! ****"
	./rocky2
elif [ $count -eq $b ]
then
	echo "**** Compile and save as $1.pd ****"
	cat $1 | ./rocky2 > $1.pd
elif [ $count -eq $c ]
then
	echo "**** Compile and save as $2.pd ****"
	cat $1 | ./rocky2 > $2.pd
else
	echo "Wrong number of arguments!!!\n"
fi
```

rocky2.ml

```
(* thunderbolt & sung *)

let _ =
	let lexbuf = Lexing.from_channel stdin in
	let ast = Parser.program Scanner.token lexbuf in
	Semant.check ast;
	let m = (Codegen.translate ast)  ^  (Connect.connect ast) in
	print_string m
```

scanner.mll

```
(* thunderbolt & sung *)

(* Ocamllex scanner for rocky2 *)

{ open Parser }

rule token = parse
```

```
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| [""][^""]*[""] as ldm { STR(ldm) }
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "int"    { INT }
| "bool"   { BOOL }
| "void"   { VOID }
| "true"   { TRUE }
| "false"  { FALSE }
| "string" { STRING }
| "canvas" { CANVAS }
| "main"   { MAIN }
| ['0'-'9']+ as lxm { LITERAL(int_of_string lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '~' '_']* as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }
```

```
and comment = parse
  "*/" { token lexbuf }
| _    { comment lexbuf }
```

semant.ml

```
(* thunderbolt & sung *)

(* semantic checking for the rocky2 compiler *)

open Ast

module StringMap = Map.Make(String)

(* Semantic checking of a program. Returns void if successful,
   throws an exception if something is wrong.

   Check each global variable, then check each function *)

let check (globals, functions) =

        (* Raise an exception if the given list has a duplicate *)
        let report_duplicate exceptf list =
                let rec helper = function
                         n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
                        | _ :: t -> helper t
                        | [] -> ()
                in helper (List.sort compare list)
        in

        (* Raise an exception if a give binding is to a void type *)
        let check_not_void exceptf = function
                  (Void, n) -> raise (Failure (exceptf n))
                | _ -> ()
        in

        (* Raise an exception of the given rvalue type cannot be assigned to
           the given lvalue type *)
        let check_assign lvaluet rvaluet err =
                if lvaluet == rvaluet then lvaluet else raise err
        in

        (**** Checking Global Variables ****)
        List.iter (check_not_void (fun n -> "illegal void global " ^ n)) globals;

        let xa = [(Void, "X"); (Void, "A"); (Void, "N")] @ globals in
        report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd xa);

        (**** Checking Functions ****)
        report_duplicate (fun n -> "duplicate function " ^ n)
                (List.map (fun fd -> fd.fname) functions);

        (* Function declaration for a named function *)
        let built_in_decls = StringMap.add "object"
                { fname = "object"; formals_opt = [0; 0; 0; 0];
                  locals = []; body = [] } (StringMap.add "kfi"
                { fname = "kfi"; formals_opt = [0; 0; 0; 0];
                  locals = []; body = [] }(StringMap.add "connectS"
                { fname = "connectS"; formals_opt = [0; 0; 0; 0];
                  locals = []; body = [] } (StringMap.singleton "connectW"
                { fname = "connectW"; formals_opt = [0; 0; 0; 0];
                  locals = []; body = [] })))

        in

        let function_decls = List.fold_left (fun m fd -> StringMap.add fd.fname fd m)
                                built_in_decls functions
        in

        let function_decl s = try StringMap.find s function_decls
                with Not_found -> raise (Failure ("unrecognized function " ^ s))
        in

        let _ = function_decl "main" in (* Ensure "main" is defined *)

        let check_function func =
                List.iter (check_not_void (fun n -> "illegal void local " ^ n ^
                        " in " ^ func.fname)) func.locals;
                let xa2 = [(Void, "X"); (Void, "A"); (Void, "N")] @ func.locals in
                report_duplicate (fun n -> "duplicate local " ^ n ^ " in " ^ func.fname)
                        (List.map snd xa2);

        (* Type of each variable (global or local *)
        let symbols = List.fold_left (fun m (t, n) -> StringMap.add n t m)
                StringMap.empty (xa @ func.locals)
        in

        let type_of_identifier s =
                try StringMap.find s symbols
                with Not_found -> raise (Failure ("undeclared identifier " ^ s))
        in

        (* Return the type of an expression or throw an exception *)
        let rec expr = function
                  Literal _ -> Int
                | BoolLit _ -> Bool
                | Str _ -> String
                | Id s -> type_of_identifier s
                | Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2 in
                  (match op with
                          Add | Sub | Mult | Div when t1 = Int && t2 = Int -> Int
                        | Equal | Neq when t1 = t2 -> Bool
                        | Less | Leq | Greater | Geq when t1 = Int && t2 = Int -> Bool
                        | And | Or when t1 = Bool && t2 = Bool -> Bool
                        | _ -> raise (Failure ("illegal binary operator " ^
                                string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                                string_of_typ t2 ^ " in " ^ string_of_expr e))
                  )
                | Unop(op, e) as ex -> let t = expr e in
                  (match op with
                          Neg when t = Int -> Int
                        | Not when t = Bool -> Bool
                        | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
                                        string_of_typ t ^ " in " ^ string_of_expr ex)))
                | Noexpr -> Void
                | Assign(var, e) as ex -> let lt = type_of_identifier var
```

