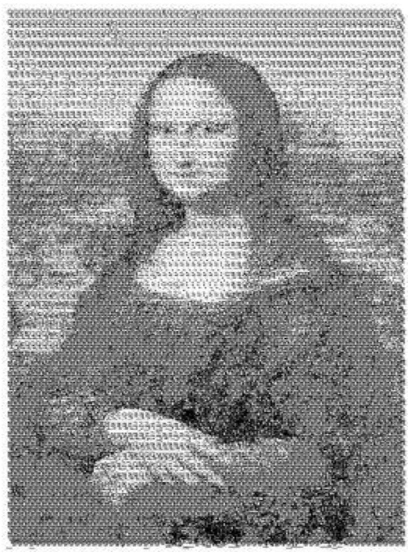
EZ-ASCII:
A Language for ASCII-Art Manipulation



Dmitriy Gromov (dg2720)

Joe Lee (jyl2157)

Yilei Wang (yw2493)

Xin Ye (xy2190)

Feifei Zhong (fz2185)

December 19th, 2012

Table of Contents

1 Introduction

1.1 Motivation

Many times when using text files or command-line interfaces, a user may have the need to display an image using ASCII characters, whether for aesthetic (decoration) or functional (diagram) purposes. Generating small, simple geometric shapes, let alone more complicated diagrams, is a tedious process for the user. For example, let us consider a simple use case where the user would like to display a square with some elements around it in a defined drawing space. If the square ever needs to be moved or resized, the user must not only re-draw the square, but also maintain or update the elements that are affected by the move. This is a non-trivial manual process and is not sustainable for larger, more complicated diagrams. Similarly, translating an image file (.bmp, .jpg, .png…) into a corresponding 2D array of ASCII characters is non-trivial.

We propose to solve this problem by implementing a programming language (EZ-ASCII) for the purpose of creating and manipulating ASCII Art, a form of drawing pictures using only the characters defined by the ASCII character set. The goals of EZ-ASCII are to 1) abstract away from the user the task of mapping/converting characters to intensities (and vice versa), 2) allow easy manipulation of the image (selection, replacement, movement, masking) with the use of a “canvas” built-in type, and 3) provide flexible features with simple syntax which allows developers to easily build on top of the built-in library functions.

1.2 Background

While drawing ASCII art, one generally only has the option of varying color intensity. Color intensity in this case is defined by how much of the space allotted for a character is actually used by the character. For example, ‘@’ would describe an intense color, while ‘,’ would describe a non-intense color. The language will provide some default mapping of characters and intensities, so that the user can work with intensities instead of spending time determining which ASCII characters to use. However, if a user feels the need to use a specific character, they will have the option of overwriting the default mapping for the color intensity. Since many characters have similar intensities, we will not provide a mapping of all characters to intensities by default. If desired, the user will be able to increase the granularity of intensities as needed.

For image manipulation, the user will have the complete freedom to change any character in that image or change all characters of a given intensity to a different one. This way, an image's color can be "inverted" or simply darkened to conform to the user's tastes. Furthermore, the user will have the ability to apply simple transformations to a range of characters (e.g. shift several cells up and to the left). One of the main uses for this programing language will naturally be to convert images from other formats into an ASCII format. Moreover, given the various abilities available to the programmer, he/she should quickly be able to set up functionality to draw various shapes and constructs and manually create images.

2 Language Tutorial

The EZ-ASCII language was primarily designed to provide simple mechanisms for creating and manipulating ASCII Art images, but may also stand as a simple general-purpose programming language. The following tutorial will give a quick tour of setting up a development environment for EZ-ASCII and creating a simple program.

2.1 Getting Started

2.1.1 Compiler Requirements

EZ-ASCII requires the following to be installed:

1. OCaml (4.00.0) - http://caml.inria.fr/ocaml/
2. Python (2.7.3) - http://www.python.org/
3. PIL (Python Imaging Library) - http://www.pythonware.com/products/pil/
    a. For Windows, a Windows installer is available from the PIL homepage.
    b. For Mac OS, download *Imaging-1.1.7.tar.gz* file from PIL website, and extract it. Use the following commands to build and install:

    ```
    $cd Imaging-1.1.7
    $python setup.py build
    $sudo python setup.py install
    ```

    c. For Linux users, use the following command to install PIL on Linux:

    ```
    $sudo apt-get install python-imaging
    ```

2.1.2 How to Compile and Run a Program

1. Extract the EZ-ASCII compiler source files into a directory.
2. Run `make` to build the executable **`ezac`**.
3. The **`ezac`** executable takes a `.eza` source file as input, and allows some command-line parameters. A usage example is `ezac [options] <source-file>`. The following options are supported:

Flag	Description
`-a`	generates the abstract syntax tree (AST) for the input program and outputs it in string format
`-s`	runs static semantic analysis on the source program and outputs any errors found
`-i`	runs the interpreter on the source program
`-b`	compiles the source file into bytecode and outputs the bytecode in string

	format
`-c`	compiles the source file into bytecode and executes it
`-cd`	performs the same operation as with the `-c` flag, but runs in debug mode (outputs debug statements)

Note that for development purposes, any output (e.g. `-a`, `-s`, `-b`, `-cd`) is currently set to stdout, giving the developer flexibility in piping it to any output stream. If a developer were to run the compiler with the –c option in an actual release of the compiler, it should output the bytecode to a file, which would in turn be passed as input to a separate bytecode-interpreter executable. This is trivial to implement, and for efficiency purposes we have kept everything in a single executable with stdout as the default output stream.

2.1.3 A First EZ-ASCII Program

A simple "Hello, world!" program could look like the following:

```
d <- "Hello, world!";  // store string in variable d
d -> out;              // output d to stdout
```

In this simple two-line example, we first store (by use of the `<-` assignment operator) the `"Hello, world!"` string in a variable `d`. Note that there is no explicit variable declaration; the assignment statement implicitly handles the creation of the variable `d` with the correct type (string). The output operator (`->`) here uses the keyword `out` which signifies to output to standard out. We will see later that the output operator may output to a file, and in addition, a second boolean `render` parameter may be specified in the case of outputting canvases.

Note the C-style line-comments (`//`), and each line delimited with a semicolon.

2.2 Variables and Arithmetic Expressions

2.2.1 Types

EZ-ASCII has four variable types: `boolean`, `integer`, `string`, and `canvas`. A boolean type may be either `true` or `false`. Integers are in the allowed range that OCaml supports (31 bits on 32-bit processors, 63 bits on 64-bit processors). Strings are sequences of characters, and are represented by surrounding double quotes. A canvas is represented internally as record with a data field which holds the 2D array of integer intensities, and a granularity field which stores the granularity level of the canvas image.

As noted above, there is no explicit type declaration, so even after a variable is assigned a type, any subsequent assignment may alter its type.

```
i <- ~(3 > 2);     // variable i assigned boolean false
```

```
i <- 2;           // variable i now assigned integer 2
i <- “hello”;     // variable i now assigned string “hello”
i <- load(“lena.jpg”, 10); // Use built-in function load which
                           // loads an image file with granularity
                           // value and stores it as a canvas
```

2.2.2 Operators

EZ-ASCII supports a variety of operators. Some example expressions are as follow (see the reference manual section for a complete listing of operators):

```
i <- -(1 + 1);                           // i assigned -2
i <- 2 * 2;                              // i assigned 4
i <- 5 / 2;                              // i assigned 2
i <- 7 % 3;                              // i assigned 1
i <- 1 * 3 + 2;                          // i assigned 5
i <- 5 – 1;                              // i assigned 4
i <- “hello “ + “world!”;                // i assigned “hello world!”
img3 <- img1[2:8, 1:4] + img2[,];  // img3 assigned additive layering
img4 <- img1[,] – img3;            // img4 assigned difference layering
i <- ~(1 > 2);                           // i assigned true
i <- (1>2 || 2>1);                       // i assigned true
i <- (1>2 && 2>1);                       // i assigned false
```

2.2.3 Selection

The selection operator can be applied to a `canvas` type variable. It returns an equal size canvas with the selected points value, and the rest of points are blank. A complete list of examples is as follows:

```
img1[1,1]   //select a single point value at coordinate (1,1) of img
img1[1, 1:4]      //select a horizon slice in row 1 from column 1 to 4
img1[1:4, 1]      //select a vertical slice in column 1 from row 1 to 4
img1[1:4, 2:5] //select a range of row from 1 to 4 and a range of
column from 2 to 5
img1[,]     //select a copy of canvas img1
```

2.3 Branch Statement

The following is an example of an `if-else` conditional statement (note that the else-block is optional):

```
if (2 < 3) {
  e <- 3;
  e -> out;
  z <- "hello world!";
  z -> out;
}
else {
   f <- 8;
   f -> out;
}
```

