Project Report of W4115: Programming Languages and Translators

by: Benjamin Kornacki (blk2129)
Thomas Rantasa (tr2286)
Chengchen Sun (cs2890)

1. Introduction

This is the final project report of Language: MoveIt, which can be used to implement 2D image movement easily yet giving stunning effects. This language supports drawing rectangles, ellipses, and lines on an empty graph and easily implementable animation features that can be combined with arithmetic operations and flow-control methods to create relatively powerful animation and physical simulations. MoveIt also implements arrays allowing for mass-processing of objects with similar purposes and operational ease.

2. Language Tutorial

Our language can be downloaded directly via google code.

2.1

Download source tar.gz file or use svn to check it out directly from: ***http***://plt-2d-compiler.googlecode.com/svn/trunk/
Compiling the source files requires a full OCaml enviroment with Thread, Graphics, Unix and String libraries on top of the basic installation. Our compiler has been fully tested and debugged on Mac OS X and Ubuntu 11.10 32bit.

2.2

We present a simple tbc file below. This is a simple hello world program called hello.tbc, which prints the String “hello, world!” onto the OCaml graphics screen at position (200, 200).

```
1 run()
2 {
3         string s;
4         s = S:{200 200 “hello, world!”};
```

```
5         print(s);
6         halt(5000);
7 }
```

The run() is the function that will be first called when program is executing (similar to main() in C). The Compiler will look for the run() function to start and begin execution in it's body. Line 3 declares a string identified by "s" and line 4 assigns a value to it. Line 5 calls the built-in print to display the string on the graph. Line 6 suspends the graph for 5000 milliseconds so that the graph remains visible to the eye for the designated period.

2.3

Write ./tbc < hello.tbc in the command line to run the program. This will cause a graph to appear, that contains the single string "hello, world!".

2.4

If you want to see the generated bytecode, simply add the option -b to the compilation command i.e. ./tbc -b < hello.tbc. Below is the resulting bytecode for the hello world program.

```
0 slots to store global variables
0 Sgraph
1 Jsr 3
2 Hlt
3 Ent 30
4 Litst200 200 "hello world!"
5 Sfp 30
6 Drp
7 Lfp 30
8 Jsr -1
9 Drp
10 Litin 5000
11 Jsr -2
12 Drp
13 Litin 0
14 Rts 0
```

These instructions will be explained in the "Architectural Design" part.

2.5

This completes a very first look at coding in MoveIt. The source package for MoveIt contains more complex code samples and demonstrations which can be run using the following commands.
./tbc < test/demo-array-bubble-sort.tbc
./tbc < test/demo-bounce.tbc
./tbc < test/demo-hello.tbc

3. Language Reference Manual

3.1: Lexical Conventions

3.1.1: Tokens

There are five types of tokens: identifiers, keywords, constants, operators, and other separators. Blanks, horizontal and vertical tabs, newlines and comments are ignored except when they're used to separate tokens. Generally, whitespace will separate two tokens.

3.1.2: Comments

The symbol (: starts a comment, and :) will terminate it. Comments do not nest.

3.1.3: Identifiers

An identifier is a sequence of letters and digits, and the underscore _. The first character of an identifier must be a letter. Identifiers can represent any variables defined in our language.

Identifiers must be global unique meaning that you cannot have a rectangle named a and an array named a at the same time.

3.1.4: Keywords

The following words are reserved as keywords and thus cannot be used as identifiers for the reasons mentioned below:
`int, string, rectangle, ellipse, line, shape.` used to declare specific variables
`for, if, else, while, return:` Used to control the flow of program.
`v1, v2, v3, v4:` Used to get the value in shape elements.

3.1.5: Constants:

There are three kinds of constant: integer-constant, string constant, and shape constant

3.1.5.1: Integer constant

An integer constant is a sequence of decimal digits 0 - 9. For example, 3042, 0097 are valid integer constants while 0x64, 0d55, 7f32 are not valid. Floating point numbers are not supported by our language.
The negative symbol (-) can be added before an integer to represent this is a negative number.
Integer constant can be assigned to an integer variable, or elements in an integer array.

3.1.5.2: String constant

A String constant has the form of S:{<x_val> <y_val> <string>}. Values are separated by space. <x_val> is the x coordinate of the starting point i.e. bottom left point of this string. <y_val> is the y coordinate of the starting point. <string> is a plain string enclosed with two quotation symbols “”. The capital S, the colon, and the two braces are compulsory to represent a string variable.
String constant cannot be assigned to an array. MoveIt does not support arrays of string constant.

3.1.5.3: Shape constant

Shape constants can be of one of three types: a rectangle, an ellipse, and a line. There is a general type called shape. This is used when a function can take any type of shape regardless of whether it’s a rectangle, an ellipse, or a line.
Shape constant must be defined like R:{<v1> <v2> <v3> <v4>}, E:{<v1> <v2> <v3> <v4>}, L:{<v1> <v2> <v3> <v4>}, S:{<v1> <v2> <v3> <v4>}. For rectangles, v1 and v2 represent the coordinate of bottom left point of the rectangle, v3 and v4 are the height and width of the rectangle respectively. For ellipses, v1 and v2 represent the center and v3, v4 represent the x-radius and y-radius. For lines, v1 and v2 represent the starting point of the line, v3 and v4 represent the coordinate of the end point. A shape it can be casted to any of the above.
One key feature for shape is that v1 to v4 can be expressions. for example, R:{i+1 j+1 3*k 4*l} is valid if i, j, k, l are valid integers.
Shape constant can be assigned to a shape variable, or an element inside a shape array.

3.2 Variables

A variable is one specific memory area of which the value can be modified and accessed through the program. You can perform read and write operations on them, and pass them to function calls.

3.2.1: Single variable

Variables can be defined by type. We have these types in our language:
int : declares this is an integer variable.
string: declares this is a string variable.
rectangle: declares this is a rectangle variable.
ellipse: declares this is an ellipse variable.
line: declares this is a line variable.
shape: declares this is a shape variable.
The definition syntax is to use the keyword followed by a valid identifier. For example, int abc9 is a valid variable of int, but string int is not a valid identifier, as int is not a valid identifier name.

3.2.2: Arrays

Arrays are defined a little differently by putting array size within brackets following the name: int ab[10] declares an integer array with 10 integers inside it, rectangle r[30] defines a rectangle array with 30 rectangles inside it and so on. To get a specific element within the array, simply put the index within the bracket. For example, ab[8] will reference the 9th element in the array. Like C, the index starts from 0 and ends at size-1.

3.2.3: Variable declaration

For simplicity, in MoveIt, we force the declaration of all variables to be at the first place of a scope i.e. at beginning of the file or at beginning of a function.
The declaration and initialization must be divided onto two lines, meaning that an operation like:
`rectangle r = R:{10 20 30 40};` will not be accepted.

3.3: Scopes

In MoveIt we have a global scope and a local scope. All variables declared in global scope can be referenced within local scope, but not vice versa. Functions cannot reference other functions' local scope variables.
The static scoping method is adopted in MoveIt. For example, the following code:

```
 1 int a;
 2 assign()
 3 {
 4     a = 4;
 5 }
 6 run()
 7 {
 8     int a;
 9     a = 44;
10     assign(a);
11     print_int(a);
12 }
```

Will print out 44, not 4 as that would imply dynamic scoping.

3.4: Functions

3.4.1: Function definitions

A function is defined with a valid identifier as its unique name, followed by a pair of parenthesis, which may have parameters inside it or not. Parameters must be single variables and declared with type. The function body is enclosed with a pair of braces {} which may have or have not return a value. If there's no return value, a default integer 0 will be returned.
A function is not required to be defined with a type.
For example, the following are valid function definitions:

```
draw_a_bar(int start, int end)
{
        line l;
        l = L:{start end start + 100 end + 200};
        print(l);
        halt(5000);
}

give_a()
{
        return 9;
}
```

3.4.2: Built-in functions

In MoveIt, the graph is drawn by calling built-in functions. We provide two built-in functions, named print() and halt() to construct the graph, and another two: print_int() and dumpstack() for debug use. Although they're debug functions, you may call them in your program if you

wish.
The definition of each built-in functions are:
`print(<var>): <var>` is a string or a single shape variable, with shape type correctly set i.e. a rectangle, an ellipse, or a line. when calling print(), it will print the string/shape according to its parameters. On error, it will print an exception to the terminal and close the program.
`halt(<var>): <var>` is an integer. This function suspends the graph for <var> milliseconds and will clear all the graph after that.
`print_int(<var>):` This function will take an integer and print it to the terminal screen.
`dumpstack():` This function will dump current stack to the terminal screen.

3.4.3: Return values
The function's return value is not defined nor checked in MoveIt. As a result, if you return a rectangle type to a variable of int type, the type check will fail. Also, only one single value will be returned, arrays are not supported.

3.5: Operations

3.5.1: Arithmetic/Logical operations
In MoveIt, arithmetic and logical operations are permitted on integers. We support the following arithmetic operations:
+ -> add two integers
- -> subtract two integers
* -> multiply two integers
/ -> first divided by second
== -> is the two integers equal?
!= -> is the two integers not equal?
>= -> is first larger or equal second?
<= -> is first smaller or equal second?
> -> is first larger than second?
< -> is first smaller than second?
Arithmetic operations, are left associate and * / will have higher priority over + -.

3.5.2: Movement operations
The basic operations on shape and string are movement. MoveIt has two definitions of movement, namely MoveTo and MoveBy.
The syntax of MoveTo is the => operator on a shape or string. For example, to move a rectangle's starting point to {100 100}, use:
```
r => {100 100}
```
For MoveBy, use ->. This will make the shape/string's starting point move by a vector. For example, to move an ellipse e to 50 pixels down and 50 pixels left, use:
```
e -> {-50 -50}
```
In short, Movement operators directly manipulate the coordinates of shapes/strings.

3.6: Array and Shape References
It's easy to load and store value into arrays. For example:
a[2] = 4 will assign value 4 to the integer array a's third element;
a[9] = a[7] will assign eighth element's value to tenth's of array a.
For shape, use the system reserved tokens v1 - v4 to access respective value. For example:
```
rectangle r;
r = R:{10 20 30 40};
print_int(r.v1);
print_int(r.v2);
print_int(r.v3);
print_int(r.v4);
```
will print out 10 20 30 40.

3.7: Expressions
Expressions are basic operations terminated by semi-colon. Expressions include:
a. constants
b. variable names
c. binary arithmetic/logical operations

d. assignments
e. function calls
f. movements
g. get index values
h. another expressions enclosed by two parenthesis.
All these must be handled properly and will be discussed in the architectural design part.

3.8: Statement:

3.8.1: The flow control

Statements are groups of expressions. Usually statements are divided into blocks by using flow-control keywords. MoveIt supports if, else, while, return, and for.

3.8.1.1: if / else

If is used to switch execution of either one blocks based on the result of an expression, or just determine if one block should be executed or not. The syntax of if is:
`if ( expression ) { block_1 } else { block_2}`, or
`if ( expression ) { block }`
The else is not mandatory. if expression is true, in the first case it will do block_1, in the second case it will do block; if it is false, in the first case it will do block_2, in the second case it will do nothing. Expression will be evaluated before blocks are ever executed.

3.8.1.2: while

While is used to do a loop based on some criterion. The syntax of while is:
`while ( expression ) { block }`
it will first evaluates the expression to see if it is true. If yes, continue on doing block. After block is done, go back to expression again and evaluate if it is true. Once the expression is evaluated to false, it will skip the block and jump out of while, continuing to execute on the next expression after while.

3.8.1.3: for

the for is a more complex version of while: it has three expressions and one statement block:
`for ( expr1; expr2; expr3 ) { block }`
`for` will evaluate expr1, then judge if expr2 is true. If yes, go on executing block. After block is finished, execute expr3, and go back to judge if expr2 is true. Once expr2 is judged as false, it will break `for` and continue on doing the statements after the block.

3.8.1.4: return

return will return a specific value to the place where this function is called. The syntax is simply
`return <var>`
Where var can be a variable, a constant, or an expression. Be careful that return cannot return the function itself. For example, in the factorial function, f(n) needs to call n*f(n-1). In this case, return cannot be applied on n*f(n-1).

3.8.2: The first run

The program will starts from a function called run(). If there's no such function, compiler will complain and terminate.

4. Project Plan

Process:

Our project was very much a team effort. Planning was the most important part of our process. We tried to constantly set deadlines for ourselves. This started at the very beginning of the semester when our group first got together. We immediately distributed responsibilities and set deadlines for ourselves.

Specification occurred mostly at the beginning of the term. However, as we were working, if we thought of a feature that we would like to add or remove, we would immediately halt the production and work to redefine the specification and change anything that needed to be changed.

When it became time to start writing code we did almost all of it together. We did not necessarily all work on the same code at the same time, but we were always in the same

room together. This allowed for constant communication in case anyone had any questions or suggestions for how we can make the project better. Testing was a constant process during development. Whenever anything was implemented, we would immediately test it to ensure it works exactly as we want it to.

Style Guide:

While writing the code, we constantly emphasized documentation. Anything that anyone wrote must be documented in some way so that the other members of the group could understand what one member had written if he wasn't around to explain it.

Other then that, we did not specify any particular style to use for the project. Since we were with each other while we were wrote the majority of the code, there was no need to have a specified style guide, since we were in constant communication. That is any question on how to write or format a chunk of code we be answered immediately by another member of the group. The only thing we stressed was clarity. We allowed ourselves to write the code however we liked, it just had to be clear.

Timeline:

September 20: First Group Meeting/ Project Idea

September 28: Proposal

October 15: Implement AST, Scanner, and Parser

October 22: Implement Interpreter and hello_world.tbc

October 31: Language Reference Manual/have current code completely debugged and running as desired

November 15: Have full implementation of interpreter/begin working on compiler and bytcode interpreter

December 1: Run hello-world.tbc using bytecode interpreter and compiler

December 15: Have full implementation of entire project

December 17: finsih debugging/ write demo's for presentation

December 20: Present to Professor Edwards

December 21: Implement arrays

December 22: Final Report

Roles and Responsibilities:

Most of the project was done as a team. We would work sitting next to each other so that we could easily collaborate with each other. We would each focus on individual aspects of the project however, but no one was the sole contributor to any part of the project.

In terms of what we individually did, Benjamin Kornacki worked a lot on testing and writing the demonstrations that would show off the compiler. Chengchen Sun worked primarily on the compiler (i.e. the compile.ml and and execute.ml). Thomas Rantasa worked primarily on the bytecode interpreter (i.e. bytecode.ml). There was an equal contribution on most of the other files (ast.ml, scanner.ml, etc). The interpreter (i.e. interpreter.ml) was implemented mostly by Ben and Chen, but was not used in the final version of the project.

Software Environment:

For the project we used the svn development tool from google code. All of our file sharing including code updates, progress reports, language reference manual as well as the final report were comitted and editted using google code and google docs.

The code is written entirely in O'Caml. Since we use the O'Caml graphics library, there was no need to translate the code to another language.