```
                        and rt = expr e in
            check_assign lt rt (Failure ("illegal assignment " ^
                        string_of_typ lt ^ " = " ^ string_of_typ rt ^ " in " ^
                        string_of_expr ex))
    | Call(fname, actuals) as call ->
            if (StringMap.mem fname function_decls) = false then
                        raise (Failure("function not exist in " ^
                                    string_of_expr call))
            else
            (if fname = "object" then
              (if  List.length actuals != 3 then
                        raise (Failure("expecting 3 arguments in " ^
                                    string_of_expr call))
               else
                        List.iter2 (fun ft e -> let et = expr e in
                          ignore (check_assign ft et
                            (Failure("illegal actual argument found " ^
                             string_of_typ et ^ " expected " ^ string_of_typ ft
                             ^ " in " ^ string_of_expr e))))
               [String; Int; Int] actuals; Void)
            else
            (if fname = "kfi" then
              (if  List.length actuals != 2 then
                        raise (Failure("expecting 2 arguments in " ^
                                    string_of_expr call))
               else
                        List.iter2 (fun ft e -> let et = expr e in
                          ignore (check_assign ft et
                            (Failure("illegal actual argument found " ^
                             string_of_typ et ^ " expected " ^ string_of_typ ft
                             ^ " in " ^ string_of_expr e))))
                        [Void; String] actuals; Void)
            else
            (if fname = "connectW" then
              (if  List.length actuals != 4 then
                        raise (Failure("expecting 4 arguments in " ^
                                    string_of_expr call))
               else
                        List.iter2 (fun ft e -> let et = expr e in
                  ignore (check_assign ft et
                    (Failure("illegal actual argument found " ^
                     string_of_typ et ^ " expected " ^ string_of_typ ft
                     ^ " in " ^ string_of_expr e))))
                [String; Int; String; Int] actuals; Void)
        else
        (if fname = "connectS" then
          (if  List.length actuals != 4 then
                raise (Failure("expecting 4 arguments in " ^
                        string_of_expr call))
           else
                List.iter2 (fun ft e -> let et = expr e in
                  ignore (check_assign ft et
                    (Failure("illegal actual argument found " ^
                     string_of_typ et ^ " expected " ^ string_of_typ ft
                     ^ " in " ^ string_of_expr e))))
                [Int; Int; Int; Int] actuals; Void)
        else
           (if List.length actuals != 3 then
                raise (Failure("expecting 3 arguments in " ^
                        string_of_expr call))
            else
                List.iter2 (fun ft e -> let et = expr e in
                        ignore (check_assign ft et
                          (Failure ("illegal actual argument found " ^
                          string_of_typ et ^ " expected " ^ string_of_typ
                          ft ^ " in " ^ string_of_expr e))))
                [String; Int; Int] actuals; Void)))))

in

let check_bool_expr e = if expr e != Bool
   then raise (Failure ("expected Boolean expression in " ^ string_of_expr e))
else () in

(* Verify a statement or throw an exception *)
let rec stmt = function
        Block sl -> let rec check_block = function
                Block sl :: ss -> check_block (sl @ ss)
```

```
                    | s :: ss -> stmt s ; check_block ss
                    | [] -> ()
                    in check_block sl
                | Expr e -> ignore (expr e)
                | If(p, b1, b2) -> check_bool_expr p; stmt b1; stmt b2
                | For(e1, e2, e3, st) -> ignore (expr e1); check_bool_expr e2;
                                         ignore (expr e3); stmt st
                | While(p, s) -> check_bool_expr p; stmt s
        in

        stmt (Block func.body)

    in
    List.iter check_function functions
```

Section IX: Appendix B

This appendix contains the sample code displaying a simple Hello, World! Program written in rocky2 (which does not necessarily constitute a program that is considered “sensible” by pd.

helloworld

```
/*thunderbolt & sung*/

/*a simple Hello, World! program in rocky2*/
/*please use rocky2 with love and care*/
/*

yours in all plans simple,

thunderbolt & sung

*/

canvas main(100, 100, 100, 100, 10){
    kfi(X, "text 4 20 Hello \,");
    kfi(X, "text 40 20 World!");
}
```

Section X: Appendix C

This appendix contains all the example code files written in rocky2 to demonstrate an experiment in lpc analysis, in which we excite an input voice signal (sm1-44k.wav) with a music signal (Juno106_Strings.wav). The code is listed below in alphabetical order.

To use this rocky2 → pd patch (and hear the excited voice signal…):

Terminal:

```
$ make lpc
$ open lpc.pd
```

Now, if you have pd-extended installed on your system (http://puredata.info/downloads) , the pd window will open up. In the lpc.pd window, which visually displays the patch written in rocky2, click on the circle above the left loadsoundfile (voice) object, go to the directory in which the lpc.pd that you just compiled (from rocky2's corresponding lpc text file) is located, and select the sm1-44k.wav audio file. Do the same thing for the loadsoundfile excitation object on the right, but choose one of the other wav files (try Juno first...). Note that only wav files work for this patch :)

audiosel~

```
/*thunderbolt & sung*/
/*audiosel~ compiles to audiosel~.pd*/

/* this is an external subpatch that helps lpc.pd main patch */

canvas main(480, 125, 347, 340, 10) {
        object("inlet~", 100, 189); /* object reference number is 0 */
        object("inlet~", 143, 189); /* this one is 1 */
        kfi(X, "obj 144 80 tgl 15 0 empty empty empty 17 7 0 10 -262144 -1 -1 0 1"); /* 2 */
        kfi(X, "obj 161 80 tgl 15 0 empty empty empty 17 7 0 10 -262144 -1 -1 0 1"); /* 3 */
        kfi(X, "msg 97 150 0"); /* 4 */
        connectS(4, 0, 3, 0);
        kfi(X, "msg 138 150 0"); /* 5 */
        connectS(5, 0, 2, 0);
        object("sel 1", 97, 129); /* 6 */
        connectS(2, 0, 6, 0);
        connectS(6, 0, 4, 0);
```