The output is:

```
3
hello world!
```

2.4 Loop Statement

The following is an example of a `for` loop in EZ-ASCII:

```
for i <- 0 | i < 5 | i <- i + 1 {
  i -> out;
}
```

The output is:

```
0
1
2
3
4
```

2.5 Functions

The following is an example of recursive function in EZ-ASCII:

```
Fun factorial(x) {
   if(x <= 1) {
      return 1;
   }
   else
      return factorial(x - 1) * x;
}

a <- factorial(6);
a -> out;
```

The output is:

```
720
```

2.6 Built-in Canvas Functions

There are three built-in functions of EZ-ASCII relating to canvas operations - `load`, `blank`, and `shift`. The following is a list of examples using these functions.

```
canvas <- load(“lena.jpg”, 10);     // load image “lena.jpg” and
                                    // normalize it with a granularity
                                    // of 10
canvas <- blank(10, 10, 8);    // output an empty canvas with size 10 *
                               // 10 and a granularity of 8
img <- shift(img2, SHIFT_UP, 6);    //shift img2 up with 6 spaces
```

2.7 Examples

2.7.1 A Simple ACSII Art file

```
canvas <- load(“bot.jpg”, 10);
sel1 <- canvas[10:20, 30:40]; // All points in that range
                              //(10:20, 30:40) padded by -1
sel2 <- canvas[<3];           // All points with intensity less
                              // than 3 padded by -1
canvas -> out, true;
sel2 -> out, true;
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@k@@@@@@@@@@@@@@@@@@@@***@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@k*@@@@@@@@@@@@@@@@@@@kk@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@*k@@@**kZZZZZZkk**@@@k*@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@k**kkZZZZZZZZZZZZk*k*@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@kZZZZZZZZZZZZZZZZZZZ*@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@*ZZZZZZZZZZZZZZZZZZZZk*@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@kZZZZZZZZZZZZZZZZZZZZkZZ*@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@kZZZZ*@*ZZZZZZZZZZZk@*ZZZZk@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@*ZZZZZ*@*ZZZZZZZZZZZk@@kZZZZ*@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@*kZZZZZk*kZZZZZZZZZZZk**kZZZZZ@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@kZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ*@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@*ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@*ZZkZZZZZZZZZZZZZZZZZZZZZZZZZZZZ*@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@kZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZk@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@kZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZk@@@@@@@@@@@@@@
@@@@@@@@@@@**@@@*ZZZkZZZZZZZZZZZZZZZZZZZZZZZZZZZZk@@**kk*@@@@@@@
@@@@@@@@@*kZZZk@@*********************************@*kZZZk*@@@@@@
@@@@@@@@@ZZZZZZ*@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZkk@kZZZZZk*@@@@@
@@@@@@@@*ZZZkZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@kZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZk@ZZZZZZZ*@@@@@
@@@@@@@@*ZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZkZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZk@kZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@ZZZZZZZ*@@@@@
@@@@@@@@kkZZZZZZ@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZk@kZZZZZZ*@@@@@
@@@@@@@@*ZZZZZZk@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZk@kZZZZZk*@@@@@
@@@@@@@@@kZZZtk@@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZk@*kZZZk*@@@@@@
@@@@@@@@@*kkZk*@@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZk@@*kkk*@@@@@@@
@@@@@@@@@@@@@@@@@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@ZZZZZZZZZZZZZkZZZZZZZZZZZZZZZZZZk@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@kZZZZZZZZZZZtZZZZZZZkZZZZZZZZZZZk@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@kkZZZZZZZZZZZkZZZZZZZZZZZZZkZZk*@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@*****ZZZZZZZ*******kZZZZZZZ****@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@ZZZZZZZ@@@@@@@*kZZZZZk@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@ZZZZZZZ@@@@@@@*ZZZZZZk@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@kZZZZZZ@@@@@@@*ZZZZZZk@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@ZZZZZZZ@@@@@@@*ZZZZZZk@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@ZZZZZZZ@@@@@@@@ZZZZZZk@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@ZZkZZZZ@@@@@@@@ZZZZZZZ@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@ZZZZZZZ@@@@@@@@ZZZZZZk@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@ZZZZZZZ@@@@@@@@ZZZZZZk@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@kZZZZZk@@@@@@@@ZZZZZZk@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@*ZZZZk@@@@@@@@@kZZZZZ*@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@kZZZk@@@@@@@@@*kZZkk@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@**k@@@@@@@@@@@**k*@@@@@@@@@@@@@@@@@@@@@




                                 ZZZZZZ
```

```
                   zzzzzzzzzzzz
               zzzzzzzzzzzzzzzzzzz
              zzzzzzzzzzzzzzzzzzzz
             zzzzzzzzzzzzzzzzzzzz zz
            zzzz   zzzzzzzzzzz   zzzz
           zzzzz   zzzzzzzzzzz    zzzz
           zzzzz   zzzzzzzzzzz    zzzzz
          zzzzzzzzzzzzzzzzzzzzzzzzzzzzz
         zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
         zz zzzzzzzzzzzzzzzzzzzzzzzzzzzz
         zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
         zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
        zzz zzzzzzzzzzzzzzzzzzzzzzzzzzzz
  zzz                                       zzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz    zzzzz
zzz zz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz  zzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz   zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzz zzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz    zzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz zzzzzzz
 zzzzzz zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz    zzzzzz
zzzzzz  zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz    zzzzz
         zzzt   zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz    zzz
   z    zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
        zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
        zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
        zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
        zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
        zzzzzzzzzzzzz zzzzzzzzzzzzzzzzzz
         zzzzzzzzzzztzzzzzzz zzzzzzzzzzz
          zzzzzzzzzzz zzzzzzzzzzzzzz zz
              zzzzzzz        zzzzzzz
              zzzzzzz         zzzzz
              zzzzzzz        zzzzzz
               zzzzzz        zzzzzz
              zzzzzzz        zzzzzz
              zzzzzzz        zzzzzz
              zz zzzz        zzzzzzz
              zzzzzzz        zzzzzz
              zzzzzzz        zzzzzz
               zzzzz         zzzzzz
               zzzzz          zzzzz
                zzz            zz
```