Project Log:

Rev	Commit log message	Date
r187	Removed unused Prt and Prtint	Today (4 hours ago)
r186	3 more testcase	Today (6 hours ago)
r185	bubble sort using arrays	Today (7 hours ago)
r184	previous testcase has a typo and caused interesting problem	Today (9 hours ago)
r183	shape arrays work perfectly	Yesterday (24 hours ago)
r182	Full functional array on int operation	Yesterday (25 hours ago)
r181	Int array works. Bug is array must be initialized and a[0] must be written.	Yesterday (26 hours ago)
r180	Edited wiki page DefineArray through web user interface.	Yesterday (30 hours ago)
r179	update scanner	Dec 20 (2 days ago)
r178	Created wiki page through web user interface.	Dec 20 (2 days ago)
r177	No run(), not no main()	Dec 20 (2 days ago)
r176	Deleted microc folder	Dec 18 (4 days ago)
r175	clean execute and test case	Dec 18 (4 days ago)
r174	execute works on strings demo-hello updated to new functions random test files	Dec 18 (4 days ago)
r173	bubble sort	Dec 18 (4 days ago)
r172	Make code cleaner	Dec 17 (5 days ago)
r171	Added three testcases for return	Dec 17 (5 days ago)
r170	Updated a tiny testcase	Dec 17 (5 days ago)
r169	Fixed a bug in returning rectangle, ellipse and line	Dec 17 (5 days ago)
r168	Minor changes on test	Dec 17 (5 days ago)
r167	Several test cases	Dec 17 (5 days ago)
r166	Deleted print out AST in tbc.ml	Dec 17 (5 days ago)
r165	Minor changes on code style	Dec 17 (5 days ago)
r164	Minor adjust of bytecode output format	Dec 17 (5 days ago)
r163	Edited wiki page LanguageGrammar through web user interface.	Dec 17 (5 days ago)

Rev	Commit log message	Date
r162	Edited wiki page LanguageGrammar through web user interface.	Dec 17 (5 days ago)
r161	Remove Bind and thus suppress	Dec 17 (5 days ago)
r160	Move test codes under test	Dec 16 (5 days ago)

	folder	
r159	Latest commit	Dec 16 (5 days ago)
r158	Latest scanner.mll	Dec 16 (5 days ago)
r157	beq/bne	Dec 16 (5 days ago)
r156	hopefully fixed bin op	Dec 16 (6 days ago)
r155	fixed binop	Dec 16 (6 days ago)
r154	implement moves in execute and bytecode	Dec 16 (6 days ago)
r153	please work now	Dec 16 (6 days ago)
r152	updated execute	Dec 16 (6 days ago)
r151	demmo hello	Dec 16 (6 days ago)
r150	demmo bounce	Dec 16 (6 days ago)
r149	fixed stack bug	Dec 16 (6 days ago)
r148	modified syntax error	Dec 16 (6 days ago)
r147	updated print	Dec 16 (6 days ago)
r146	lfp	Dec 16 (6 days ago)
r145	implemented sfp	Dec 16 (6 days ago)
r144	very minor update	Dec 16 (6 days ago)
r143	new stack size	Dec 16 (6 days ago)
r142	dump stack implemented	Dec 16 (6 days ago)
r141	debug debug	Dec 16 (6 days ago)
r140	debug print int added to jsr	Dec 16 (6 days ago)
r139	jsr-2	Dec 16 (6 days ago)
r138	moved jsr	Dec 16 (6 days ago)

Rev	Commit log message	Date
r137	new print	Dec 16 (6 days ago)
r136	Add support for GetV1	Dec 16 (6 days ago)
r135	Finished execute	Dec 16 (6 days ago)
r134	Tbc for debug	Dec 16 (6 days ago)
r133	Edited wiki page BytecodeGrammar through web user interface.	Dec 16 (6 days ago)
r132	Add code to avoid the three unmatched warnings	Dec 16 (6 days ago)
r131	changes to bytecode and updated scanner	Dec 16 (6 days ago)
r130	Modified one of your typo in Sub -> op1 op2 to Sub -> op1 - op2	Dec 16 (6 days ago)
r129	Adding declaration of Shape, delete dot, float, time, add bind.	Dec 16 (6 days ago)
r128	Adding declaration of Shape and delete Movein function	Dec 16 (6 days ago)
r127	Deleting ref.txt	Dec 16 (6 days ago)
r126	Deleting listtest.ml	Dec 16 (6 days ago)
r125	added negative numbers to	Dec 16 (6 days ago)

	scanner	
r124	bug in execute and bytecode..	Dec 16 (6 days ago)
r123	Add the test folder	Dec 16 (6 days ago)
r122	Full version of tbc	Dec 16 (6 days ago)
r121	fixed bytecode bug	Dec 16 (6 days ago)
r120	Delete listtest.cmo	Dec 16 (6 days ago)
r119	delete listtest.cmi	Dec 16 (6 days ago)
r118	updated bytecode and execute	Dec 16 (6 days ago)
r117	Adding makefile	Dec 16 (6 days ago)
r116	Add compiler.ml	Dec 16 (6 days ago)
r115	started debugging bytecode interpreter (cant figure out this one set of bugs maybe you guys can help)	Dec 15 (6 days ago)
r114	updated scanner parser ast for new commands (in collaboration with chen) current version of bytecode interpreter (not fully finished)	Dec 15, 2011
r113	Created wiki page through web user interface.	Dec 15, 2011

Rev	Commit log message	Date
r112	Edited wiki page BytecodeGrammar through web user interface.	Dec 15, 2011
r111	Edited wiki page BytecodeGrammar through web user interface.	Dec 15, 2011
r110	Edited wiki page BytecodeGrammar through web user interface.	Dec 15, 2011
r109	Edited wiki page BytecodeGrammar through web user interface.	Dec 15, 2011
r108	Edited wiki page BytecodeGrammar through web user interface.	Dec 15, 2011
r107	Edited wiki page RootPage through web user interface.	Dec 15, 2011
r106	Created wiki page through web user interface.	Dec 15, 2011
r105	Special makefile to use for ONLY BYTECODE and EXECUTE	Dec 15, 2011
r104	Sorry, this is the Correct File	Dec 15, 2011
r103	Add Modified makefile to compile bytecode, execute and compile.	Dec 15, 2011
r102	temporary commit until bytecode interpreter is finished	Dec 15, 2011
r101	Created Full Bytecode:	Dec 14, 2011

	Please take a look because I made a few important realizations and put them as comments above the code!!!! Started Bytecode interpreter: more to follow in the next hours	
r100	Start on Compiler	Dec 14, 2011
r99	wed morning2	Dec 14, 2011
r98	wed morning	Dec 14, 2011
r97	end of Tue	Dec 13, 2011
r96	Line	Dec 13, 2011
r95	Weird problem in declaration order	Dec 13, 2011
r94	tbc file	Dec 13, 2011
r93	tbc file	Dec 13, 2011
r92	Lots of work	Dec 13, 2011
r91	Created wiki page through web user interface.	Dec 11, 2011
r90	added rectangle to scanner and started parser work	Nov 30, 2011
r89	Thread works	Nov 30, 2011
r88	Updated commit	Nov 30, 2011

Rev	Commit log message	Date
r87	Using Thread	Nov 28, 2011
r86	Edited wiki page RootPage through web user interface.	Nov 19, 2011
r85	Edited wiki page RootPage through web user interface.	Nov 19, 2011
r84	Edited wiki page RootPage through web user interface.	Nov 19, 2011
r83	Edited wiki page Progress through web user interface.	Nov 19, 2011
r82	helloworld program succeed	Nov 19, 2011
r81	Uploaded complete Scanner	Nov 16, 2011
r80	Edited wiki page Progress through web user interface.	Nov 16, 2011
r79	Edited wiki page Progress through web user interface.	Nov 16, 2011
r78	Edited wiki page Progress through web user interface.	Nov 16, 2011
r77	Edited wiki page RootPage through web user interface.	Nov 16, 2011
r76	Original Version of MicroC	Nov 13, 2011
r75	Edited wiki page Progress through web user interface.	Nov 7, 2011
r74	Edited wiki page RootPage through web user interface.	Nov 7, 2011
r73	Edited wiki page RootPage through web user interface.	Nov 7, 2011
r72	Edited wiki page RootPage	Nov 7, 2011

	through web user interface.	
r71	Edited wiki page RootPage through web user interface.	Nov 7, 2011
r70	Edited wiki page RootPage through web user interface.	Nov 7, 2011
r69	Edited wiki page Progress through web user interface.	Nov 7, 2011
r68	Created wiki page through web user interface.	Nov 7, 2011
r67	Edited wiki page RootPage through web user interface.	Oct 31, 2011
r66	Edited wiki page RootPage through web user interface.	Oct 31, 2011
r65	Edited wiki page RootPage through web user interface.	Oct 31, 2011
r64	Add our names and uni.	Oct 31, 2011
r63	Modified to let identifiers start with '$' for simpler parser.	Oct 31, 2011

Rev	Commit log message	Date
r62	First submitted version to TA on Oct 23. HelloWorld Sample code out.	Oct 23, 2011
r61	Edited wiki page RootPage through web user interface.	Oct 22, 2011
r60	Edited wiki page RootPage through web user interface.	Oct 22, 2011
r59	Edited wiki page RootPage through web user interface.	Oct 22, 2011
r58	\	Oct 22, 2011
r57	Edited wiki page RootPage through web user interface.	Oct 22, 2011
r56	Edited wiki page RootPage through web user interface.	Oct 22, 2011
r55	Edited wiki page RootPage through web user interface.	Oct 22, 2011
r54	Edited wiki page RootPage through web user interface.	Oct 22, 2011
r53	Edited wiki page RootPage through web user interface.	Oct 22, 2011
r52	Edited wiki page RootPage through web user interface.	Oct 22, 2011
r51	Edited wiki page RootPage through web user interface.	Oct 22, 2011
r50	Updated Scanner	Oct 21, 2011
r49	Created Folders and did Scanner work	Oct 21, 2011
r48	Created the project files and started the scanner	Oct 21, 2011
r47	Edited wiki page RootPage through web user interface.	Oct 21, 2011
r46	Edited wiki page RootPage through web user interface.	Oct 21, 2011

r45	Edited wiki page RootPage through web user interface.	Oct 21, 2011
r44	Edited wiki page RootPage through web user interface.	Oct 21, 2011
r43	Edited wiki page RootPage through web user interface.	Oct 21, 2011
r42	Edited wiki page RootPage through web user interface.	Oct 20, 2011
r41	Edited wiki page RootPage through web user interface.	Oct 20, 2011
r40	Edited wiki page RootPage through web user interface.	Oct 20, 2011
r39	Edited wiki page RootPage through web user interface.	Oct 20, 2011
r38	Edited wiki page RootPage through web user interface.	Oct 20, 2011

Rev	Commit log message	Date
r37	Edited wiki page RootPage through web user interface.	Oct 20, 2011
r36	Edited wiki page RootPage through web user interface.	Oct 20, 2011
r35	[No log message]	Oct 20, 2011
r34	Modification 3	Oct 20, 2011
r33	Try to update folders	Oct 20, 2011
r32	Delete README.txt	Oct 20, 2011
r31	hello	Oct 20, 2011
r30	[No log message]	Oct 20, 2011
r29	Edited wiki page RootPage through web user interface.	Oct 20, 2011
r28	Edited wiki page RootPage through web user interface.	Oct 19, 2011
r27	Edited wiki page RootPage through web user interface.	Oct 19, 2011
r26	Edited wiki page RootPage through web user interface.	Oct 19, 2011
r25	Edited wiki page RootPage through web user interface.	Oct 19, 2011
r24	Edited wiki page RootPage through web user interface.	Oct 19, 2011
r23	Edited wiki page RootPage through web user interface.	Oct 19, 2011
r22	Edited wiki page RootPage through web user interface.	Oct 19, 2011
r21	Edited wiki page RootPage through web user interface.	Oct 19, 2011
r20	Edited wiki page RootPage through web user interface.	Oct 19, 2011
r19	Edited wiki page RootPage through web user interface.	Oct 19, 2011
r18	Edited wiki page RootPage through web user interface.	Oct 19, 2011

r17	Edited wiki page RootPage through web user interface.	Oct 19, 2011
r16	Edited wiki page RootPage through web user interface.	Oct 19, 2011
r15	Edited wiki page RootPage through web user interface.	Oct 18, 2011
r14	Edited wiki page RootPage through web user interface.	Oct 18, 2011
r13	Edited wiki page RootPage through web user interface.	Oct 18, 2011

Rev	Commit log message	Date
r12	Edited wiki page RootPage through web user interface.	Oct 18, 2011
r11	Edited wiki page RootPage through web user interface.	Oct 18, 2011
r10	Edited wiki page RootPage through web user interface.	Oct 18, 2011
r9	Edited wiki page RootPage through web user interface.	Oct 18, 2011
r8	Edited wiki page RootPage through web user interface.	Oct 18, 2011
r7	Edited wiki page RootPage through web user interface.	Oct 18, 2011
r6	Edited wiki page RootPage through web user interface.	Oct 18, 2011
r5	Edited wiki page RootPage through web user interface.	Oct 18, 2011
r4	Edited wiki page RootPage through web user interface.	Oct 18, 2011
r3	Edited wiki page RootPage through web user interface.	Oct 18, 2011
r2	Created wiki page through web user interface.	Oct 18, 2011
r1	Initial directory structure.	Oct 16, 2011

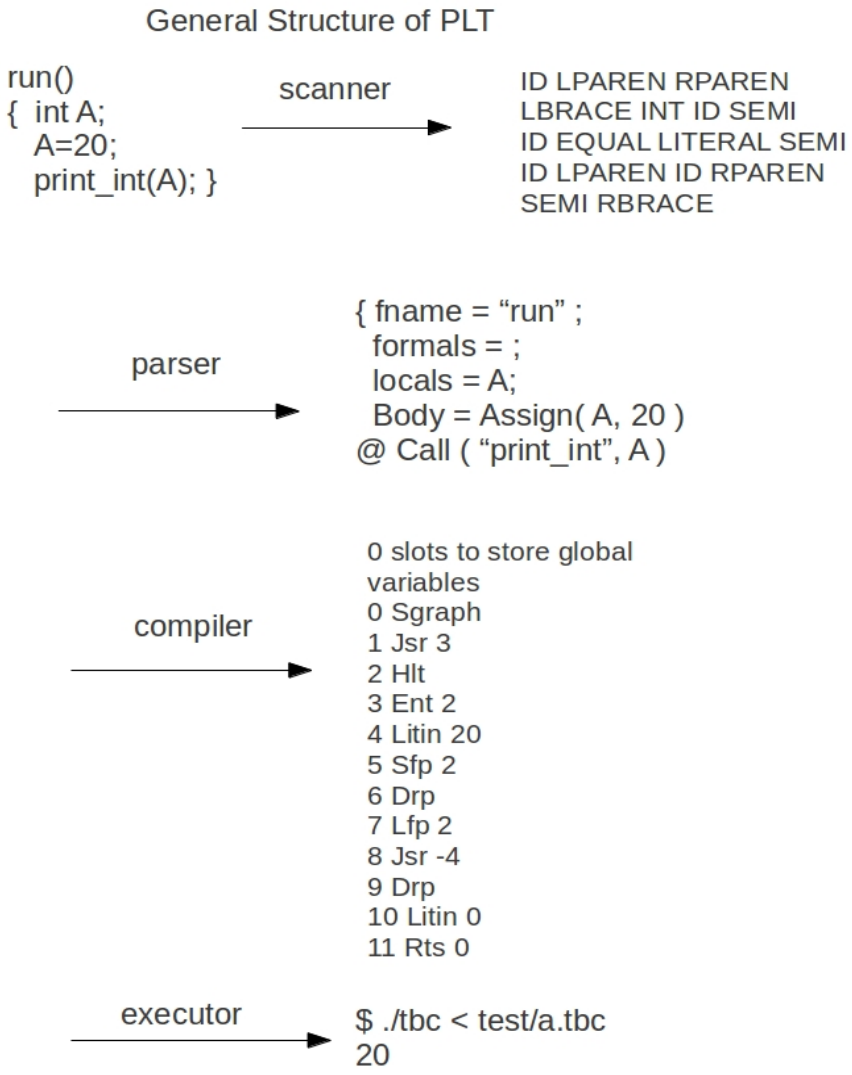
5. Architectural Design

5.1: Ast/Parser/Scanner (Ben, Thomas, Chen)

5.1.1: General Idea

The Parser loads the tbc source file and generates the abstract syntax tree (AST) for the program. This is done in the following manner:

a. the rule token is called in scanner to convert source code into discrete tokens.
b. parser calls the program routine to parse the token sequence into two large lists: one is for variable declaration, one is for function declaration. The variable declaration list is for global variables, and the function declaration is for all functions defined. Since our implementation only has global variable declarations outside functions, this suffices.
The block diagram is on the next page.