```
object("sel 1", 138, 129); /* 7 */
connectS(3, 0, 7, 0);
connectS(7, 0, 5, 0);
object("*~", 100, 225); /* 8 */
connectS(0, 0, 8, 0);
connectS(2, 0, 8, 1);
object("*~", 143, 225); /* 9 */
connectS(1, 0, 9, 0);
connectS(3, 0, 9, 1);
object("outlet~", 172, 256); /* 10 */
connectS(8, 0, 10, 0);
connectS(9, 0, 10, 0);
object("inlet~", 186, 189); /* 11 */
object("inlet~", 229, 189); /* 12 */
kfi(X, "obj 178 80 tgl 15 0 empty empty empty 17 7 0 10 -262144 -1 -1 0 1"); /* 13 */
connectS(4, 0, 13, 0);
connectS(5, 0, 13, 0);
kfi(X, "obj 195 80 tgl 15 0 empty empty empty 17 7 0 10 -262144 -1 -1 0 1"); /* 14 */
connectS(4, 0, 14, 0);
connectS(5, 0, 14, 0);
kfi(X, "msg 178 150 0"); /* 15 */
kfi(X, "msg 219 150 0"); /* 16 */
object("sel 1", 178, 129); /* 17 */
connectS(13, 0, 17, 0);
object("sel 1", 219, 129); /* 18 */
object("*~", 186, 225); /* 19 */
connectS(11, 0, 19, 0);
connectS(13, 0, 19, 0);
object("*~", 229, 225); /* 20 */
connectS(12, 0, 20, 0);
object("loadbang", 144, 33); /* 21 */
kfi(X, "msg 144 54 1"); /* 22 */
kfi(X, "text 17 -20 audiosel"); /* 23 */
kfi(X, "text 18 -2 Select between 4 audio inputs."); /* 24 */
kfi(X, "text 17 11 Selecting one disables others."); /* 25 */
kfi(X, "text 23 294 thunderbolt & sung"); /* 26 */
object("inlet", 243, -14); /* 27 */
object("sel 1", 243, 9); /* 28 */
object("sel 2", 243, 29); /* 29 */
```

```
	object("sel 3", 243, 49); /* 30 */
	object("sel 4", 243, 69); /* 31 */
	kfi(X, "coords 0 -1 1 1 127 21 1 85 77"); /*no count*/
	connectS(14, 0, 18, 0);
	connectS(14, 0, 20, 0);
	connectS(15, 0, 14, 0);
	connectS(15, 0, 2, 0);
	connectS(15, 0, 3, 0);
	connectS(16, 0, 13, 0);
	connectS(16, 0, 2, 0);
	connectS(16, 0, 3, 0);
	connectS(17, 0, 15, 0);
	connectS(18, 0, 16, 0);
	connectS(19, 0, 10, 0);
	connectS(20, 0, 10, 0);
	connectS(21, 0, 22, 0);
	connectS(22, 0, 2, 0);
	connectS(27, 0, 28, 0);
	connectS(28, 0, 2, 0);
	connectS(28, 1, 29, 0);
	connectS(29, 0, 3, 0);
	connectS(29, 1, 30, 0);
	connectS(30, 0, 13, 0);
	connectS(30, 1, 31, 0);
	connectS(31, 0, 14, 0);
}

loadsoundfile
/*thunderbolt & sung*/
/*loadsoundfile compiles to loadsoundfile.pd*/
/* this is an external subpatch that helps lpc.pd main patch */


int i;
canvas main(735, 33, 450, 300, 10) {
  object("soundfiler", 36, 204); /* 0 */
  object("outlet", 36, 250); /* 1 */
  object("openpanel", 36, 126); /* 2 */
  object("inlet", 36, 80); /* 3 */
  object("/ 44.1", 36, 228); /* 4 */
  kfi(X, "msg 36 180 read -resize \$1 \$2"); /* 5 */
  object("t b b", 36, 103); /* 6 */
  object("symbol \$1", 101, 126); /* 7 */
  object("pack s s", 36, 152); /* 8 */
  kfi(X, "text 4 4 loadsoundfile"); /* 9 */
  kfi(X, "text 86 249 Return value is sound duration in ms"); /* 10 */
  kfi(X, "text 79 80 Inlet is a bang to launch file selection panel"); /* 11 */
  kfi(X, "text 103 152 Stick array name (\$1) onto file name"); /* 12 */
  kfi(X, "text 164 180 Build message"); /* 13 */
  kfi(X, "text 4 22 Opens a file selection panel to get a soundfile \, then loads it into an array.");
/* 14 */
  kfi(X, "text 6 52 Creation arg: \$1 - name of array to write data into"); /* 15 */
  kfi(X, "text 7 279 thunderbolt & sung"); /* 16 */
  i = 0;
  connectS(i, i, 4, i);
  connectS(2, i, 8, i);
  connectS(3, i, 6, i);
  connectS(4, i, 1, i);
  connectS(5, i, 0, i);
  connectS(6, i, 2, i);
  connectS(6, 1, 7, i);
  connectS(7, i, 8, 1);
  connectS(8, i, 5, i);
}
```

lpc

```
/*thunderbolt & sung*/
/*lpc is the main patch you play with*/
/*lpc compiles to lpc.pd*/

/* use kfi for abstraction levels, build from pd code */
/* use for loops, while loops, if to build the main patch objects, connect only this level in front of
edwards. gonna be so sexy */

string lpcanalysis; /*declare*/
int i;
int j;
int count;
int a;
int b;
int c;
int d;