2.7.2 Draw New ACSII Art

```
canv <- blank(10,10,10);
for i <- 0 | i < 9 | i <- i + 1
{
      canv[i,i] <- 9;
}
canv -> out, true;
```

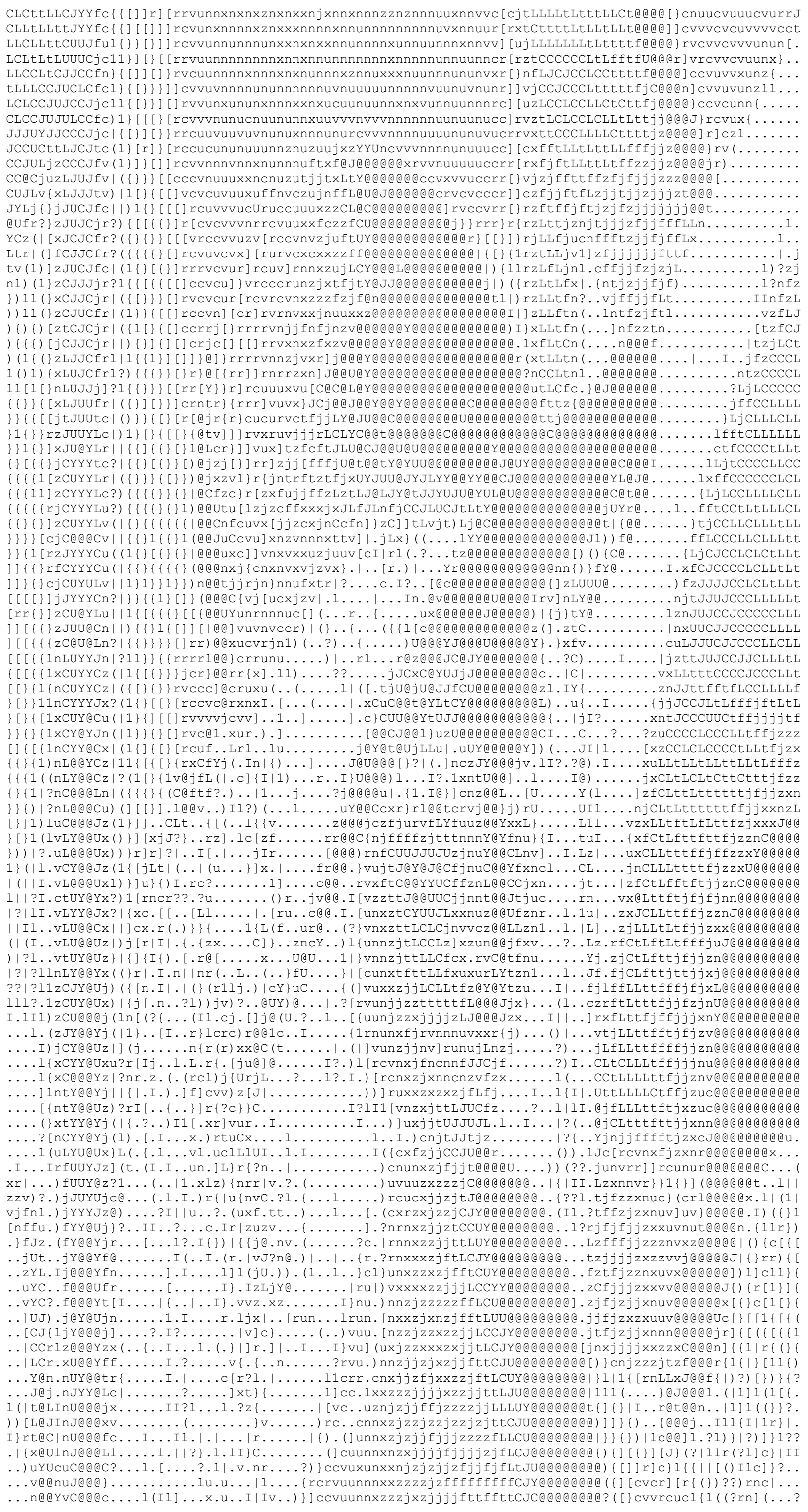
```
@.........
.@........
..@.......
...@......
....@.....
.....@....
......@...
.......@..
........@.
..........
```