5.1.2: Abstract Syntax Tree (ast.ml)

The Abstract Syntax Tree consists of three parts:

a. define tag-union of binary operands and shapes

This defines the valid binary operations and shape types.

b. define how different types of variables are represented

This defines how our compiler defines variable metadata internally. We treat a variable declaration and variable value separately, since when a variable is declared, all we can know is its name and type.

A variable declaration structure is like:

```
type vari_decl = { vtype: string; vsize: int; name: string; }
```

The vtype means the type of this variable. It can be:
'int', 'string', 'rect', 'ellipse', 'line', 'shape',
'arrayi', 'arrayr', 'arraye', 'arrayl', 'arrays'.
The vsize represents the size of this variable. Except for arrays, which is the number of elements in arrays, all other variables' vsize should be 1.
The name is straightforward: this is the identification of this variable.
A variable value structure is like:

```
type vari_value = { ltype : string; value: string; }
```

The ltype is simply the same as vtype, representing the data type of this variable.
The value is actual value of this variable, stored as string for further processing. For example, if this is a rectangle, the value would be a string like "10 20 30 40".
A function declaration structure is like:

```
type func_decl = { fname : string; formals: (vari_decl) list; locals:
(vari_decl) list; body: stmt list; }
```

It takes the

c. define expression and statement 'types' i.e. what can be substituted by expr / stmt

This defines what an expression / statement can be reduced to, and by how. Details:
Expressions:
Literal of int: This gets a sequence of digits put decimal number on top of stack
String of string: This gets a string and returns each character as its ASCII number, plus the metadata to identify that this area is a string, on top of the stack;
Rectangle of string: converts string to rectangle structure
Ellipse of string: Identical to Rectangle
Line of string: Identical to Rectangle
Shape of string: Identical to Rectangle
Id of string: perform Id operation
Binop of expr * op * expr
Assign of string * expr
Call of string * expr list
Moveto of string * expr * expr : Move to a specific coordinate
Moveby of string * expr * expr : Move by a specific vector
Noexpr
GetV1 of string : Get the first value of a shape
GetV2 of string : Get the second value of a shape
GetV3 of string : Get the second value of a shape
GetV4 of string : Get the second value of a shape
Aid of string * expr : Array id
AAssign of string * expr * expr : Assign value to array element
REvaluation of expr * expr * expr * expr : Make Rectangle definition accept expressions, not only numbers.
EEvaluation of expr * expr * expr * expr
LEvaluation of expr * expr * expr * expr

5.2: Compiler (Chen) compile.ml

The compiler's work is to
a. convert methods used by ast into bytecode
b. allocate global and local variables' index using enum and string_map_pairs
c. initialize an empty graph, find run() function and concatenate functions.

5.2.1 Bytecode used by compiler:

Litin of int: push an int literal on top of stack.
Litst of string: push string on top of stack.
Litsh of string: parse the string to form shapes' parameter and store these parameters on top of stack.
Drp: drop the value on top of the stack
Bin of Ast.op: get two values on top of stack and apply them to the operation defined as Ast.op
Lod of int: load a global variable by index to top of stack.
Str of int: store whatever on top of stack to the global variable area referenced by index
Lfp of int: load a local variable by index to top of stack
Sfp of int: store whatever on top of stack to local variable referenced by index

Jsr of int: Jump to a specific offset
Ent of int: Allocate local variable space
Rts of int: Clear all local variables, set sp and fp pointer back, to return to a previous location.
Beq, Bne, Bra of int: flow control bytecode which is identical to those in MicroC
Loda of int: Load from global array variables with base address as int. Fetch the integer currently on top of stack as element offset, and store the value back to top of stack.
For example, there's a rectangle array r[10] as a global variable, and you want to load r[5]. Suppose r[0] is laid on global offset 2 (this is the case if an int variable is defined before this rectangle array), so the bytecode to fetch r[5] is

```
Litin 5; Lfpa 2;
```

The first is to store 5 on top of stack, the second will load from it, and calculate the specific location of r[5]. Finally, r[5] will be put back on top of stack.
Stra of int: identical to Loda, except the value it will store is placed starting from stack.(sp-3), because stack.(sp-1) and stack.(sp-2) are the element offset value.
Lfpa of int: identical to Loda;
Sfpa of int: -identical to Stra;
Hlt: terminate the program;
Sgraph: start to paint a new empty graph
Egraph: erase the whole graph
Susp: Load the int on top of stack and suspend for that number of milliseconds.
Movby: move object by vector;
Movto: move object by absolute location
GetC: fetch elements' value inside a shape.
MakeS: Grab the top five integers on stack and place back a shape make by these integers.
Detailed implementation of how to convert each Ast method to Bytecodes can be found in our attached compile.ml file.

5.2.2 Allocate variables' space

The compiler first calculates the offset of local and global variables and store them in a StringMap called global_indexes and local_indexes. These indexes are assigned to environmental variable env and are used in executing bytecode numbers like Lod/Str, Lfp/Sfp.

5.2.3 Initialize empty graph, find run(), make functions work

compiler will add Sgraph at thte beginning of the program, and search for run() function and place it at the beginning of the bytecode sequence. For other functions, compiler will map all non-built-in function to func_offset.(i) and use Jsr to jump to their entry point. At each entry point, compiler also adds a Ent command to allocate local variable space.

5.3: Bytecode/Bytecode Interpreter (Thomas) bytecode.ml, execute.ml

The Bytecode used by the interpreter is, of course, identical to the bytecode that the compiler uses. The interpreter will read the bytecode instructions and interpret them for the OCaml execution environment. Furthermore, the interpreter handles all stack and variable operations, maintaining control of the stack-pointer, frame-pointer and program-counter. The general structure of these was derived form the MicroC language, but modified for the graphics purposes of MoveIt. Since MoveIt operates on logically complex objects as opposed to only ints, the stack operations are significantly more involved. All stack data is identified by an integer-ID pushed on top of the actual object data. Based on this ID the interpreter is able to process the following data correctly.

5.4: Interpreter (Ben) interpret.ml

Early on in the coding stages the interpreter was used to test various implementation methods and to give us general feedback about the output our language created without having to write the full compiler and bytecode interpreter.

6. Test Plan

Example programs:

This first program shows how to write a simple hello world function that uses the power of our language. That is, instead of simply printing the string "hello" to the screen, this program using various shapes draws out the letter "h e l l o". This code can be found in appendix 1.

As stated above, the compiler is capable of implementing real world movement. To

demonstrate this, this next program shows a ball falling under the of some gravity and having it bounce when it hits the bottom. This code can be found in appendix 2.

MoveIt also has the power to implement more complex algorithms such as bubble sort. This last program shows 8 rectangles of various sizes being sorted based on there height. This program takes advantage of all this language has to offer. It uses arrays, shapes movement, and functions to visually show the sorting of a list using bubble sort. Before, we implemented arrays, this code was 1700 lines long, however, after we implemented arrays, the code is now only 130 lines long. The code can be found in appendix 3.

Test Suites:

All our test programs can be found in Appendix 4. These test cases were chosen based on the desired functionality of our compiler. We wanted some of the basic functionality as C. That is, the basic arithmetic, loops (i.e. while and for), if statements, and functions. There is also functionality that we needed to test that is specific to MoveIt. For example, initializing shapes, printing shapes, returning and passing shapes as arguments to function, and moving shapes.

The purpose of our testing was to ensure that not only our compiler would generate the correct output, but also that the compiler allocates and manipulates memory on the stack correctly. To test this we implemented two functions, dumpstack() which prints the entire stack to standard out and print_int(int i) which prints the integer i to standard out.

There was no automation used in testing. Since our compiler deals with movement of shapes it would have been very difficult to create automated test cases. For example, if we were to have a circle move from point a to point b in 5 seconds there will not be a single output but potentially thousands of outputs, (one for every movement of the object). This added level of complexity made it difficult to create an automated test case.

Who Did What:

The majority of this project was done as a team. That is, we we worked individually on various parts of the project but we always worked in the same room. This allowed constant communication and brainstorming. If one person thought of a way to make the project better, then we would immediately discuss and decide what changes need to be made. Also if anyone of us got stuck or couldn't figure out how to implement something, as a team we could help that individual out. So even though we were working individually on various sections, we were constantly collaborating so no one person had the sole input on any given section. This also allowed for much faster implementation time.

In terms of what we individually did, Benjamin Kornacki worked a lot on testing and writing the demonstrations that would show off the compiler. Chengchen Sun worked primarily on the compiler (i.e. the compile.ml). Thomas Rantasa worked primarily on the bytecode and bytecode interpreter (i.e. bytecode.ml and execute.ml). There was an equal contribution on most of the other files (ast.ml, scanner.ml, etc). The interpreter (i.e. interpreter.ml) was implemented mostly by Ben and Chen, but was not used in the final version of the project.

Again, saying that someone implemented some part of the project only means that they did the grunt work for that section. Every member of the group made a significant contribution to every aspect of this project. This includes testing. Even though Ben was in charge of testing, we all tested our code as we were writing it.

7. Lessons Learned

Chen:

I think the most important thing I learned from the project is how to team with other cool guys, especially in understanding their ideas, trying to explain to them my ideas and persuade them, discuss about the right implementation. Meanwhile, in team work I must be responsible for my work, always trying to make them perfect because my careless can make others wasting much time in finding MY errors, and always trying to help others. Besides, another important skill is to think, talk quickly in geek English. This is critical for my future work.
For techniques, learning how to manage work efficiently using svn, really powerful tool.

Ben:

This was the first major project that I worked on with a group so there were a lot of

things that I have learned through out the semester. The most important was the willingness to be open to new ideas and to not be afraid to to voice your opinion when you believe that you have an idea that will be beneficial. I believe that the key to success is a group that works well together. This means that everyone does what they are supposed to do, and that people are respective of each other.

From the technical side, before this project I had no experience using svn. Having completed the project I would highly recommended this to anyone working on a large project with other people. Without svn, the compiler would have been vary difficult to implement and there would have been many more mistakes throughout the code.

The biggest piece of advice that I can give to future groups is to start early and to write test cases as you go along. Starting early allowed to include new ideas that we did not think of in our initial conception of the project. If we did not start early we would not have had the time to think through the new ideas that we ended up implementing. Also writing test cases was key to the success of our project. These test cases helped us find bugs that were hidden in the code. That is, the code would compile but since we knew the expected output of each test case we were able to know whether or not our code was running as we had intended.

Thomas:

The most important lesson I learned from this was planning. On smaller projects and assignments, “coding by doing” is often a very appealing planning strategy. A project of this scale, implemented collaboratively between 3 people did not allow for such freedom. The parts of the project that we planned effectively were implemented twice as quickly on average, simply because we knew exactly what we could expect from our teammates. At the same time, I learned to have faith in my team. This experience showed me that teamwork can be a very effective way to complete larger projects as long as members are willing and able (which was luckily the case here).

Lastly, working on MoveIt showed me the importance of deadlines. When working alone, it is easy to forgive oneself for procrastination, but having others rely on me completing my work on time made me realize that deadlines exist for a reason and need to be kept to.

Appendix 1:

Here is the code for hello.tbc

```
rectangle h1;
rectangle h2;
rectangle h3;
rectangle h4;
rectangle h5;
rectangle h6;
rectangle h7;
rectangle h8;
rectangle h9;
rectangle h10;

string e1;
string e2;
string e3;
string e4;
string e5;
string e6;
string e7;
string e8;
string e9;
string e10;
string e11;
```

```
        string e12;
        string e13;
        string e14;
        string e15;
        string e16;
        string e17;
        string e18;
        string e19;
        string e20;

        line l1;
        line l2;
        line l3;
        line l4;
        line l5;
        line l6;
        line l7;
        line l8;
        line l9;
        line l10;
        line l11;
        line l12;
        line l13;
        line l14;
        line l15;
        line l16;

        ellipse o1;
        ellipse o2;
        ellipse o3;
        ellipse o4;
        ellipse o5;
        ellipse o6;
        initialize_h(){
                h1 = R:{ 0 0 20 20 };
                h2 = R:{ 0 0 20 20 };
                h3 = R:{ 0 0 20 20 };
                h4 = R:{ 0 0 20 20 };
                h5 = R:{ 0 0 20 20 };
                h6 = R:{ 0 0 20 20 };
                h7 = R:{ 0 0 20 20 };
                h8 = R:{ 0 0 20 20 };
                h9 = R:{ 0 0 20 20 };
                h10 = R:{ 0 0 20 20 };
        }

        initialize_e(){
                e1 = S:{ 0 0 "e" };
                e2 = S:{ 0 0 "e" };
                e3 = S:{ 0 0 "e" };
                e4 = S:{ 0 0 "e" };
                e5 = S:{ 0 0 "e" };
                e6 = S:{ 0 0 "e" };
                e7 = S:{ 0 0 "e" };
                e8 = S:{ 0 0 "e" };
                e9 = S:{ 0 0 "e" };
                e10 = S:{ 0 0 "e" };
                e11 = S:{ 0 0 "e" };
                e12 = S:{ 0 0 "e" };
                e13 = S:{ 0 0 "e" };
                e14 = S:{ 0 0 "e" };
                e15 = S:{ 0 0 "e" };
```

```
        e16 = S:{ 0 0 "e" };
        e17 = S:{ 0 0 "e" };
        e18 = S:{ 0 0 "e" };
        e19 = S:{ 0 0 "e" };
        e20 = S:{ 0 0 "e" };
}

initialize_l(){
        l1 = L:{ 0 0 0 120 };
        l2 = L:{ 0 0 0 120 };
        l3 = L:{ 0 0 0 120 };
        l4 = L:{ 0 0 0 120 };
        l5 = L:{ 0 0 0 120 };
        l6 = L:{ 0 0 20 0 };
        l7 = L:{ 0 0 20 0 };
        l8 = L:{ 0 0 0 120 };
        l9 = L:{ 0 0 0 120 };
        l10 = L:{ 0 0 20 0 };
        l11 = L:{ 0 0 20 0 };
        l12 = L:{ 0 0 20 0 };
        l13 = L:{ 0 0 20 0 };
        l14 = L:{ 0 0 20 0 };
        l15 = L:{ 0 0 20 0 };
        l16 = L:{ 0 0 20 0 };

}

initialize_o(){
        o1 = E:{ 0 0 20 20 };
        o2 = E:{ 0 0 15 15 };
        o3 = E:{ 0 0 10 10 };
        o4 = E:{ 0 0 5 5 };
        o5 = E:{ 0 0 0 0 };
        o6 = E:{ 0 0 25 25 };
}

print_all(){
        print(h1);
        print(h2);
        print(h3);
        print(h4);
        print(h5);
        print(h6);
        print(h7);
        print(h8);
        print(h9);
        print(h10);

        print(e1);
        print(e2);
        print(e3);
        print(e4);
        print(e5);
        print(e6);
        print(e7);
        print(e8);
        print(e9);
        print(e10);
        print(e11);
        print(e12);
        print(e13);
        print(e14);
        print(e15);
```

```
		print(e16);
		print(e17);
		print(e18);
		print(e19);
		print(e20);

		print(l1);
		print(l2);
		print(l3);
		print(l4);
		print(l5);
		print(l6);
		print(l7);
		print(l8);
		print(l9);
		print(l10);
		print(l11);
		print(l12);
		print(l13);
		print(l14);
		print(l15);
		print(l16);

		print(o1);
		print(o2);
		print(o3);
		print(o4);
		print(o5);
		print(o6);
	}

	move_h(int x){
		int i;
		int increment;
		i = 0;
		increment = 10;
		while (i <= x){
			h1 -> { (100*i)/(x) (300*i)/(x) };
			h2 -> { (100*i)/(x) (280*i)/(x) };
			h3 -> { (100*i)/(x) (260*i)/(x) };
			h4 -> { (100*i)/(x) (240*i)/(x) };
			h5 -> { (100*i)/(x) (220*i)/(x) };
			h6 -> { (100*i)/(x) (200*i)/(x) };
			h7 -> { (120*i)/(x) (240*i)/(x) };
			h8 -> { (140*i)/(x) (240*i)/(x) };
			h9 -> { (140*i)/(x) (220*i)/(x) };
			h10 -> { (140*i)/(x) (200*i)/(x) };
			print_all();
			i = i+increment;
			halt(increment);
		}
	}

	move_e(int x){
		int i;
		int increment;
		i = 0;
		increment = 10;
		while (i <= x){
			e1 -> { (190*i)/(x) (220*i)/(x) };
			e2 -> { (200*i)/(x) (220*i)/(x) };
			e3 -> { (210*i)/(x) (220*i)/(x) };
```

```
                e4 -> { (220*i)/(x) (220*i)/(x) };
                e5 -> { (220*i)/(x) (220*i)/(x) };
                e6 -> { (215*i)/(x) (230*i)/(x) };
                e7 -> { (210*i)/(x) (235*i)/(x) };
                e8 -> { (205*i)/(x) (237*i)/(x) };
                e9 -> { (200*i)/(x) (237*i)/(x) };
                e10 -> { (195*i)/(x) (237*i)/(x) };
                e11 -> { (190*i)/(x) (235*i)/(x) };
                e12 -> { (185*i)/(x) (230*i)/(x) };
                e13 -> { (180*i)/(x) (225*i)/(x) };
                e14 -> { (180*i)/(x) (220*i)/(x) };
                e15 -> { (180*i)/(x) (215*i)/(x) };
                e16 -> { (185*i)/(x) (208*i)/(x) };
                e17 -> { (190*i)/(x) (205*i)/(x) };
                e18 -> { (195*i)/(x) (200*i)/(x) };
                e19 -> { (202*i)/(x) (200*i)/(x) };
                e20 -> { (210*i)/(x) (200*i)/(x) };
                print_all();
                i = i+increment;
                halt(increment);
        }
}

move_l(int x){
        int i;
        int increment;
        i = 0;
        increment = 10;
        while (i <= x){
                l1 -> { 240*i/x 200*i/x };
                l2 -> { 245*i/x 200*i/x };
                l3 -> { 250*i/x 200*i/x };
                l4 -> { 255*i/x 200*i/x };
                l5 -> { 260*i/x 200*i/x };
                l6 -> { 240*i/x 200*i/x };
                l7 -> { 240*i/x 320*i/x };
                l8 -> { 280*i/x 200*i/x };
                l9 -> { 300*i/x 200*i/x };
                l10 -> { 280*i/x 200*i/x };
                l11 -> { 280*i/x 220*i/x };
                l12 -> { 280*i/x 240*i/x };
                l13 -> { 280*i/x 260*i/x };
                l14 -> { 280*i/x 280*i/x };
                l15 -> { 280*i/x 300*i/x };
                l16 -> { 280*i/x 320*i/x };
                print_all();
                i = i+increment;
                halt(increment);
        }
}

move_o(int x){
        int i;
        int increment;
        i = 0;
        increment = 10;
        while (i <= x){
                o1 -> { 350*i/x 230*i/x };
                o2 -> { 350*i/x 230*i/x };
                o3 -> { 350*i/x 230*i/x };
                o4 -> { 350*i/x 230*i/x };
                o5 -> { 350*i/x 230*i/x };
```

```
                o6 -> { 350*i/x 230*i/x };
                print_all();
                i = i+increment;
                halt(increment);
        }
}

run(){
        initialize_h();
        initialize_e();
        initialize_l();
        initialize_o();
        move_h(1000);
        move_e(1000);
        move_l(1000);
        move_o(1000);
        print_all();
        halt(5000);
        halt(2000);
}
```

Appendix 2:

Here is the code for bounce.tbc:

```
ellipse c;

bounce(int velocity, int height, int direction){
    int v;
    int y;
    int new_y;
    int i;
    int d;
    int a;
    a = 50;
    d = direction;
    v = velocity;
    new_y = height;
    i = 0;
    while (i < 10000){
        y = new_y;
        if (d == 1){
            if (v > y){
               new_y = v-y;
               moveIN(y, 0, y*1000/v+1);
               moveIN(0, new_y, 1000-(y*1000/v)+1);
               v = v*new_y/v;
               d = 0;
            }
            else {
                new_y = y-v;
                moveIN(y, new_y, 1000);
                v = v+a;
            }
        }
        else{
            if (v < a){
                new_y = y;
                moveIN(y, new_y, 1);
                d = 1;
```

```
                v = a-v;
            }
            else {
                new_y = y+v;
                moveIN(y, new_y, 1000);
                v = v-a;
            }
        }
        i = i+1;
    }
}


moveIN(int old_y, int new_y, int t){
    int i;
    int increment;
    i = 0;
    increment = 10;
    while (i <= t){
            if (new_y < old_y){
            c -> { 200 (old_y-(old_y-new_y)*i/t)+25 };
        }
        else {
            c -> { 200 (old_y+(new_y-old_y)*i/t)+25 };
        }
            print(c);
        halt(increment);
        i = i+increment;
   }
}

run(){
    int i_h;
    i_h = 400;
    c = E:{ 200 250 25 25 };
    bounce(50, i_h, 1);
}
```

Appendix 3:

Here is the code for bubble_sort.tbc. Originally, before arrays were implemented, this code was 1700 lines long. Now that arrays have been implemented, the code is only 130 lines long.

```
rectangle r[8];

print_all(){
    int i;
    for (i = 0; i < 8; i = i+1){
        print(r[i]);
    }
}

test_height(int i){
    int x;
    int y;
    rectangle r1;
    rectangle r2;
    r1 = r[i];
    r2 = r[i+1];
    x = r1.v4;
    y = r2.v4;
```

```
        if (x > y){
            return 1;
        }
        else {
            return 0;
        }
    }

    test_area(int i){
        int x1;
        int x2;
        int y1;
        int y2;
        rectangle r1;
        rectangle r2;
        r1 = r[i];
        r2 = r[i+1];
        x1 = r1.v3;
        x2 = r1.v4;
        y1 = r2.v3;
        y2 = r2.v4;
        if (x1*x2 > y1*y2){
            return 1;
        }
        else {
            return 0;
        }
    }

    switch(int i){
        rectangle temp1;
        rectangle temp2;
        int val1;
        int val2;
        int val3;
        int val4;
        int tick;
        int delay;
        int height;
        delay = 10;
        height = 60;
        temp1 = r[i];
        temp2 = r[i+1];
        tick = 0;
        while (tick < height){
            temp1 => { 0 1 };
            temp2 => { 0 -1 };
            r[i] = temp1;
            r[i+1] = temp2;
            print_all();
            halt(delay);
            tick = tick + 1;
        }
        tick = 0;
        while (tick < 50){
            temp1 => { 1 0 };
            temp2 => { -1 0 };
            r[i] = temp1;
            r[i+1] = temp2;
            print_all();
            halt(delay);
            tick = tick + 1;
```

```
    }
    tick = 0;
    while (tick < height){
        temp1 => { 0 -1 };
        temp2 => { 0 1 };
        r[i] = temp1;
        r[i+1] = temp2;
        print_all();
        halt(delay);
        tick = tick + 1;
    }
    val1 = temp2.v1;
    val2 = temp2.v2;
    val3 = temp2.v3;
    val4 = temp2.v4;
    r[i] = R:{ val1 val2 val3 val4 };
    val1 = temp1.v1;
    val2 = temp1.v2;
    val3 = temp1.v3;
    val4 = temp1.v4;
    r[i+1] = R:{ val1 val2 val3 val4 };
}

bubble_sort(){
    int i;
    i = 0;
    while (i < 7){
        if (test_height(i)){
            switch(i);
            i = 0;
        }
        else{
            i = i+1;
        }
    }
}

run(){
    r[0] = R:{ 110 200 15 15 };
    r[1] = R:{ 160 200 30 40 };
    r[2] = R:{ 210 200 40 30 };
    r[3] = R:{ 260 200 5 70 };
    r[4] = R:{ 310 200 40 40 };
    r[5] = R:{ 360 200 35 20 };
    r[6] = R:{ 410 200 5 5 };
    r[7] = R:{ 460 200 20 10 };
    bubble_sort();
    print_all();
    halt(5000);
}
```

Appendix 4:

Following are all of the test cases used to test our compiler.

arith.tbc

```
run()
{
        int i;
        i = 3 + 5;
```

```
        print_int(i);
        print_int( 5 - 3);
        print_int( 3 * 5 );
        print_int( 3 - 5 * 20 / 4 + 1);
}
```

---

array_int.tbc

```
int a[10];

run()
{
        int b;
        int c[20];
        for (b=0;b<10;b=b+1) {
          a[b] = b * 2;
          print_int(a[b]);
        }
        for (b=0;b<20;b=b+1) {
          c[b] = b + 5;
          print_int(c[b]);
        }
        for (b=0;b<10;b=b+1) {
          print_int(c[b]-a[b]);
    }
}
```

---

array_shape.tbc

```
rectangle a[10];
ellipse e[10];
line l[10];

run()
{
        rectangle r;
        int b;
        for (b=0;b<9;b=b+1) {
          a[b] = R:{20+b*10 20+b*20 20+b*30 20+b*40};
          e[b] = E:{10+b*2 20*b*5 b*3 b*4};
          l[b] = E:{100+b*2 20*b*5 200+b 300+b*4};
          print(a[b]);
          print(e[b]);
          print(l[b]);
          r = a[b];
          print_int(r.v1);
        }
        halt(5000);
}
```

---

dumpstack.tbc

```
run()
{
        int i;
        i = 99999;
        dumpstack();
}
```

```
-----------------
func1.tbc

func(int i)
{
        return i * 2;
}

run()
{
        int i;
        i = func(2);
        print_int(i);
}


------------------
hello.tbc

run()
{
        string s;
        s = S:{100 100 "hello"};
}


--------------------
if.tbc

run()
{
        int i;
        if (2 < 3)
        {
                i = 9000;
        }
        else
        {
                i = 8000;
        }
        print_int(i);
}


----------------------
if2.tbc

run()
{
        if (0)
        {
                print_int(100);
        }
        else
        {
                print_int(111);
        }
}



-------------------
memberin.tbc
```

```
run()
{
        rectangle r;
        line l;
        ellipse e;
        r = R:{100 100 30 30};
        l = L:{200 100 20 30};
        e = E:{250 200 30 30};
        print_int(r.v1);
        print_int(e.v2);
        print_int(l.v3);
        print_int(r.v4);
}
```

---

move.tbc

```
run()
{
        int i;
        rectangle r;
        i = 10 + 20;
        r = R:{ 100 200 100 100 };
        print(r);

        halt(2000);
        r -> { 300 10 };
        print(r);
        halt(3000);
        r => { i 0 };
        print(r);
        halt(5000);
}
```

---

print.tbc

```
run()
{
        print_int(3);
}
```

---

return1.tbc

```
retr()
{
        return(R:{100 100 200 200});
}
run()
{
        rectangle r;
        r = retr();
        print(r);
        halt(5000);
}
```

---

return2.tbc

```
retr()
{
        return(R:{100 100 200 200});
}

rete()
{
        return(E:{100 200 50 50});
}

retl()
{
        return(L:{200 200 50 50});
}

reti()
{
        return(9);
}
run()
{
        rectangle r;
        r = retr();
        print(r);
        print(rete());
        print(retl());
        print_int(reti());
        halt(5000);
}
```

---

return3.tbc

```
returni(int i)
{
        return (i * 2);
}
run()
{
        print_int(returni(7));
}
```

---

shape.tbc

```
run()
{
        rectangle r;
        ellipse e;
        line l;
        r = R:{ 100 200 30 40 };
        e = E:{ 100 200 20 40 };
        l = L:{ 200 10 200 300 };
        print(e);
        print(r);
        print(l);
        halt(5000);
}
```

---

string-test.tbc

```
run()
{
        string s;
        rectangle r;
        r = R:{ 100 100 100 100 };
        s = S:{ 100 200 "hi hi hi" };
        halt(1000);

        print(s);
        halt(1000);
        s=> {50 50};
        print(s);

        halt(5000);

}
```

```
---------------------
while.tbc

run()
{
        int i;
        i = 0;
        while(i<5)
                {
                        print_int(i);
                        i = i + 1;
                }

}
```

8. Appendix

Source code:

ast.ml:

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

 type shape = Rect | Ellipse | Line | Undef

 type expr =
     Literal of int
   | String of string
   | Rectangle of string
   | Ellipse of string
   | Line of string
   | Shape of string
   | Id of string
   | Binop of expr * op * expr
   | Assign of string * expr
   | Call of string * expr list
   | Moveto of string * expr * expr
   | Moveby of string * expr * expr
   | Noexpr
   | GetV1 of string
   | GetV2 of string
   | GetV3 of string
```

```
    | GetV4 of string
    | Aid of string * expr
    | AAssign of string * expr * expr
    | REvaluation of expr * expr * expr * expr
    | EEvaluation of expr * expr * expr * expr
    | LEvaluation of expr * expr * expr * expr

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

 type vari_decl = {
     vtype : string;
         vsize : int;
     name : string;
   }

 type vari_value = {
     ltype : string;
     value : string;
   }

 type func_decl = {
     fname : string;
     formals : (vari_decl) list;
     locals : (vari_decl) list;
     body : stmt list;
   }

 type program = (vari_decl) list * func_decl list
```

bytecode.ml:

```
 type bstmt =
         (*push commands*)
           Litin of int (* Push a literal *)
         | Litst of string (*push string*)
         | Litsh of string (*push string*)

         | Drp (* Discard a value the bytecode interpreter will
handle the different types that can be dropped*)

         | Bin of Ast.op (* Perform arithmetic on top of stack *)

         (*copy of global with id of int to stack top*)
         | Lod of int (* puts global variable on top of stack *)

         (*store stack object in global variables given id*)
         | Str of int (* create global variable from top of stack *)