/*lpcanalysis internal subpatch, doesnt open on load*/

canvas lpcan(533, 49, 647, 519){
  /* look scary? dw, this code is heavily based on pd compiled code because the authors are used
to it and can navigate it quite easily*/
  object("inlet~", -6, 104);
  object("rzero~ 0.97", -6, 128);
  object("rpole~ 0.97", 25, 398);
  object("outlet~", 25, 447);
  kfi(X, "text -47 397 De-emphasis");
  kfi(X, "text -85 127 Pre-emphasis");
  object("outlet", 194, 447);
  object("inlet", 108, 104);

  kfi(X, "text 114 464 Residual");
  kfi(X, "text 193 464 LPC coeffs");
  kfi(X, "text 279 36 with 2x (50%) overlap");
  object("samplerate~", 394, 124);
  kfi(X, "floatatom 349 146 5 0 0 0 - - -");
  kfi(X, "obj 349 105 bng 15 250 50 0 empty empty empty 17 7 0 10 -262144 -1 -1");
  kfi(X, "text 343 164 Read back actual SR");
  object("*~", 25, 426);
  object("lpc~ 12", 26, 166);
  object("outlet~", 115, 447);
  kfi(X, "text 279 24 Set block size to 512 points");
  kfi(X, "text 278 49 running at 0.5 of parent SR");
  kfi(X, "text 282 194 512/22050 = 23.2 ms windows");
  object("inlet~", 185, 105);
  object("lpreson~", 25, 337);
  object("*~", 3, 273);
  object("rzero~ 0.97", 185, 129);
  object("*~", 115, 426);
  object("rpole~ 0.97", 115, 398);
  object("*~ 20", 185, 153);
  kfi(X, "text -71 3 LPC Analysis sub-patch");