2.7.3 More Examples

```
// Filter Extremes
// This is sort of like contrast raising.
```

```
include "../demo/demo_lib.eza";

intensity <- 60;

c <- load_image("../image/lena.jpeg", 60);
c -> out, true;

c[<15] <- 0;
c[>40] <- c$g - 1;

nothing <- "";
nothing -> out;
c -> out, true;
```

```
CLCttLLCJYYfc{{[]]r][rrvunnxnxnxznxnxxnjxnnxnnnzznznnnuuxnnvvc[cjtLLLLtLtttLLCtQkba[}cnuucvuuucvurrJ
CLLtLLttJYYfc{{][[]]]rcvunxnnnnxznxxxnnnnxnnunnnnnnnnnuvxnnuur[rxtCttttLtLLtLLtOkbh]]cvvvcvcuvvvvcct
LLCLLttCUUJfu1{}}[}]]rcvvunnunnnunxxnxnxxnnunnnnxunnuunnnxnnvv][ujLLLLLLLtLttttfabdZ}rvcvvcvvvunun[>
LCLtLtLUUUCjc11}][}[[rrvuunnnnnnnnxnnxnnunnnnnnnnxnnnnnunnuunncr[rztCCCCCCLtLfftfUhdbr]vrcvvcvuunx}+<
LLCCLtCJJCCfn}{[]][]}rvcuunnnnxnnnxnxnunnnxznnuxxxnuunnnununvxr[}nfLJCJCCLCCttttfobdO]ccvuvvxunz{<>>
tLLLCCJUCLCfc1}{[}}}]]cvvuvnnnnununnnnnnuvnuuuuunnnnvuunuvnunr]]vjCCJCCCLtttttfjChdkn]cvvuvunz1l<<>
LCLCCJUJCCJjc11{[}}][]rvvunxununxnnnxxnxucuunuunnxnxvunnuunnnrc][uzLCCLCCLLCtCttfjQbpa}ccvcunn{<<>>>
CLCCJUJULCCfc)1}[[[}[rcvvvnunucnuununnxuuvvvnvvvnnnnnnnuunuunucc]rvztLCLCCLCLLtLttjjhpdJ}rcvux{++l>>i
JJJUYJJCCCJjc|{[}][}}rrcuuvuuvuvnunuxnnnunurcvvvnnnuuuunuvucrrvxttCCCLLLLCttttjzObph]r]cz1+<+i><<
JCCUCttLJCJtc(1}[r]}[rccucununuuunnznuzuujxzYYUncvvvnnnnunuuucc][cxfftLLtLttLLfffjjzhddQ}rv(<<<!l!>>
CCJULjzCCCJfv(1}]}[]]rcvvnnnvnnxnunnnuftxfXJ0Z0X00xrvvnuuuuuccrr[rxfjftLLttLtffzzjjzXbppjr)<<<>>i><<
CCXCjuzLJUJfv|({}}}[[cccvnuuuxxncnuzutjjtxLtYOOOO00occvxvvuccrr[}vjzjfftffzfjfjjjzzzhpqb[<<>>i!_><+
CUJLv{xLJJJtv)|1[}{[[]vcvcuvuuxuffnvczujnffLZUOJXZ00QOcrvcvcccr]]czfjjftfLzjjtjjzjjjztbpp<+_i>l>>i++