         (*these stay the same from micro C*)
         | Lfp of int (* Load frame pointer relative *)
         | Sfp of int (* Store frame pointer relative *)
         | Jsr of int (* Call function by absolute address *)
         | Ent of int (* Push FP, FP -> SP, SP += i *)
         | Rts of int (* Restore FP, SP, consume formals, push result
*)
         | Beq of int (* Branch relative if topofstack is zero *)
         | Bne of int (* Branch relative if topofstackis nonzero*)
         | Bra of int (* Branch relative *)
```

```
     | Lfpa of int (* This is the start index of this array variable.
Index is evaluated and
                                          put on top of stack in an
int structure. *)
     | Sfpa of int
     | Loda of int
     | Stra of int
         | Hlt (* Terminate *)

         (*these are added for graphic functions*)
         | Sgraph (*create empty graph*)
         | Egraph (* Erase the whole graph *)
         (*freeze and move commands*)
         | Susp (* Freeze for int milliseconds specified on top of
stack*)
         | Movby (* move object on top of stack by the x and y
ammounts specified by the two integers on teh stack above it*)
         | Movto
         (*get specfic value of object*)
         | GetC (*get value specified by int on top of stack *)
         | MakeS (* Take top 5 ints and then make a new shape. from
top to bottom: type, v1, v2, v3, v4 *)

 type prog = {
         size_globals : int; (* Number of global variables *)
         text : bstmt array; (* Code for all the functions *)
 }

 let string_of_stmt = function
                 Litin(i) -> "Litin " ^ string_of_int i
         |         Drp                -> "Drp"
         |        Bin(Ast.Add) -> "Add"
         |   Bin(Ast.Sub) -> "Sub"
         |   Bin(Ast.Mult) -> "Mult"
         |   Bin(Ast.Div) -> "Div"
         |   Bin(Ast.Equal) -> "Equal"
         |   Bin(Ast.Neq) -> "Neq"
         |   Bin(Ast.Less) -> "Less"
         |   Bin(Ast.Leq) -> "Leq"
         |   Bin(Ast.Geq) -> "Geq"
         |   Bin(Ast.Greater) -> "Greater"
         |   Lod(i) -> "Lod " ^ string_of_int i
         |   Str(i) -> "Str " ^ string_of_int i
         |   Lfp(i) -> "Lfp " ^ string_of_int i
         |   Sfp(i) -> "Sfp " ^ string_of_int i
         |   Jsr(i) -> "Jsr " ^ string_of_int i
         |   Ent(i) -> "Ent " ^ string_of_int i
         |   Rts(i) -> "Rts " ^ string_of_int i
         |   Bne(i) -> "Bne " ^ string_of_int i
         |   Beq(i) -> "Beq " ^ string_of_int i
         |   Bra(i) -> "Bra " ^ string_of_int i
         |   Litsh(i) -> "Litsh" ^ i
         |   Litst(i) -> "Litst" ^ i
         |   Egraph -> "Egraph"
         |   Sgraph -> "Sgraph"
         |   Hlt -> "Hlt"
         |   Movto -> "Movto"
         |   Movby -> "Movby"
         |   Susp -> "Susp"
         |   GetC -> "GetC"
     |   MakeS -> "MakeS"
         |        Loda(i) -> "Loda " ^ string_of_int i
```

```
          |        Stra(i) -> "Stra " ^ string_of_int i
          |        Lfpa(i) -> "Lfpa " ^ string_of_int i
          |        Sfpa(i) -> "Sfpa " ^ string_of_int i

 let string_of_prog p =
         string_of_int p.size_globals ^ " slots to store global
variables\n" ^
         let funca = Array.mapi
                 (fun i s -> string_of_int i ^ " " ^ string_of_stmt
s) p.text
         in String.concat "\n" (Array.to_list funca)
```

compile.ml

```
 (* Compile: Convert predefined functions into bytecode series. *)
 open Ast
 open Bytecode

 module StringMap = Map.Make(String)

 (* Symbol table: Information about all the names in scope *)
 type env = {
     function_index : int StringMap.t; (* Index for each function *)
     global_index   : int StringMap.t; (* "Address" for global
variables *)
     local_index    : int StringMap.t; (* FP offset for args, locals
*)
   }

 (* val enum : int -> 'a list -> (int * 'a) list *)
 let rec enum stride n = function (*Change here: this func maps
variable name to offsets. *)
     [] -> []
   | hd::tl ->  (* hd is the vari_decl  *)
         if stride > 0 then
                 match hd.vtype with
                         "int" -> (* Allocate global storage space
for an int *)
                                 (n + 1, hd.name) :: enum stride
(n+stride * 2) tl
                 |        "string" -> (* Here's the question: how
many slots need to be allocated to string? *)
                                                 (* To make it
simpler, allocate 30 slots for it *)
                                 (n + 29, hd.name) :: enum stride
(n+stride * 30) tl
                 |        "line" ->
                                 (n + 4, hd.name) :: enum stride
(n+stride * 5) tl
                 |        "rect" ->
                                 (n + 4, hd.name) :: enum stride
(n+stride * 5) tl
                 |        "ellipse" ->
                                 (n + 4, hd.name) :: enum stride
(n+stride * 5) tl
                 |        "shape" ->
                                 (n + 4, hd.name) :: enum stride
(n+stride * 5) tl
         |   "arrayi" ->
                                 (n + 2*hd.vsize-1, hd.name) :: enum
stride (n+stride * 2 * hd.vsize) tl
         |   "arrayr" ->
                                 (n + 5*hd.vsize-1, hd.name) :: enum
stride (n+stride * 5 * hd.vsize) tl
```

```
         |   "arraye" ->
                                  (n + 5*hd.vsize-1, hd.name) :: enum
stride (n+stride * 5 * hd.vsize) tl
         |   "arrayl" ->
                                  (n + 5*hd.vsize-1, hd.name) :: enum
stride (n+stride * 5 * hd.vsize) tl
         |   "arrays" ->
                                  (n + 5*hd.vsize-1, hd.name) :: enum
stride (n+stride * 5 * hd.vsize) tl
                 |        "bind" ->
                                                  (* so the problem is
with bind: allocate 100 slots. *)
                                  (n + 99, hd.name) :: enum stride
(n+stride * 100) tl
                 |         _ -> raise(Failure ("Undefined type with
variable" ^ hd.name))
         else
                 match hd.vtype with
                         "int" -> (* Allocate global storage space
for an int *)
                                  (n, hd.name) :: enum stride
(n+stride * 2) tl
                 |        "string" -> (* Here's the question: how
many slots need to be allocated to string? *)
                                                  (* To make it
simpler, allocate 30 slots for it *)
                                  (n, hd.name) :: enum stride
(n+stride * 30) tl
                 |        "line" ->
                                  (n, hd.name) :: enum stride
(n+stride * 5) tl
                 |        "rect" ->
                                  (n, hd.name) :: enum stride
(n+stride * 5) tl
                 |        "ellipse" ->
                                  (n, hd.name) :: enum stride
(n+stride * 5) tl
                 |        "shape" ->
                                  (n, hd.name) :: enum stride
(n+stride * 5) tl
                 |        "arrayi" ->
                                  (n, hd.name) :: enum stride
(n+stride * 2 * hd.vsize) tl
                 |        "arrayr" ->
                                  (n, hd.name) :: enum stride
(n+stride * 5 * hd.vsize) tl
                 |        "arraye" ->
                                  (n, hd.name) :: enum stride
(n+stride * 5 * hd.vsize) tl
                 |        "arrayl" ->
                                  (n, hd.name) :: enum stride
(n+stride * 5 * hd.vsize) tl
                 |        "arrays" ->
                                  (n, hd.name) :: enum stride
(n+stride * 5 * hd.vsize) tl
                 |         _ -> raise(Failure ("Undefined type with
variable " ^ hd.name))

 (* val enum : int -> 'a list -> (int * 'a) list *)
 let rec enum_func stride n = function
     [] -> []
   | hd::tl -> (n, hd) :: enum_func stride (n+stride) tl
```

```
 let get_vari_size a vlist =
          List.fold_left (fun a b -> a + (match b.vtype with
                                                                                                  "int"
-> 2
                                                                                                |
  "string" -> 30
                                                                                                |
  "rect" -> 5
                                                                                                |
  "ellipse" -> 5
                                                                                                |
  "line" -> 5
                                                                                                |
  "shape" -> 5
                                                                                                |
  "bind"  -> 100
                                                                                                |
        "arrayi" -> b.vsize*2
                                                                                                |
        "arrayr" -> b.vsize*5
                                                                                                |
        "arraye" -> b.vsize*5
                                                                                                |
        "arrayl" -> b.vsize*5
                                                                                                |
        "arrays" -> b.vsize*5
                                                                                                |
  _ -> raise(Failure("Error in get_vari_size !!"))
                                                                                               ))
0 vlist

 (* val string_map_pairs StringMap 'a -> (int * 'a) list -> StringMap
'a *)
 let string_map_pairs map pairs =
   List.fold_left (fun m (i, n) -> StringMap.add n i m) map pairs (*
Add (var_name, offset) pairs *)

 (** Translate a program in AST form into a bytecode program.  Throw
an
     exception if something is wrong, e.g., a reference to an unknown
     variable or function *)
 (* The input of translate are two lists: one for global, one for
function. Compiler
    takes the two lists, which is generated by parser, to be bytecode
stuff. *)
 let translate (globals, functions) = (* globals has the form of
vari_decl *)

   (* Allocate "addresses" and storing place for each global variable
*)
   let global_indexes = string_map_pairs StringMap.empty (enum 1 0
globals) in

   (* Assign indexes to function names; built-ins are special *)
   let built_in_functions = StringMap.add "print" (-1)
StringMap.empty in
   let built_in_functions = StringMap.add "halt" (-2)
built_in_functions in
         (* -3 maps to MakeBind which is deleted *)
   let built_in_functions = StringMap.add "printarray" (-3)
built_in_functions in
   let built_in_functions = StringMap.add "print_int" (-4)
```

```
built_in_functions in
   let built_in_functions = StringMap.add "dumpstack" (-5)
built_in_functions in
   let function_indexes = string_map_pairs built_in_functions
       (enum_func 1 1 (List.map (fun f -> f.fname) functions)) in

   (* Translate a function in AST form into a list of bytecode
statements *)
   let translate env fdecl =
     (* Bookkeeping: FP offsets for locals and arguments *)
     let size_formals = get_vari_size 0 fdecl.formals
     and size_locals = get_vari_size 0 fdecl.locals
     and local_offsets = enum 1 1 fdecl.locals (* This is list of
pair (#, local_name) *)
     and formal_offsets = enum (-1) (-2) fdecl.formals in (* The 1
and -2 in this and last line means the first variable's address, if
any. *)
     let env = { env with local_index = string_map_pairs (* Not quite
understand this line. *)
                   StringMap.empty (local_offsets @ formal_offsets) }
in

     let rec expr = function (* Try to implement built-in functions
here. *)
         Literal i -> [Litin i] (* Give input as an int. *)
       | String s -> (let s2 = String.sub s 3 (String.length s - 4)
                in [Litst s2])
       | Rectangle r -> let r2 = String.sub r 3 (String.length r - 4)
(* No need of type check. Done in parser. *)
                in [Litsh ("3 " ^ r2)]
       | Ellipse e -> let e2 = String.sub e 3 (String.length e - 4)
                in [Litsh ("4 " ^ e2)]
       | Line l -> let l2 = String.sub l 3 (String.length l - 4)
                in [Litsh ("5 " ^ l2)]
       | Shape s -> let s2 = String.sub s 3 (String.length s - 4)
                in [Litsh ("6 " ^ s2)]
       | Id s -> (* fetch variables according to id. put this on top
of stack. *)
           (try [Lfp (StringMap.find s env.local_index)]
           with Not_found -> try [Lod (StringMap.find s
env.global_index)]
           with Not_found -> raise (Failure ("undeclared variable " ^
s)))
       | Binop (e1, op, e2) -> expr e1 @ expr e2 @ [Bin op] (* Take
care + might be overloaded to binds. *)
       | Assign (s, e) -> expr e @
           (try [Sfp (StringMap.find s env.local_index)]
             with Not_found -> try [Str (StringMap.find s
env.global_index)]
           with Not_found -> raise (Failure ("undeclared variable " ^
s)))
       | Call (fname, actuals) -> (try (* How about the special
functions about opengraph, close graph, halt,? *)
           (List.concat (List.map expr (List.rev actuals))) @
           [ Jsr (StringMap.find fname env.function_index) ]
         with Not_found -> raise (Failure ("undefined function " ^
fname)))
       | Noexpr -> []
           | Moveby(id, e1, e2) -> (* Move only handles with
numbers*)
                 (try [Lfp (StringMap.find id env.local_index)]
                         with Not_found -> try[Lod (StringMap.find id
```

```
env.global_index)]
                        with Not_found -> raise (Failure
("undeclared variable " ^ id)))
                @ expr e2 @ expr e1 @ [Movby] @ [Drp] @ [Drp] @
                (try [Sfp (StringMap.find id env.local_index)]
                        with Not_found -> try[Str (StringMap.find id
env.global_index)]
                        with Not_found -> raise (Failure
("undeclared variable " ^ id)))

          | Moveto(id, e1, e2) -> (* Move only handles with
numbers*)
                (try [Lfp (StringMap.find id env.local_index)]
                        with Not_found -> try[Lod (StringMap.find id
env.global_index)]
                        with Not_found -> raise (Failure
("undeclared variable " ^ id)))
                @ expr e2 @ expr e1 @ [Movto] @ [Drp] @ [Drp] @
                (try [Sfp (StringMap.find id env.local_index)]
                        with Not_found -> try[Str (StringMap.find id
env.global_index)]
                        with Not_found -> raise (Failure
("undeclared variable " ^ id)))

          | GetV1(id) -> (* Get first value of object *)
                (try [Lfp (StringMap.find id env.local_index)]
                        with Not_found -> try[Lod (StringMap.find id
env.global_index)]
                        with Not_found -> raise (Failure
("undeclared variable " ^ id)))
                @ [Litin 1] @ [GetC]

          | GetV2(id) -> (* Get second value of object *)
                (try [Lfp (StringMap.find id env.local_index)]
                        with Not_found -> try[Lod (StringMap.find id
env.global_index)]
                        with Not_found -> raise (Failure
("undeclared variable " ^ id)))
                @ [Litin 2] @ [GetC]

          | GetV3(id) -> (* Get third value of object *)
                (try [Lfp (StringMap.find id env.local_index)]
                        with Not_found -> try[Lod (StringMap.find id
env.global_index)]
                        with Not_found -> raise (Failure
("undeclared variable " ^ id)))
                @ [Litin 3] @ [GetC]

          | GetV4(id) -> (* Get fourth value of object *)
                (try [Lfp (StringMap.find id env.local_index)]
                        with Not_found -> try[Lod (StringMap.find id
env.global_index)]
                        with Not_found -> raise (Failure
("undeclared variable " ^ id)))
                @ [Litin 4] @ [GetC]

          | Aid(id, e) -> (* Retrieve Array value on top of stack *)
                expr e @ (try [Lfpa(StringMap.find id
env.local_index)]
                        with Not_found -> try[Loda (StringMap.find
id env.global_index)]
                        with Not_found -> raise (Failure
```