  kfi(N, "canvas 0 22 450 300 (subpatch) 0");
  kfi(X, "array \$0-hanning 1027 float 1");
  /*who likes arrays*/
  kfi(A, "0 9.41234e-06 0 9.41234e-06 3.7649e-05 8.4709e-05 0.00015059 0.000235291
0.000338807 0.000461135 0.000602271 0.000762208 0.000940942 0.00113846
0.00135477 0.00158985 0.00184369 0.00211629 0.00240763 0.00271771 0.00304651
0.00339402 0.00376023 0.00414512 0.00454867 0.00497089 0.00541174 0.00587121
0.00634928 0.00684594 0.00736117 0.00789494 0.00844724 0.00901805 0.00960734
0.0102151 0.0108413 0.0114859 0.0121489 0.0128303 0.01353 0.014248
0.0149843 0.0157389 0.0165117 0.0173027 0.0181119 0.0189393 0.0197847
0.0206482 0.0215298 0.0224294 0.0233469 0.0242824 0.0252359 0.0262072
0.0271963 0.0282032 0.0292279 0.0302703 0.0313304 0.0324082 0.0335035
0.0346165 0.0357469 0.0368948 0.0380602 0.0392429 0.040443 0.0416604
0.0428951 0.0441469 0.0454159 0.0467021 0.0480053 0.0493255 0.0506627
0.0520168 0.0533878 0.0547755 0.0561801 0.0576014 0.0590393 0.0604938
0.0619649 0.0634524 0.0649564 0.0664768 0.0680135 0.0695664 0.0711356
0.0727209 0.0743223 0.0759397 0.0775731 0.0792224 0.0808875 0.0825684
0.0842651 0.0859773 0.0877052 0.0894486 0.0912074 0.0929817 0.0947712
0.0965761 0.0983961 0.100231 0.102081 0.103947 0.105827 0.107722 0.109631
0.111556 0.113495 0.115448 0.117416 0.119399 0.121395 0.123406 0.125432
0.127471 0.129524 0.131591 0.133673 0.135768 0.137876 0.139999 0.142134
0.144284 0.146446 0.148622 0.150812 0.153014 0.155229 0.157458 0.159699
0.161953 0.16422 0.1665 0.168792 0.171096 0.173413 0.175743 0.178084
0.180437 0.182803 0.185181 0.18757 0.189971 0.192384 0.194808 0.197244
0.199691 0.20215 0.20462 0.207101 0.209593 0.212096 0.214609 0.217134
0.219669 0.222215 0.224771 0.227337 0.229914 0.232501 0.235098 0.237705
0.240322 0.242948 0.245585 0.24823 0.250886 0.253551 0.256225 0.258908
0.2616 0.264301 0.267011 0.26973 0.272458 0.275194 0.277939 0.280691
0.283453 0.286222 0.288999 0.291785 0.294578 0.297379 0.300187 0.303004
0.305827 0.308658 0.311496 0.314341 0.317193 0.320052 0.322918 0.32579
0.328669 0.331555 0.334446 0.337344 0.340248 0.343159 0.346075 0.348997
0.351924 0.354857 0.357796 0.36074 0.363689 0.366643 0.369602 0.372567
0.375536 0.378509 0.381488 0.38447 0.387457 0.390449 0.393444 0.396444
0.399447 0.402454 0.405465 0.408479 0.411497 0.414518 0.417543 0.42057
0.423601 0.426634 0.42967 0.432709 0.43575 0.438794 0.44184 0.444888
0.447939 0.450991 0.454045 0.457101 0.460158 0.463217 0.466277 0.469339
0.472402 0.475466 0.47853 0.481596 0.484662 0.487729 0.490796 0.493864
```

0.496931 0.499999 0.503067 0.506135 0.509203 0.51227 0.515337 0.518403
0.521468 0.524533 0.527597 0.53066 0.533721 0.536782 0.539841 0.542898
0.545954 0.549008 0.55206 0.55511 0.558159 0.561205 0.564248 0.56729
0.570328 0.573364 0.576398 0.579428 0.582456 0.58548 0.588501 0.591519
0.594534 0.597544 0.600552 0.603555 0.606554 0.60955 0.612541 0.615528
0.618511 0.621489 0.624463 0.627432 0.630396 0.633356 0.63631 0.639259
0.642203 0.645142 0.648075 0.651002 0.653924 0.65684 0.65975 0.662654
0.665552 0.668444 0.67133 0.674209 0.677081 0.679947 0.682806 0.685658
0.688503 0.691341 0.694172 0.696995 0.699811 0.70262 0.705421 0.708214
0.710999 0.713777 0.716546 0.719307 0.72206 0.724805 0.727541 0.730269
0.732987 0.735698 0.738399 0.741091 0.743774 0.746448 0.749113 0.751768
0.754414 0.757051 0.759677 0.762294 0.764901 0.767498 0.770085 0.772662
0.775228 0.777784 0.78033 0.782865 0.78539 0.787903 0.790406 0.792898
0.795379 0.797849 0.800308 0.802755 0.805191 0.807615 0.810028 0.812429
0.814818 0.817196 0.819561 0.821915 0.824256 0.826586 0.828903 0.831207
0.833499 0.835779 0.838046 0.8403 0.842541 0.84477 0.846985 0.849187
0.851377 0.853553 0.855715 0.857865 0.860001 0.862123 0.864232 0.866326
0.868408 0.870475 0.872528 0.874568 0.876593 0.878604 0.880601 0.882583
0.884551 0.886505 0.888444 0.890368 0.892278 0.894173 0.896053 0.897918
0.899768 0.901603 0.903423 0.905228 0.907018 0.908792 0.910551 0.912294
0.914022 0.915734 0.917431 0.919112 0.920777 0.922426 0.92406 0.925677
0.927278 0.928864 0.930433 0.931986 0.933523 0.935043 0.936547 0.938035
0.939506 0.94096 0.942398 0.943819 0.945224 0.946612 0.947983 0.949337
0.950674 0.951994 0.953297 0.954584 0.955853 0.957104 0.958339 0.959556
0.960757 0.961939 0.963105 0.964253 0.965383 0.966496 0.967591 0.968669
0.969729 0.970772 0.971796 0.972803 0.973792 0.974764 0.975717 0.976653
0.97757 0.97847 0.979351 0.980215 0.98106 0.981888 0.982697 0.983488
0.984261 0.985015 0.985752 0.98647 0.987169 0.987851 0.988514 0.989158
0.989785 0.990392 0.990982 0.991553 0.992105 0.992639 0.993154 0.99365
0.994129 0.994588 0.995029 0.995451 0.995855 0.99624 0.996606 0.996953
0.997282 0.997592 0.997884 0.998156 0.99841 0.998645 0.998861 0.999059
0.999238 0.999398 0.999539 0.999661 0.999765 0.999849 0.999915 0.999962
0.999991 1 0.999991 0.999962 0.999915 0.999849 0.999765 0.999661 0.999539
0.999398 0.999238 0.999059 0.998862 0.998645 0.99841 0.998156 0.997884
0.997593 0.997282 0.996954 0.996606 0.99624 0.995855 0.995452 0.995029
0.994588 0.994129 0.993651 0.993154 0.992639 0.992105 0.991553 0.990982
0.990393 0.989785 0.989159 0.988514 0.987851 0.98717 0.98647 0.985752
0.985016 0.984261 0.983489 0.982698 0.981888 0.981061 0.980216 0.979352
0.978471 0.977571 0.976653 0.975718 0.974765 0.973793 0.972804 0.971797

0.970773 0.96973 0.96867 0.967592 0.966497 0.965384 0.964254 0.963106
0.96194 0.960758 0.959558 0.95834 0.957105 0.955854 0.954585 0.953299
0.951995 0.950675 0.949338 0.947984 0.946613 0.945225 0.943821 0.942399
0.940961 0.939507 0.938036 0.936548 0.935044 0.933524 0.931987 0.930434
0.928865 0.92728 0.925678 0.924061 0.922428 0.920778 0.919113 0.917432
0.915736 0.914023 0.912296 0.910552 0.908793 0.907019 0.90523 0.903425
0.901605 0.89977 0.897919 0.896054 0.894174 0.892279 0.89037 0.888445
0.886506 0.884553 0.882585 0.880602 0.878605 0.876594 0.874569 0.87253
0.870477 0.868409 0.866328 0.864233 0.862125 0.860002 0.857867 0.855717
0.853555 0.851379 0.849189 0.846987 0.844772 0.842543 0.840302 0.838048
0.835781 0.833501 0.831209 0.828905 0.826588 0.824259 0.821917 0.819564
0.817198 0.81482 0.812431 0.81003 0.807617 0.805193 0.802757 0.80031
0.797851 0.795381 0.7929 0.790408 0.787906 0.785392 0.782867 0.780332
0.777787 0.77523 0.772664 0.770087 0.7675 0.764903 0.762296 0.759679
0.757053 0.754417 0.751771 0.749115 0.746451 0.743777 0.741093 0.738401
0.7357 0.73299 0.730271 0.727543 0.724807 0.722063 0.71931 0.716549
0.713779 0.711002 0.708216 0.705423 0.702622 0.699814 0.696998 0.694174
0.691343 0.688505 0.68566 0.682808 0.679949 0.677083 0.674211 0.671332
0.668447 0.665555 0.662657 0.659753 0.656843 0.653927 0.651005 0.648077
0.645144 0.642206 0.639262 0.636312 0.633358 0.630399 0.627435 0.624466
0.621492 0.618514 0.615531 0.612544 0.609552 0.606557 0.603558 0.600554
0.597547 0.594536 0.591522 0.588504 0.585483 0.582458 0.579431 0.576401
0.573367 0.570331 0.567292 0.564251 0.561207 0.558161 0.555113 0.552063
0.549011 0.545956 0.542901 0.539843 0.536784 0.533724 0.530662 0.5276
0.524536 0.521471 0.518406 0.515339 0.512273 0.509205 0.506138 0.50307
0.500002 0.496934 0.493866 0.490799 0.487731 0.484665 0.481598 0.478533
0.475468 0.472404 0.469342 0.46628 0.46322 0.460161 0.457103 0.454048
0.450993 0.447941 0.444891 0.441843 0.438797 0.435753 0.432712 0.429673
0.426637 0.423603 0.420573 0.417545 0.414521 0.4115 0.408482 0.405468
0.402457 0.39945 0.396446 0.393447 0.390451 0.38746 0.384473 0.38149
0.378512 0.375538 0.372569 0.369605 0.366646 0.363691 0.360742 0.357798
0.35486 0.351927 0.348999 0.346077 0.343161 0.340251 0.337347 0.334449
0.331557 0.328672 0.325793 0.32292 0.320054 0.317196 0.314343 0.311498
0.30866 0.305829 0.303006 0.30019 0.297381 0.29458 0.291787 0.289002
0.286224 0.283455 0.280694 0.277941 0.275196 0.27246 0.269733 0.267014
0.264304 0.261602 0.25891 0.256227 0.253553 0.250888 0.248233 0.245587
0.242951 0.240324 0.237707 0.2351 0.232503 0.229916 0.227339 0.224773
0.222217 0.219671 0.217136 0.214611 0.212098 0.209595 0.207103 0.204622
0.202152 0.199694 0.197246 0.19481 0.192386 0.189973 0.187572 0.185183

```
0.182805 0.18044 0.178086 0.175745 0.173415 0.171098 0.168794 0.166502
0.164222 0.161955 0.159701 0.15746 0.155231 0.153016 0.150814 0.148624
0.146448 0.144286 0.142136 0.14 0.137878 0.135769 0.133674 0.131593
0.129526 0.127473 0.125433 0.123408 0.121397 0.1194 0.117418 0.11545
0.113496 0.111557 0.109633 0.107723 0.105828 0.103948 0.102083 0.100233
0.0983977 0.0965776 0.0947728 0.0929832 0.091209 0.0894501 0.0877067
0.0859788 0.0842665 0.0825699 0.080889 0.0792238 0.0775745 0.0759411
0.0743237 0.0727223 0.0711369 0.0695678 0.0680148 0.0664781 0.0649577
0.0634537 0.0619661 0.0604951 0.0590405 0.0576026 0.0561813 0.0547768
0.053389 0.052018 0.0506638 0.0493266 0.0480064 0.0467032 0.045417
0.044148 0.0428961 0.0416615 0.0404441 0.0392439 0.0380612 0.0368958
0.0357479 0.0346174 0.0335045 0.0324091 0.0313314 0.0302712 0.0292288
0.0282041 0.0271972 0.026208 0.0252367 0.0242833 0.0233477 0.0224302
0.0215306 0.020649 0.0197854 0.01894 0.0181126 0.0173034 0.0165124
0.0157396 0.014985 0.0142487 0.0135306 0.0128309 0.0121495 0.0114865
0.0108418 0.0102156 0.00960786 0.00901855 0.00844773 0.00789541 0.00736162
0.00684638 0.0063497");

  kfi(A, "1000 0.00587161 0.00541213 0.00497126 0.00454903 0.00414546 0.00376055
0.00339433 0.0030468 0.00271799 0.00240789 0.00211653 0.00184392 0.00159006
0.00135496 0.00113864 0.000941105 0.000762355 0.000602401 0.000461249
0.000338905 0.000235372 0.000150656 8.47578e-05 3.76816e-05 9.42863e-06
7.04153e-12 9.39607e-06");

  kfi(X, "coords 0 1 1026 0 100 70 1");
  kfi(X, "restore 284 248 graph");

  /* comment the pd code */
  kfi(X, "text 281 349 Tapered window for overlap-add");
  kfi(X, "text -48 85 Signal to be modeled");
  kfi(X, "text 94 85 Model order");
  kfi(X, "text 177 86 Excitation");
  kfi(X, "text -65 427 Tapered window");
  kfi(X, "text -64 160 Analyze to LPC");
  kfi(X, "text -81 212 Find envelope");
  kfi(X, "text -81 224 of residual");
  kfi(X, "text -7 182 residual");
  kfi(X, "text 54 183 LPC coeffs");
  kfi(X, "text -80 256 Scale excitation");
```

```
 kfi(X, "text -80 267 by envelope");
 kfi(X, "text -66 344 to excitation");
 kfi(X, "text -53 171 filters");
 kfi(X, "text -81 333 Apply LPC filters");
 kfi(X, "text 154 308 Normalize residual");
 kfi(X, "text 163 336 .. and scale down");
 kfi(X, "text 277 69 Block length must match");
 kfi(X, "text 277 80 length of hanning array.");
 kfi(X, "text -72 16 We run this in a sub-patch to take advantage of block~ which allows us to
vary the frame length \, overlap \, and sample rate.");
 kfi(X, "text 187 170 Scale-up");
 kfi(X, "text 183 181 excitation");
 kfi(X, "text 281 206 2x overlap -> 11.6 ms frames");
 kfi(X, "text 19 464 Synthesis");