```
("undeclared variable" ^ id)))

          | AAssign(id, e1, e2) -> (* Assign value to an array
element. *)
                expr e2 @ expr e1 @ (try [Sfpa(StringMap.find id
env.local_index)]
                          with Not_found -> try[Stra(StringMap.find id
env.global_index)]
                          with Not_found -> raise (Failure
("undelcared variable" ^ id)))

          | REvaluation(v1, v2, v3, v4) -> (* Evaluate every
variable in a rectangle*)
                expr v4 @ expr v3 @ expr v2 @ expr v1 @ [Litin 3] @
[MakeS]

          | EEvaluation(v1, v2, v3, v4) -> (* Evaluate every
variable in a rectangle*)
                expr v4 @ expr v3 @ expr v2 @ expr v1 @ [Litin 4] @
[MakeS]

          | LEvaluation(v1, v2, v3, v4) -> (* Evaluate every
variable in a rectangle*)
                expr v4 @ expr v3 @ expr v2 @ expr v1 @ [Litin 5] @
[MakeS]

    in let rec stmt = function
        Block sl     ->  List.concat (List.map stmt sl)
      | Expr e       -> expr e @ [Drp]
      | Return e     -> expr e @ [Rts size_formals]
      | If (p, t, f) -> let t' = stmt t and f' = stmt f in
        expr p @ [Beq(2 + List.length t')] @
        t' @ [Bra(1 + List.length f')] @ f'
      | For (e1, e2, e3, b) ->
          stmt (Block([Expr(e1); While(e2, Block([b; Expr(e3)]))]))
      | While (e, b) ->
          let b' = stmt b and e' = expr e in
          [Bra (1+ List.length b')] @ b' @ e' @
          [Bne (-(List.length b' + List.length e'))]

    in [Ent size_locals] @      (* Entry: allocate space for locals.
Need to change as each variable has different size. *)
    stmt (Block fdecl.body) @  (* Body *)
    [Litin 0; Rts size_formals]   (* Default = return 0 *)

  in let env = { function_index = function_indexes;
                 global_index = global_indexes;
                 local_index = StringMap.empty } in

  (* Code executed to start the program: Jsr run; halt *)
  let entry_function = try
    [Sgraph; Jsr (StringMap.find "run" function_indexes); Hlt]
  with Not_found -> raise (Failure ("no \"run\" function"))
  in

  (* Compile the functions *)
  let func_bodies = entry_function :: List.map (translate env)
functions in

  (* Calculate function entry points by adding their lengths *)
  let (fun_offset_list, _) = List.fold_left
      (fun (l,i) f -> (i :: l, (i + List.length f))) ([],0)
func_bodies in
```

```
  let func_offset = Array.of_list (List.rev fun_offset_list) in

 (* This is what compiler generates: *)
  { size_globals = get_vari_size 0 globals; (* This globals must be
a complete layout of global var. *)
    (* Concatenate the compiled functions and replace the function
       indexes in Jsr statements with PC values *)
    text = Array.of_list (List.map (function
        Jsr i when i > 0 -> Jsr func_offset.(i)
      | _ as s -> s) (List.concat func_bodies))
  }
```

Execute.ml

```
open Ast
open Bytecode
open Thread

exception IllegalPrint;;
exception IllegalMove;;

let trim s =
  let s' = Str.replace_first (Str.regexp "^[ \t\n]+") "" s in
  Str.replace_first (Str.regexp "[ \t\n]+$") "" s';;

let explode s =
  let rec f acc = function
    | -1 -> acc
    | k -> f (s.[k] :: acc) (k - 1)
  in f [] (String.length s - 1) ;;

(*execute the program*)
let execute_prog prog =
        let stack = Array.make 8192 0
        and globals = Array.make prog.size_globals 0 in

        (*exec runs op at stack position
          fp is frame pointer
          sp is stack pointer
          pc is program counter *)

        let rec exec fp sp pc = match prog.text.(pc) with

                (*lits will push values on stack*)
                (*int: type 1 and value*)
                Litin i -> stack.(sp) <- i; stack.(sp+1) <- 1;  exec
fp (sp+2) (pc+1)

                (* string: type 2, x, y, # of char, chars -- total
size is automatically 30*)
        |        Litst str ->
                let trimmed = trim str in
                 let split_trim = Str.split(Str.regexp "\"") trimmed
 in
                  let ascii_list = List.rev (List.map Char.code
(explode (List.nth(split_trim)(1)))) in
                   let length = List.length ascii_list in
                    let diff = 26 - length in
                     let coord_list = Str.split(Str.regexp "[ \t]+")
(List.nth(split_trim)(0)) in
                      let rec fill_string remaining = if (remaining
> 0) then (stack.(sp+diff-remaining) <- 0;
                                                fill_string
(remaining-1)) else exec fp (sp+30) (pc+1) in
```

```
                        let rec push_elements list index = if
List.length list > 0
                                            then (stack.(sp+diff+index)
<- (List.hd list);
                                                    push_elements
(List.tl list) (index+1))
                                               else (stack.(sp+26) <-
length;
                                                       stack.(sp+27)
<- int_of_string (List.nth(coord_list)(1));
                                                         stack.(sp+28)
<- int_of_string (List.nth(coord_list)(0));
                                                         stack.(sp+29)
<- 2;
                                                         fill_string
diff)
                                                      in
                                             push_elements ascii_list
0

                (* shape: type , push x1, push y1, push x2, push
y2*)
        |         Litsh shp ->
                 let s_value = Str.split(Str.regexp("[ \t]+"))(shp)
in
                          stack.(sp+4) <- int_of_string
(List.nth(s_value)(0));
                          stack.(sp+3) <- int_of_string
(List.nth(s_value)(1));
                          stack.(sp+2) <- int_of_string
(List.nth(s_value)(2));
                          stack.(sp+1) <- int_of_string
(List.nth(s_value)(3));
                          stack.(sp) <- int_of_string
(List.nth(s_value)(4));
                          exec fp (sp+5) (pc+1)

                (*whatever is stored on top of stack*)
        | Drp ->
                let obj_id = stack.(sp-1) in
                (
                        match obj_id with
                                1 -> exec fp (sp-2) (pc+1)
                        |        2 -> exec fp (sp-30) (pc+1)
                        |        3 -> exec fp (sp-5) (pc+1)
                        |        4 -> exec fp (sp-5) (pc+1)
                        |        5 -> exec fp (sp-5) (pc+1)
                        |        6 -> exec fp (sp-5) (pc+1)
                        | _ -> raise(Failure("Unmatched type!!")))

        (*binary operation*)
        | Bin op -> let op1 = stack.(sp-4) and op2 = stack.(sp-2) in

             stack.(sp-4) <- (let boolean i = if i then 1 else 0 in
             match op with
                        Add     -> op1 + op2
             |          Sub     -> op1 - op2
             |          Mult    -> op1 * op2
             |          Div     -> op1 / op2
             |          Equal   -> boolean (op1 =  op2)
             |          Neq     -> boolean (op1 != op2)
             |          Less    -> boolean (op1 <  op2)
```

```
                |         Leq     -> boolean (op1 <= op2)
                |         Greater -> boolean (op1 >  op2)
                |         Geq     -> boolean (op1 >= op2)) ;
                exec fp (sp-2) (pc+1)

          (*copy global variable onto stack
            this is passed the first index on the array of globals
that will hold the identifier*)
          | Lod index -> (* Fetch global variable *)
                 let obj_id = globals.(index) in
                 (
                         match obj_id with
                                 1 ->
                                         stack.(sp) <- globals.
(index-1);
                                         stack.(sp+1) <- globals.
(index);
                                         exec fp (sp+2) (pc+1)

                         |       2 ->
                                         for i=0 to 29 do
                                                 stack.(sp+i) <-
globals.(index-29+i)
                                         done;
                                         exec fp (sp+30) (pc+1)

                         |       3 ->
                                         stack.(sp) <- globals.
(index-4);
                                         stack.(sp+1) <- globals.
(index-3);
                                         stack.(sp+2) <- globals.
(index-2);
                                         stack.(sp+3) <- globals.
(index-1);
                                         stack.(sp+4) <- globals.
(index);
                                         exec fp (sp+5) (pc+1)
                         |       4 ->
                                         stack.(sp) <- globals.
(index-4);
                                         stack.(sp+1) <- globals.
(index-3);
                                         stack.(sp+2) <- globals.
(index-2);
                                         stack.(sp+3) <- globals.
(index-1);
                                         stack.(sp+4) <- globals.
(index);
                                         exec fp (sp+5) (pc+1)
                         |       5 ->
                                         stack.(sp) <- globals.
(index-4);
                                         stack.(sp+1) <- globals.
(index-3);
                                         stack.(sp+2) <- globals.
(index-2);
                                         stack.(sp+3) <- globals.
(index-1);
                                         stack.(sp+4) <- globals.
(index);
                                         exec fp (sp+5) (pc+1)
```

```
                            |        6 ->
                                    stack.(sp) <- globals.
(index-4);
                                    stack.(sp+1) <- globals.
(index-3);
                                    stack.(sp+2) <- globals.
(index-2);
                                    stack.(sp+3) <- globals.
(index-1);
                                    stack.(sp+4) <- globals.
(index);
                                    exec fp (sp+5) (pc+1)

                            | _ -> raise(Failure("Unmatched type!!"))
                )
          | Loda index -> (* Fetch global array variable *)
                if (stack.(sp-1) <> 1) then raise(Failure("Array
type check failure!")) else
                let loffset = stack.(sp-2) in
                let obj_id = globals.(index) in
                let lsize = (match obj_id with
                                    1 -> 2
                                |        2 -> raise(Failure("No
support of string array!"))
                                |   3 -> 5
                                |   4 -> 5
                                |   5 -> 5
                                |   6 -> 5
                                |    _ ->
(ignore(print_endline(string_of_int(globals.(index))));
raise(Failure("TTType check error!"))))
                in
                (
                        match obj_id with
                                1 ->
                                    stack.(sp) <- globals.
(index-1-lsize*loffset);
                                    stack.(sp+1) <- globals.
(index-lsize*loffset);
                                    exec fp (sp+2) (pc+1)

                            |        2 ->
                                    for i=0 to 29 do
                                            stack.(sp+i) <-
globals.(index-29+i)
                                    done;
                                    exec fp (sp+30) (pc+1)

                            |        3 ->
                                    stack.(sp) <- globals.
(index-4-lsize*loffset);
                                    stack.(sp+1) <- globals.
(index-3-lsize*loffset);
                                    stack.(sp+2) <- globals.
(index-2-lsize*loffset);
                                    stack.(sp+3) <- globals.
(index-1-lsize*loffset);
                                    stack.(sp+4) <- globals.
(index-lsize*loffset);
                                    exec fp (sp+5) (pc+1)
                            |        4 ->
                                    stack.(sp) <- globals.
```

```
(index-4-lsize*loffset);
                                        stack.(sp+1) <- globals.
(index-3-lsize*loffset);
                                        stack.(sp+2) <- globals.
(index-2-lsize*loffset);
                                        stack.(sp+3) <- globals.
(index-1-lsize*loffset);
                                        stack.(sp+4) <- globals.
(index-lsize*loffset);
                                        exec fp (sp+5) (pc+1)
                        |       5 ->
                                        stack.(sp) <- globals.
(index-4-lsize*loffset);
                                        stack.(sp+1) <- globals.
(index-3-lsize*loffset);
                                        stack.(sp+2) <- globals.
(index-2-lsize*loffset);
                                        stack.(sp+3) <- globals.
(index-1-lsize*loffset);
                                        stack.(sp+4) <- globals.
(index-lsize*loffset);
                                        exec fp (sp+5) (pc+1)
                        |       6 ->
                                        stack.(sp) <- globals.
(index-4-lsize*loffset);
                                        stack.(sp+1) <- globals.
(index-3-lsize*loffset);
                                        stack.(sp+2) <- globals.
(index-2-lsize*loffset);
                                        stack.(sp+3) <- globals.
(index-1-lsize*loffset);
                                        stack.(sp+4) <- globals.
(index-lsize*loffset);
                                        exec fp (sp+5) (pc+1)

                        | _ -> raise(Failure("Unmatched type!!"))
                )

        (*store stack object in global variables given id*)
        | Str index ->
                let obj_id = stack.(sp-1) in
                (
                        match obj_id with
                                1 ->
                                        globals.(index-1) <- stack.
(sp-2);
                                        globals.(index) <- stack.
(sp-1);
                                        exec fp (sp) (pc+1)
                        |       2 ->

                                        for i=0 to 29 do
                                                globals.(index-29+i)
<- stack.(sp-30+i)
                                        done;
                                        exec fp (sp) (pc+1)

                        |       3 ->
                                        globals.(index-4) <- stack.
(sp-5);
                                        globals.(index-3) <- stack.
(sp-4);
```

```
                                        globals.(index-2) <- stack.
(sp-3);
                                        globals.(index-1) <- stack.
(sp-2);
                                        globals.(index) <- stack.
(sp-1);
                                        exec fp (sp) (pc+1)

                        |       4 ->
                                        globals.(index-4) <- stack.
(sp-5);
                                        globals.(index-3) <- stack.
(sp-4);
                                        globals.(index-2) <- stack.
(sp-3);
                                        globals.(index-1) <- stack.
(sp-2);
                                        globals.(index) <- stack.
(sp-1);
                                        exec fp (sp) (pc+1)
                        |       5 ->
                                        globals.(index-4) <- stack.
(sp-5);
                                        globals.(index-3) <- stack.
(sp-4);
                                        globals.(index-2) <- stack.
(sp-3);
                                        globals.(index-1) <- stack.
(sp-2);
                                        globals.(index) <- stack.
(sp-1);
                                        exec fp (sp) (pc+1)
                        |       6 ->
                                        globals.(index-4) <- stack.
(sp-5);
                                        globals.(index-3) <- stack.
(sp-4);
                                        globals.(index-2) <- stack.
(sp-3);
                                        globals.(index-1) <- stack.
(sp-2);
                                        globals.(index) <- stack.
(sp-1);
                                        exec fp (sp) (pc+1)

                        | _ -> raise(Failure("Unmatched type!!")))

        (*store stack object in global variables given id*)
        | Stra index ->
                if (stack.(sp-1) <> 1) then raise(Failure("Array
type check failure!")) else
                let obj_id = stack.(sp-3)
                and loffset = stack.(sp-2)
                in
                (
                        match obj_id with
                                1 ->
                                        globals.(index-1-2*loffset)
<- stack.(sp-2-2);
                                        globals.(index-2*loffset) <-
stack.(sp-1-2);
                                        exec fp (sp) (pc+1)
```

```
                                | 2 ->
                                        for i=0 to 29 do
                                                globals.(index-29+i)
        <- stack.(sp-30+i)
                                        done;
                                        exec fp (sp) (pc+1)

                                | 3 ->
                                        globals.(index-4-5*loffset)
        <- stack.(sp-5-2);
                                        globals.(index-3-5*loffset)
        <- stack.(sp-4-2);
                                        globals.(index-2-5*loffset)
        <- stack.(sp-3-2);
                                        globals.(index-1-5*loffset)
        <- stack.(sp-2-2);
                                        globals.(index-5*loffset) <-
        stack.(sp-1-2);
                                        exec fp (sp) (pc+1)

                                | 4 ->
                                        globals.(index-4-5*loffset)
        <- stack.(sp-5-2);
                                        globals.(index-3-5*loffset)
        <- stack.(sp-4-2);
                                        globals.(index-2-5*loffset)
        <- stack.(sp-3-2);
                                        globals.(index-1-5*loffset)
        <- stack.(sp-2-2);
                                        globals.(index-5*loffset) <-
        stack.(sp-1-2);
                                        exec fp (sp) (pc+1)
                                | 5 ->
                                        globals.(index-4-5*loffset)
        <- stack.(sp-5-2);
                                        globals.(index-3-5*loffset)
        <- stack.(sp-4-2);
                                        globals.(index-2-5*loffset)
        <- stack.(sp-3-2);
                                        globals.(index-1-5*loffset)
        <- stack.(sp-2-2);
                                        globals.(index-5*loffset) <-
        stack.(sp-1-2);
                                        exec fp (sp) (pc+1)
                                | 6 ->
                                        globals.(index-4-5*loffset)
        <- stack.(sp-5-2);
                                        globals.(index-3-5*loffset)
        <- stack.(sp-4-2);
                                        globals.(index-2-5*loffset)
        <- stack.(sp-3-2);
                                        globals.(index-1-5*loffset)
        <- stack.(sp-2-2);
                                        globals.(index-5*loffset) <-
        stack.(sp-1-2);
                                        exec fp (sp) (pc+1)

                                | _ -> raise(Failure("Unmatched type!!")))