 /* ext subpatch myenv */
 /* could have used kfi to make this a subsubpatch of lpc.pd*/
 /*object("myenv", 3, 219);*/

 kfi(N, "canvas 320 356 450 300 myenv 0");
 kfi(X, "obj 94 129 pack~");
 kfi(X, "obj 94 151 sum");
 kfi(X, "obj 94 108 *~");
 kfi(X, "obj 94 84 +~ 0");
 kfi(X, "obj 94 208 sqrt");
 kfi(X, "obj 95 229 sig~");
 kfi(X, "obj 92 49 inlet~");
 kfi(X, "obj 95 270 outlet~");
 kfi(X, "obj 94 178 / 512");
 kfi(X, "obj 145 86 iem_blocksize~");
 kfi(X, "obj 178 268 outlet");
 kfi(X, "text 230 269 frame-rate gain");
 kfi(X, "connect 0 0 1 0");
 kfi(X, "connect 1 0 8 0");
 kfi(X, "connect 2 0 0 0");
 kfi(X, "connect 3 0 2 0");
 kfi(X, "connect 3 0 2 1");
 kfi(X, "connect 4 0 5 0");
 kfi(X, "connect 4 0 10 0");
 kfi(X, "connect 5 0 7 0");
 kfi(X, "connect 6 0 3 0");
 kfi(X, "connect 6 0 9 0");
 kfi(X, "connect 8 0 4 0");
 kfi(X, "connect 9 0 8 1");
 kfi(X, "restore 3 219 pd myenv");

 object("list", 67, 278);
 object("t l b", 65, 254);
 object("block~ 1024 2 0.5", 284, 6);
 kfi(X, "text 35 305 one frame");
 kfi(X, "text 34 316 to match gain");
 object("outlet", 275, 447);
 kfi(X, "text 274 463 Gain");
 kfi(X, "msg 284 321 \; \$0-hanning cosinesum 1024 0.5 -0.5");
 object("tabreceive~ \$0-hanning", 284, 365);
 object("/~ 1", 115, 313);
 object("pack~", 115, 255);
 object("unpack~", 115, 275);
 kfi(X, "text 165 251 Delay residual");
 kfi(X, "text 165 263 one block");
 kfi(X, "text 36 296 Delay coeffs");
 kfi(X, "text 164 275 to match gain");
 kfi(X, "text 154 319 by gain (envelope)");
 kfi(X, "text -69 60 thunderbolt & sung");
 object("*~ 0.01", 115, 337);

 /*yes...even connections*/
 kfi(X, "connect 0 0 1 0");
 kfi(X, "connect 1 0 16 0");
 kfi(X, "connect 2 0 15 0");
 kfi(X, "connect 7 0 16 1");
 kfi(X, "connect 11 0 12 0");
 kfi(X, "connect 13 0 11 0");
 kfi(X, "connect 15 0 3 0");
 kfi(X, "connect 16 0 54 0");
 kfi(X, "connect 16 0 65 0");
 kfi(X, "connect 16 1 56 0");
 kfi(X, "connect 21 0 24 0");
```

```
  kfi(X, "connect 22 0 2 0");
  kfi(X, "connect 23 0 22 0");
  kfi(X, "connect 24 0 27 0");
  kfi(X, "connect 25 0 17 0");
  kfi(X, "connect 26 0 25 0");
  kfi(X, "connect 27 0 23 1");
  kfi(X, "connect 54 0 23 0");
  kfi(X, "connect 54 1 60 0");
  kfi(X, "connect 54 1 64 1");
  kfi(X, "connect 55 0 22 0");
  kfi(X, "connect 55 0 6 0");
  kfi(X, "connect 56 0 55 1");
  kfi(X, "connect 56 1 55 0");
  kfi(X, "connect 63 0 15 1");
  kfi(X, "connect 63 0 25 1");
  kfi(X, "connect 64 0 73 0");
  kfi(X, "connect 65 0 66 0");
  kfi(X, "connect 66 0 64 0");
  kfi(X, "connect 73 0 26 0");

}

/* main canvas */

canvas main(13, 108, 496, 416, 10) {
  /*declare any necessary variables*/
  lpcanalysis = "lpcanalysis";

  /* voice graph */
  kfi(N, "canvas 0 22 450 300 (subpatch) 0");
  kfi(X, "array voice 153259 float 0");
  kfi(X, "coords 0 1 153258 -1 100 70 1");
  kfi(X, "restore 334 94 graph");

  /* excitation graph */
  kfi(N, "canvas 0 22 450 300 (subpatch) 0");
  kfi(X, "array excitation 194810 float 0");
  kfi(X, "coords 0 1 194809 -1 100 70 1");
  kfi(X, "restore 334 183 graph");
```

```
/*kfi(X, "TEST");*/
i = 0;
j = 0;
while (i < 1) {
  /* variables don't work with kfi */
  kfi(X, "obj -5 71 bng 15 250 50 0 empty empty empty -38 -6 0 8 -262144 -1 -1");
  kfi(X, "obj 154 71 bng 15 250 50 0 empty empty empty -38 -6 0 8 -262144 -1 -1");
  kfi(X, "obj 41 133 bng 15 250 50 0 empty empty play 0 -6 0 8 -262144 -1 -1");
  kfi(X, "floatatom 219 163 5 1 100 2 - - -");
  kfi(X, "obj 122 133 tgl 15 0 empty empty loop 0 -6 0 8 -262144 -1 -1 0 1");
  object("loadbang", 219, 120);
  j = j + 9;
  object("output~", j, 304);
  kfi(X, "text -37 362 thunderbolt & sung");
  object("loadsoundfile voice", -5, 95);
  lpcan(lpcanalysis, 9, 255);
  object("playloop voice", -4, 163);
  kfi(X, "text 255 169 order");
  /* can even do objects with kfi */
  kfi(X, "obj 90 163 playloop excitation");
  object("loadsoundfile excitation", 153, 94);
  object("noise~", 120, 184);
  kfi(X, "text -39 8 LPC analysis-synthesis");
  kfi(X, "text 151 38 Click here to select sound");
  kfi(X, "text 151 51 for excitation (or use noise)");
  kfi(X, "text -39 38 Click here to select sound");
  kfi(X, "text -40 50 that LPC filter will model");
  kfi(X, "text 255 159 Filter");
  kfi(X, "msg 219 141 24");
  object("audiosel~", 90, 229);
  object("*~ 0.1", 120, 205);
  kfi(X, "text -16 271 synth");
  kfi(X, "text 17 271 resid");
  kfi(X, "text 51 271 coeffs");
  kfi(X, "text -37 379 there's a lot going on here");
  kfi(X, "text -37 391 like lots and lots!!!!!!!!!");
  kfi(X, "text 93 271 gain");
  i = i + 2;
  /*object("TEST", i, i);*/
}