                (* same operations as in microc *)
                | Lfp i ->
```

```
                    let obj_id = stack.(fp+i) in
                    (
                            match obj_id with
                                    1 ->
                                            stack.(sp) <- stack.(fp+i-
1);
                                            stack.(sp+1) <- stack.
(fp+i);
                                            exec fp (sp+2) (pc+1)
                            |       2 ->
                                            for j=0 to 29 do
                                                    stack.(sp+j) <-
stack.(fp+i-29+j)
                                            done;
                                            exec fp (sp+30) (pc+1)
                            |       3 ->
                                            stack.(sp) <- stack.(fp+i-
4);
                                            stack.(sp+1) <- stack.(fp+i-
3);
                                            stack.(sp+2) <- stack.(fp+i-
2);
                                            stack.(sp+3) <- stack.(fp+i-
1);
                                            stack.(sp+4) <- stack.
(fp+i);
                                            exec fp (sp+5) (pc+1)
                            |       4 ->
                                            stack.(sp) <- stack.(fp+i-
4);
                                            stack.(sp+1) <- stack.(fp+i-
3);
                                            stack.(sp+2) <- stack.(fp+i-
2);
                                            stack.(sp+3) <- stack.(fp+i-
1);
                                            stack.(sp+4) <- stack.
(fp+i);
                                            exec fp (sp+5) (pc+1)
                            |       5 ->
                                            stack.(sp) <- stack.(fp+i-
4);
                                            stack.(sp+1) <- stack.(fp+i-
3);
                                            stack.(sp+2) <- stack.(fp+i-
2);
                                            stack.(sp+3) <- stack.(fp+i-
1);
                                            stack.(sp+4) <- stack.
(fp+i);
                                            exec fp (sp+5) (pc+1)
                            |       6 ->
                                            stack.(sp) <- stack.(fp+i-
4);
                                            stack.(sp+1) <- stack.(fp+i-
3);
                                            stack.(sp+2) <- stack.(fp+i-
2);
                                            stack.(sp+3) <- stack.(fp+i-
1);
                                            stack.(sp+4) <- stack.
(fp+i);
```

```
                                        exec fp (sp+5) (pc+1)

                        | _ -> raise(Failure("Unmatched type!!")))

         | Lfpa i -> (* This is the base address of array. Now, on
top of stack, it's the element offset*)
                if (stack.(sp-1) <> 1 ) then raise(Failure("Array
index type check failure!!")) else
                let obj_id = stack.(fp+i)
                and loffset = stack.(sp-2) in
                (
                        match obj_id with
                                1 ->
                                        stack.(sp) <- stack.(fp+i-1-
loffset*2);
                                        stack.(sp+1) <- stack.(fp+i-
loffset*2);
                                        exec fp (sp+2) (pc+1)
                        |       2 ->
                                        for j=0 to 29 do
                                                stack.(sp+j) <-
stack.(fp+i-29+j)
                                        done;
                                        exec fp (sp+30) (pc+1)
                        |       3 ->
                                        stack.(sp) <- stack.(fp+i-4-
loffset*5);
                                        stack.(sp+1) <- stack.(fp+i-
3-loffset*5);
                                        stack.(sp+2) <- stack.(fp+i-
2-loffset*5);
                                        stack.(sp+3) <- stack.(fp+i-
1-loffset*5);
                                        stack.(sp+4) <- stack.(fp+i-
loffset*5);
                                        exec fp (sp+5) (pc+1)
                        |       4 ->
                                        stack.(sp) <- stack.(fp+i-4-
loffset*5);
                                        stack.(sp+1) <- stack.(fp+i-
3-loffset*5);
                                        stack.(sp+2) <- stack.(fp+i-
2-loffset*5);
                                        stack.(sp+3) <- stack.(fp+i-
1-loffset*5);
                                        stack.(sp+4) <- stack.(fp+i-
loffset*5);
                                        exec fp (sp+5) (pc+1)
                        |       5 ->
                                        stack.(sp) <- stack.(fp+i-4-
loffset*5);
                                        stack.(sp+1) <- stack.(fp+i-
3-loffset*5);
                                        stack.(sp+2) <- stack.(fp+i-
2-loffset*5);
                                        stack.(sp+3) <- stack.(fp+i-
1-loffset*5);
                                        stack.(sp+4) <- stack.(fp+i-
loffset*5);
                                        exec fp (sp+5) (pc+1)
                        |       6 ->
                                        stack.(sp) <- stack.(fp+i-4-
```

```
loffset*5);
                                        stack.(sp+1) <- stack.(fp+i-
3-loffset*5);
                                        stack.(sp+2) <- stack.(fp+i-
2-loffset*5);
                                        stack.(sp+3) <- stack.(fp+i-
1-loffset*5);
                                        stack.(sp+4) <- stack.(fp+i-
loffset*5);
                                        exec fp (sp+5) (pc+1)

                        | _ -> raise(Failure("Unmatched type!!")))

        | Sfp i ->
                let obj_id = stack.(sp-1) in
                (
                        match obj_id with
                                1 -> stack.(fp+i) <- stack.(sp-1);
stack.(fp+i-1) <- stack.(sp-2); exec fp (sp) (pc+1)

                        |       2 ->
                                        for j=0 to 29 do
                                                stack.(fp+i-j) <-
stack.(sp-j-1)
                                        done;
                                        exec fp (sp) (pc+1)

                        |       3 ->
                                        stack.(fp+i) <- stack.(sp-
1);
                                        stack.(fp+i-1) <- stack.(sp-
2);
                                        stack.(fp+i-2) <- stack.(sp-
3);
                                        stack.(fp+i-3) <- stack.(sp-
4);
                                        stack.(fp+i-4) <- stack.(sp-
5);
                                        exec fp (sp) (pc+1)
                        |       4 ->
                                        stack.(fp+i) <- stack.(sp-
1);
                                        stack.(fp+i-1) <- stack.(sp-
2);
                                        stack.(fp+i-2) <- stack.(sp-
3);
                                        stack.(fp+i-3) <- stack.(sp-
4);
                                        stack.(fp+i-4) <- stack.(sp-
5);
                                        exec fp (sp) (pc+1)
                        |       5 ->
                                        stack.(fp+i) <- stack.(sp-
1);
                                        stack.(fp+i-1) <- stack.(sp-
2);
                                        stack.(fp+i-2) <- stack.(sp-
3);
                                        stack.(fp+i-3) <- stack.(sp-
4);
                                        stack.(fp+i-4) <- stack.(sp-
5);
```

```
                                        exec fp (sp) (pc+1)
                        |       6 ->
                                        stack.(fp+i) <- stack.(sp-
1);
                                        stack.(fp+i-1) <- stack.(sp-
2);
                                        stack.(fp+i-2) <- stack.(sp-
3);
                                        stack.(fp+i-3) <- stack.(sp-
4);
                                        stack.(fp+i-4) <- stack.(sp-
5);
                                        exec fp (sp) (pc+1)

                        | _ -> raise(Failure("Unmatched type!!")))

        | Sfpa i ->
                if (stack.(sp-1) <> 1) then raise(Failure("Array
index type error in Sfpa!!")) else
                let obj_id = stack.(sp-3)
                and loffset = stack.(sp-2) in
                (
                        match obj_id with
                                1 -> stack.(fp+i-2*loffset) <-
stack.(sp-1-2);
                                         stack.(fp+i-1-2*loffset) <-
stack.(sp-2-2);
                                         exec fp (sp) (pc+1)
                        |       2 ->
                                        for j=0 to 29 do
                                                stack.(fp+i-j) <-
stack.(sp-j-1)
                                        done;
                                        exec fp (sp) (pc+1)
                        |       3 ->
                                        stack.(fp+i-5*loffset) <-
stack.(sp-1-2);
                                        stack.(fp+i-1-5*loffset) <-
stack.(sp-2-2);
                                        stack.(fp+i-2-5*loffset) <-
stack.(sp-3-2);
                                        stack.(fp+i-3-5*loffset) <-
stack.(sp-4-2);
                                        stack.(fp+i-4-5*loffset) <-
stack.(sp-5-2);
                                        exec fp (sp) (pc+1)
                        |       4 ->
                                        stack.(fp+i-5*loffset) <-
stack.(sp-1-2);
                                        stack.(fp+i-1-5*loffset) <-
stack.(sp-2-2);
                                        stack.(fp+i-2-5*loffset) <-
stack.(sp-3-2);
                                        stack.(fp+i-3-5*loffset) <-
stack.(sp-4-2);
                                        stack.(fp+i-4-5*loffset) <-
stack.(sp-5-2);
                                        exec fp (sp) (pc+1)
                        |       5 ->
                                        stack.(fp+i-5*loffset) <-
stack.(sp-1-2);
                                        stack.(fp+i-1-5*loffset) <-
```

```
stack.(sp-2-2);
                                        stack.(fp+i-2-5*loffset) <-
stack.(sp-3-2);
                                        stack.(fp+i-3-5*loffset) <-
stack.(sp-4-2);
                                        stack.(fp+i-4-5*loffset) <-
stack.(sp-5-2);
                                        exec fp (sp) (pc+1)
                        |       6 ->
                                        stack.(fp+i-5*loffset) <-
stack.(sp-1-2);
                                        stack.(fp+i-1-5*loffset) <-
stack.(sp-2-2);
                                        stack.(fp+i-2-5*loffset) <-
stack.(sp-3-2);
                                        stack.(fp+i-3-5*loffset) <-
stack.(sp-4-2);
                                        stack.(fp+i-4-5*loffset) <-
stack.(sp-5-2);
                                        exec fp (sp) (pc+1)

                        | _ -> raise(Failure("Unmatched type!!")))

        (*draw shape currently on top of stack*)
        | Jsr(-1) ->
                let obj_id = stack.(sp-1) in
                (
                        match obj_id with
                                1 -> raise (IllegalPrint);
                        |       2 ->
                                        let size = stack.(sp-4) in
                                                let rec make_string
current_string counter = if counter < size
                                                                then
 (make_string (current_string ^ Char.escaped(Char.chr stack.(sp-5-
counter)))(counter+1))
                                                                else
(current_string) in
                                                                        let
this_will_work = (make_string(""))(0)) in
                                                                        Graphics.moveto(stack.
(sp-2))(stack.(sp-3));
                                                                        Graphics.draw_string
(this_will_work);
                                                                        exec
fp sp (pc+1)

                        |       3 ->
                                        Graphics.draw_rect(stack.
(sp-2)) (stack.(sp-3)) (stack.(sp-4)) (stack.(sp-5));
                                        exec fp sp (pc+1)
                        |       4 ->
                                        Graphics.draw_ellipse(stack.
(sp-2)) (stack.(sp-3)) (stack.(sp-4)) (stack.(sp-5));
                                        exec fp sp (pc+1)
                        |       5 ->
                                        Graphics.moveto(stack.(sp-
2)) (stack.(sp-3));
                                          Graphics.lineto(stack.(sp-
4)) (stack.(sp-5));
                                          exec fp sp (pc+1)
                        |       6 -> raise (IllegalPrint)
```

```
                        |   _ -> raise (IllegalPrint))

        (*suspend program*)
        | Jsr(-2) ->
                                let duration = stack.(sp-2) in
                                        Thread.join(Thread.create(Thread.delay)
(float_of_int duration /. 1000.0));
                                        Graphics.clear_graph ();
                                        exec fp (sp) (pc+1)

        (*print int to command line for debug purposes*)
        | Jsr(-4) -> print_endline(string_of_int stack.(sp-2)); exec
fp sp (pc+1)

        (*dump complete stack to command line*)
        | Jsr(-5) -> Array.iter print_endline (Array.map
string_of_int stack);

        | Jsr i -> stack.(sp) <- pc+1; exec fp (sp+1) i (* stores
current pc and execute on i. *)
        | Ent i -> stack.(sp) <- fp; exec sp (sp+i+1) (pc+1)
        | Rts i ->
                let new_fp = stack.(fp) and new_pc = stack.(fp-1)
and base = fp-i-1 in
                        (
                        let obj_id = stack.(sp-1) in
                        match obj_id with
                                1 -> (stack.(base+1) <- stack.(sp-
1);  (* Construct an int on top of stack*)
                                        stack.(base) <- stack.(sp-
2);
                                        exec new_fp (base+2) new_pc
                                        )
                        |       2 ->
                                        let sp_temp = Array.make 30
0 in
                                        for j=0 to 29 do
                                                sp_temp.(j) <-
stack.(sp+j-30)
                                        done;
                                        for j=0 to 29 do
                                                stack.(base+j) <-
sp_temp.(j)
                                        done;
                                        exec new_fp (base+30) new_pc

                        |       3 -> let sp1 = stack.(sp-1)
                                        and sp2 = stack.(sp-2)
                                        and sp3 = stack.(sp-3)
                                        and sp4 = stack.(sp-4)
                                        and sp5 = stack.(sp-5) in
                                        (stack.(base+4) <- sp1;  (*
Construct an int on top of stack*)
                                        stack.(base+3) <- sp2;
                                        stack.(base+2) <- sp3;
                                        stack.(base+1) <- sp4;
                                        stack.(base) <- sp5;
                                        exec new_fp (base+5) new_pc
                                        )
                        |       4 -> let sp1 = stack.(sp-1)
                                        and sp2 = stack.(sp-2)
                                        and sp3 = stack.(sp-3)
```

```
                                        and sp4 = stack.(sp-4)
                                        and sp5 = stack.(sp-5) in
                                        (stack.(base+4) <- sp1;  (*
Construct an int on top of stack*)
                                        stack.(base+3) <- sp2;
                                        stack.(base+2) <- sp3;
                                        stack.(base+1) <- sp4;
                                        stack.(base) <- sp5;
                                        exec new_fp (base+5) new_pc
                                        )
                        |        5 -> let sp1 = stack.(sp-1)
                                        and sp2 = stack.(sp-2)
                                        and sp3 = stack.(sp-3)
                                        and sp4 = stack.(sp-4)
                                        and sp5 = stack.(sp-5) in
                                        (stack.(base+4) <- sp1;  (*
Construct an int on top of stack*)
                                        stack.(base+3) <- sp2;
                                        stack.(base+2) <- sp3;
                                        stack.(base+1) <- sp4;
                                        stack.(base) <- sp5;
                                        exec new_fp (base+5) new_pc
                                        )
                        | _ -> raise(Failure("Unmatched type!!"));
                        );
        | Beq i -> exec fp (sp-1) (pc + if stack.(sp-2) = 0 then i
else 1)
        | Bne i -> exec fp (sp-1)(pc + if stack.(sp-2)!= 0 then i
else 1)
        | Bra i -> exec fp sp (pc+i)
        | Hlt -> ()

        (*funcitonality that differs from micro c*)
        (*create empty graph*)
        | Sgraph -> Graphics.open_graph ""; exec fp (sp) (pc+1)
        (*clear graph*)
        | Egraph -> Graphics.clear_graph (); exec fp (sp) (pc+1)

        (*move object on top of stack by ints in stack above*)
        | Movby ->
                let deltax = stack.(sp-2) in
                 let deltay = stack.(sp-4) in
                  let obj_id = stack.(sp-5) in
                  (
                        match obj_id with
                                1 -> raise (IllegalMove)
                        |        2 ->
                                        stack.(sp-6) <- (stack.(sp-
6) + deltax);
                                        stack.(sp-7) <- (stack.(sp-
7) + deltay);
                                        exec fp sp (pc+1)
                        |        3 ->
                                        stack.(sp-6) <- (stack.(sp-
6) + deltax);
                                        stack.(sp-7) <- (stack.(sp-
7) + deltay);
                                        exec fp sp (pc+1)
                        |        4 ->
                                        stack.(sp-6) <- (stack.(sp-
6) + deltax);
                                        stack.(sp-7) <- (stack.(sp-
```

```
7) + deltay);
                                        exec fp sp (pc+1)
                        |       5 ->
                                        stack.(sp-6) <- (stack.(sp-
6) + deltax);
                                        stack.(sp-7) <- (stack.(sp-
7) + deltay);
                                        stack.(sp-8) <- (stack.(sp-
8) + deltax);
                                        stack.(sp-9) <- (stack.(sp-
9) + deltay);
                                        exec fp sp (pc+1)
                        |       6 ->
                                        stack.(sp-6) <- (stack.(sp-
6) + deltax);
                                        stack.(sp-7) <- (stack.(sp-
7) + deltay);
                                        exec fp sp (pc+1)

                        | _ -> raise(Failure("Unmatched type!!")))

        (*move object on top of stack by ints in stack above*)
        | Movto ->
                let newx = stack.(sp-2) in
                 let newy = stack.(sp-4) in
                  let obj_id = stack.(sp-5) in
                  (
                        match obj_id with
                                1 -> raise (IllegalMove)
                        |       2 ->
                                        stack.(sp-6) <- newx;
                                        stack.(sp-7) <- newy;
                                        exec fp sp (pc+1)
                        |       3 ->
                                        stack.(sp-6) <- newx;
                                        stack.(sp-7) <- newy;
                                        exec fp sp (pc+1)
                        |       4 ->
                                        stack.(sp-6) <- newx;
                                        stack.(sp-7) <- newy;
                                        exec fp sp (pc+1)
                        |       5 ->
                                        let deltax = ( newx - stack.
(sp-6)) in
                                         let deltay = ( newy -
stack.(sp-7)) in
                                                stack.(sp-6) <-
newx;
                                                stack.(sp-7) <-
newy;
                                                stack.(sp-8) <-
(stack.(sp-8) + deltax);
                                                stack.(sp-9) <-
(stack.(sp-9) + deltay);
                                                exec fp sp (pc+1)
                        |       6 ->
                                        stack.(sp-6) <- newx;
                                        stack.(sp-7) <- newy;
                                        exec fp sp (pc+1)
                        | _ -> raise(Failure("Unmatched type!!")))

        (*get coordinate of object on stack -- the type of
```

```
coordinate is specified by the int on top of the object on teh
stack*)
          | GetC ->
                  let coord_type = stack.(sp-2) in
                          let coord = stack.(sp-3-coord_type) in
                                  stack.(sp) <- coord;
                                  stack.(sp+1) <- 1;
                                  exec fp (sp+2) (pc+1)

     | MakeS -> (* Takes the top five integers of stack and put back
a shape. *)
                  let v1 = stack.(sp-1)
                  and v2 = stack.(sp-3)
                  and v3 = stack.(sp-5)
                  and v4 = stack.(sp-7)
                  and v5 = stack.(sp-9)
                  in
                  if ((v1 <> 1) || (v2 <> 1) || (v3 <> 1) || (v4 <> 1)
|| (v5 <> 1)) then
                          raise(Failure("MakeS type check error!"))
else
                  (
                          stack.(sp-1) <- stack.(sp-2);
                          stack.(sp-2) <- stack.(sp-4);
                          stack.(sp-3) <- stack.(sp-6);
                          stack.(sp-4) <- stack.(sp-8);
                          stack.(sp-5) <- stack.(sp-10);
                          exec fp sp (pc+1)
                  )
          | _ -> raise(Failure("Illegal Bytecode Operation!!"))


 (*execute first line*)
 in exec 0 0 0
```

Parser.mly

```
%{ open Ast %}

%token SEMI LPAREN RPAREN LBRACE RBRACE RBRACKET LBRACKET COMMA
%token RTOKEN LTOKEN ETOKEN
%token STRUCT INT CHARS
%token OBJLINE OBJTRIANGLE OBJELLIPSE OBJRECTANGLE OBJSHAPE
%token V1 V2 V3 V4
%token PLUS MINUS TIMES DIVIDE ASSIGN
%token EQ NEQ LT LEQ GT GEQ
%token AND OR NOT LBRACKET RBRACKET
%token MEMBERINDEX SHIFT MOVETO SERIALIZED PARALLELIZED
%token RETURN IF ELSE FOR WHILE
%token MEMBERIN
%token <string>STRING
%token <string>RECTANGLE
%token <string>ELLIPSE
%token <string>LINE
%token <string>SHAPE
%token <int> LITERAL
%token <string> ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left EQ NEQ
```

```
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE

%start program
%type <Ast.program> program

%%

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

fdecl:
   ID LPAREN formals_opt RPAREN LBRACE vdecl_opt stmt_list RBRACE
     { { fname = $1;
         formals = $3;
         locals = List.rev $6;
         body = List.rev $7 } }

formals_opt:
    /* nothing */ { [] }
  | formal_list   { List.rev $1 }

formal_list:
    formal_decl                   { [$1] }
  | formal_list COMMA formal_decl   { $3 :: $1 }

formal_decl:
    CHARS ID                  { { vtype = "string"; name = $2;
vsize = 1} }
   | INT ID                   { { vtype = "int"; name = $2; vsize
= 1} }
   | OBJRECTANGLE ID                       { { vtype = "rect"; name
= $2; vsize = 1} }
   | OBJELLIPSE ID                             { { vtype =
"ellipse"; name = $2; vsize = 1} }
   | OBJLINE ID                                    { { vtype =
"line"; name = $2; vsize = 1} }
   | OBJSHAPE ID                               { { vtype = "shape";
name = $2; vsize = 1} }

vdecl_opt:
    /* nothing */    { [] }
  | vdecl_list { List.rev $1 }

vdecl_list:
  | vdecl                     { [$1] }
  | vdecl_list vdecl         { $2 :: $1 }

vdecl:
    CHARS ID SEMI                { {vtype = "string"; name = $2;
vsize = 1} }
   | INT ID SEMI {         {vtype = "int"; name = $2; vsize = 1} }
   | OBJRECTANGLE ID SEMI { {vtype = "rect"; name = $2; vsize = 1} }
   | OBJELLIPSE ID SEMI   { {vtype = "ellipse"; name = $2; vsize = 1}
}
   | OBJLINE ID SEMI      { {vtype = "line"; name = $2; vsize = 1} }
   | OBJSHAPE ID SEMI     { {vtype = "shape"; name = $2; vsize = 1} }
   | INT ID LBRACKET LITERAL RBRACKET SEMI { { vtype = "arrayi"; name
= $2; vsize = $4} }
   | OBJRECTANGLE ID LBRACKET LITERAL RBRACKET SEMI { { vtype =
```

```
"arrayr"; name = $2; vsize = $4} }
   | OBJELLIPSE ID LBRACKET LITERAL RBRACKET SEMI { { vtype =
"arraye"; name = $2; vsize = $4} }
   | OBJLINE ID LBRACKET LITERAL RBRACKET SEMI { { vtype = "arrayl";
name = $2; vsize = $4} }
   | OBJSHAPE ID LBRACKET LITERAL RBRACKET SEMI { { vtype = "arrays";
name = $2; vsize = $4} }

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

 stmt:
     expr SEMI { Expr($1) }
   | RETURN expr SEMI { Return($2) }
   | 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_opt 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) }
   | STRING           { String($1) }
   | RECTANGLE                { Rectangle($1) }
   | LINE                       { Line($1) }
   | ELLIPSE                  { Ellipse($1) }
   | SHAPE                      { Shape($1) }
   | ID               { Id($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) }
   | ID ASSIGN expr   { Assign($1, $3) }
   | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
   | LPAREN expr RPAREN { $2 }
   | ID SHIFT LBRACE expr expr RBRACE                  {Moveby($1,
$4, $5)}
   | ID MOVETO LBRACE expr expr RBRACE                  {Moveto($1,
$4, $5)}
   | ID MEMBERIN V1   { GetV1($1) }
   | ID MEMBERIN V2   { GetV2($1) }
   | ID MEMBERIN V3   { GetV3($1) }
   | ID MEMBERIN V4   { GetV4($1) }
   | ID LBRACKET expr RBRACKET { Aid($1, $3) }
   | ID LBRACKET expr RBRACKET ASSIGN expr { AAssign($1, $3, $6) }
   | RTOKEN LBRACE expr expr expr expr RBRACE  { REvaluation($3, $4,
$5, $6) }
   | ETOKEN LBRACE expr expr expr expr RBRACE  { EEvaluation($3, $4,
$5, $6) }
```

```
   | LTOKEN LBRACE expr expr expr expr RBRACE  { LEvaluation($3, $4,
$5, $6) }

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

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

Scanner.mll

```
 { open Parser }

 let letter = ['a'-'z' 'A'-'Z']
 let digit = ['0' - '9']
 let character = ['`' '~' ' ' '!' '@' '#' '$' '%' '^' '&' '*' '(' ')'
'-' '_' '=' '+' '[' '{' ']' '}' '|' ';' ':' ''' ',' '<' '.' '>' '/'
'?']
 let whitespace = ['\t' ' ' '\r' '\n']
 let id = letter (letter | digit | '_' )*
  (* Attention '\' are not included here. *)

 rule token = parse
   whitespace { token lexbuf }
 | "(:" { comment lexbuf }
 | "v1" { V1 }
 | "v2" { V2 }
 | "v3" { V3 }
 | "v4" { V4 }
 | "int" { INT }
 | "string" { CHARS }
 | "line" { OBJLINE }
 | "triangle" { OBJTRIANGLE }
 | "ellipse" { OBJELLIPSE }
 | "rectangle" { OBJRECTANGLE }
 | "shape" { OBJSHAPE }
 | "else" { ELSE }
 | "while" { WHILE }
 | "for" { FOR }
 | "if" { IF }
 | "return" { RETURN }
 | id as lit { ID(lit)}
 | "S:{"whitespace*(digit)+whitespace+(digit+)whitespace+'"'(letter |
digit | character )*'"'whitespace*'}' as lit { STRING(lit) }
 | "R:{"whitespace*(digit)+whitespace+(digit)+whitespace+
(digit)+whitespace+(digit)+whitespace*'}' as lit { RECTANGLE(lit) }
 | "E:{"whitespace*(digit)+whitespace+(digit)+whitespace+
(digit)+whitespace+(digit)+whitespace*'}' as lit { ELLIPSE(lit) }
 | "L:{"whitespace*(digit)+whitespace+(digit)+whitespace+
(digit)+whitespace+(digit)+whitespace*'}' as lit { LINE(lit) }
 | '-'?(digit)+ as lit { LITERAL(int_of_string lit) }
 | "R:" { RTOKEN }
 | "E:" { ETOKEN }
 | "L:" { LTOKEN }
 | "==" { EQ }
 | "!=" { NEQ }
 | "<=" { LEQ }
 | ">=" { GEQ }
 | '>' { GT }
 | '<' { LT }
 | '!' { NOT }
 | '(' { LPAREN }
```

```
 | ')' { RPAREN }
 | '{' { LBRACE }
 | '}' { RBRACE }
 | '[' { LBRACKET }
 | ']' { RBRACKET }
 | '.' { MEMBERIN }
 | "=>" { SHIFT }
 | "->" { MOVETO }
 | ';' { SEMI }
 | '=' { ASSIGN }
 | ',' { COMMA }
 | '+' { PLUS }
 | '-' { MINUS }
 | '*' { TIMES }
 | '/' { DIVIDE }
 | eof { EOF }

 and comment =
         parse ":)" { token lexbuf }
         | _ {comment lexbuf}
```

tbc.ml

```
 type action = Bytecode | Compile

 let _ =
         let action = if Array.length Sys.argv > 1 then
                 List.assoc Sys.argv.(1) [ ("-b", Bytecode); ("-c",
Compile)]
         else Compile in
         let lexbuf = Lexing.from_channel stdin in
         let program = Parser.program Scanner.token lexbuf in
         match action with
           | Bytecode -> let listing =
                 Bytecode.string_of_prog (Compile.translate program)
                 in print_endline listing
         | Compile -> ignore(Execute.execute_prog (Compile.translate
program))
```

Makefile:

```
 OBJS = ast.cmo parser.cmo scanner.cmo compile.cmo bytecode.cmo
execute.cmo tbc.cmo

 LIBS=$(WITHGRAPHICS) $(WITHUNIX) $(WITHTHREADS) $(WITHSTR)

 CONF=-I +threads

 # Should be set to -custom if you use any of the libraries above
 # or if any C code have to be linked with your program
 # (irrelevant for ocamlopt)

 # CUSTOM=-custom

 # Default setting of the WITH* variables. Should be changed if your
 # local libraries are not found by the compiler.
 WITHGRAPHICS =graphics.cma -cclib -lgraphics -cclib -L/usr/X11R6/lib
-cclib -lX11

 WITHUNIX =unix.cma -cclib -lunix

 WITHSTR =str.cma -cclib -lstr

 WITHNUMS =nums.cma -cclib -lnums

 WITHTHREADS =threads.cma -cclib -lthreads
```

```
WITHDBM =dbm.cma -cclib -lmldbm -cclib -lndbm

tbc : $(OBJS)
        ocamlc $(CONF) -o tbc $(LIBS) $(OBJS)

scanner.ml : scanner.mll
        ocamllex scanner.mll

parser.ml parser.mli : parser.mly
        ocamlyacc parser.mly

%.cmo : %.ml
        ocamlc $(CONF) -c $<

%.cmi : %.mli
        ocamlc -c $<

.PHONY : clean
clean :
        rm -f tbc parser.ml parser.mli scanner.ml testall.log \
        *.cmo *.cmi *.out *.diff

# Generated by ocamldep *.ml *.mli
ast.cmo:
ast.cmx:
bytecode.cmo: ast.cmo
bytecode.cmx: ast.cmx
compile.cmo: bytecode.cmo ast.cmo
compile.cmx: bytecode.cmx ast.cmx
execute.cmo: bytecode.cmo ast.cmo
execute.cmx: bytecode.cmx ast.cmx
tbc.cmo: scanner.cmo parser.cmi execute.cmo \
    bytecode.cmo ast.cmo
tbc.cmx: scanner.cmx parser.cmx execute.cmx \
    bytecode.cmx ast.cmx
parser.cmo: ast.cmo parser.cmi
parser.cmx: ast.cmx parser.cmi
scanner.cmo: parser.cmi
scanner.cmx: parser.cmx
parser.cmi: ast.cmo
```