/* handle some connections via rocky2 ... */

/*count = 2;
finishes the loop before exiting even if the cond isnt met
while (count < 5) {
  a = 14;
  b = 0;
  c = 24;
  d = 0;
  connectS(a,b,c,d);
  count = count + 1;
  a = 16;
  b = 0;
  c = 25;
  d = 0;
  connectS(a,b,c,d);
  count = count + 1;
  a = 23;
  b = 0;
  c = 5;
  d = 0;
  connectS(a,b,c,d);
  count = count + 1;
  a = 24;
  b = 0;
  c = 11;
  d = 2;
  connectS(a,b,c,d);
  count = count + 1;
  a = 25;
  b = 0;
  c = 24;
  d = 1;
  connectS(a,b,c,d);
  count = count + 1;
}
```

```
  */
  /*do the rest manually.*/

  kfi(X, "connect 2 0 10 0");
  kfi(X, "connect 3 0 15 0");
  kfi(X, "connect 4 0 12 0");
  kfi(X, "connect 4 0 14 0");
  kfi(X, "connect 5 0 11 1");

  kfi(X, "connect 6 0 12 1");
  kfi(X, "connect 6 0 14 1");
  kfi(X, "connect 7 0 23 0");
  kfi(X, "connect 11 0 8 0");
  kfi(X, "connect 11 0 8 1");
  kfi(X, "connect 12 0 11 0");

  kfi(X, "connect 14 0 24 0");
  kfi(X, "connect 16 0 25 0");
  kfi(X, "connect 23 0 5 0");
  kfi(X, "connect 24 0 11 2");
  kfi(X, "connect 25 0 24 1");
}

myenv
/*thunderbolt & sung*/
/*myenv compiles to myenv.pd*/
/* this is an external subpatch that helps lpcanalysis subpatch  — not actually used, commented
out in the lpc main patch code :) */

/*int i;*/

canvas main(320, 356, 450, 300, 10) {
  /*i = 94;
  while (i < 95) {
    object("pack", i, 129);
    object("",,);

    i = i + 1;
    object("sig~", i, 229);
  }*/
  /*object("TESTTESTTEST", i, i);*/
  kfi(X, "obj 94 129 pack~");
  kfi(X, "obj 94 151 sum");
  kfi(X, "obj 94 108 *~");
  kfi(X, "obj 94 84 +~ 0");
  kfi(X, "obj 94 208 sqrt");
  kfi(X, "obj 95 229 sig~");
  kfi(X, "obj 92 49 inlet~");
  kfi(X, "obj 95 270 outlet~");
  kfi(X, "obj 94 178 / 512");
  kfi(X, "obj 145 86 iem_blocksize~");
  kfi(X, "obj 178 268 outlet");
  kfi(X, "text 230 269 frame-rate gain");
  kfi(X, "connect 0 0 1 0");
  kfi(X, "connect 1 0 8 0");
  kfi(X, "connect 2 0 0 0");
  kfi(X, "connect 3 0 2 0");
  kfi(X, "connect 3 0 2 1");
  kfi(X, "connect 4 0 5 0");
  kfi(X, "connect 4 0 10 0");
  kfi(X, "connect 5 0 7 0");
  kfi(X, "connect 6 0 3 0");
  kfi(X, "connect 6 0 9 0");
  kfi(X, "connect 8 0 4 0");
  kfi(X, "connect 9 0 8 1");
}

playloop
/*thunderbolt & sung*/
/*playloop compiles to playloop.pd*/
/* this is an external subpatch that helps lpc.pd main patch */

string f;
canvas main(416, 342, 415, 285, 10) {
  kfi(X, "text 13 6 playloop"); /* 0 */
  kfi(X, "text 12 29 Play a sound in a table once or continuously"); /* 1 */
  object("inlet", 37, 90); /* 2 */
  kfi(X, "text 12 42 Creation args: \$1 - name of array containing sound"); /* 3 */
  object("inlet", 185, 114); /* 4 */
  object("tabplay~ \$1", 92, 186); /* 5 */
  object("select 1", 113, 157); /* 6 */
  f = "f";
  object(f, 170, 157); /* 7 */
  object("outlet~", 92, 211); /* 8 */
  kfi(X, "obj 94 157 bng 15 250 50 0 empty empty empty 17 7 0 10 -262144 -1 -1"); /* 9 */
  kfi(X, "text 13 71 Bang 1st inlet to play once"); /* 10 */
  kfi(X, "text 124 97 Toggle looping with 2nd inlet"); /* 11 */
  kfi(X, "text 168 185 End-of-sound bang optionally restarts"); /* 12 */
  kfi(X, "text 23 241 thunderbolt & sung"); /* 13 */
  object("select 1", 95, 136); /* 14 */
  connectS(2, 0, 5, 0);
  connectS(4, 0, 7, 1);
  connectS(4, 0, 14, 0);
  connectS(5, 0, 8, 0);
  connectS(5, 1, 7, 0);
  connectS(6, 0, 5, 0);
  connectS(7, 0, 6, 0);
  connectS(9, 0, 5, 0);
  connectS(14, 0, 9, 0);
}
```

Welcome to the secret appendix u kfi! Here you shall find rules to hack rocky2 such that you can gain full functional control over pd -- that’s right! Bangs, toggles, sliders, and more!

What’s that you say? Arrays?

Yes, those bad boys too.

So how does one use these further functionalities previously thought unreachable by rocky2’s powerful compiler?