JYLj{}jJUCJfc||)1{}[[[]rcuvvvucUruccuuuxzzCLZCXXQQO0QQo]rvccvrr[}rzftffjftjzjfzjjjjjjjopt_+>!!i>!<>!
XUfr?}zJUJCjr?){[[{{}]r[cvcvvvnrrcvuuxxfczzfCU0OXoQoQa0oj}}rrr}r{rzLttjznjtjjjzfjjfffLLn<+><!i>>>>l-
YCz(|[xJCJCfr?({}{}}[[[vrccvvuzv[rccvnvzjuftUYZX0QOQoO0Okbr}[[}]}rjLLfjucnfffftzjjfjffLx<+_>i<<!><l+>
Ltr|(]fCJJCfr?({{{{}[]rcvuvcvx][rurvcxcxxzzffXQZXoQXZbkbbdp|{[}{1rztLLjv1]zfjjjjjjfttf+>_i<!+i<<i|<j
tv(1)]zJUCJfc|(1{}[{}]rrrvcvur]rcuv]rnnxzujLCYXX0Lobhahhbkda|){11rzLfLjnl!cffjjfzjzjL>++!>><!><l)?zj
n1)(1}zCJJJjr?1{{[{{[[]ccvcu]}vrcccrunzjxtfjtYZJJaoahahahbbpj|)({rzLtLfx|<{ntjzjjfjf)_<>i><i>>il?nfz
})11(}xCJJCjr|({[}}}[]rvcvcur[rcvrcvnxzzzfzjfXnOaaaQQQhoQahkptl|)rzLLtfn?_lvjffjjfLt<+<i><+!<!IInfzL
))11(}zCJUCfr|(1}}{[[]rccvn][cr]rvrnvxxjnuuxxzoQ0oooOaahhhhhkdQI|]zLLftn(>+1ntfzjftl_+ii!<!<>i!vzfLJ
){){)[ztCJCjr|({1[}{[]ccrrj[}rrrrvnjjfnfjnzv0ZaOOOYQahhhakhhkbd)I}xLLtfn(<+>]nfzztn<<<!>!>>i><[tzfCJ
){{{)[jCJJCjr||)}}]{][]crjc[][[]rrvxnxzfxzvZQQO0Yoa0ohQhkOahhkdqi1xfLtCn(<<-inkhZf<>ii>!iii>>|tzjLCt
)(1{(}zLJJCfrl|1{{1}][]]}b]}rrrrvnnzjvxr]jXaZYOOOQQoQQoQhkahhkkdr(xtLLtn(<+-Qk0kpk+<ii|ii>IiijfzCCCL
1()1){xLUJCfrl?){{{}}[}r}Q[{rr]]rnrrzxn]JZXUQYZX000oQXooaoohaakbp?nCCLtnl__kZkdbdm++>!>i<>i<ntzCCCCL
11[1[}nLUJJj]?l{{}}}[[rr[Y}}r]rcuuuxvu[CXCXLZY00ZZOXQoQaaoaaQohhkQutLCfci}dJbkhhkq++!<<>+!<?LjLCCCCC
{{}}{[xLJUUfr|({}][}}]crntr}{rrr]vuvx}JCjZXJXQY0ZYXQX0oQoaCaooaaakkfttz{h0Zkhaohdp+>>>>>!>_jffCCLLLL
}}{{[[jtJUUtc|()}}{[}[r[0jr{r}cucurvctfjjLYXJU0ZCZXO0X00aUOQQQaaQQhttjQkookhaoakdd<><!<+>+}LjCLLLCLL
}1{}}rzJUUYLc|)1}[}{[[}{dtv]]rvxruvjjjrLCLYC0Ot0XZO0O0CZOZ00QZQ0OQXCaOZkaoaohakdd<>>>>i!lfftCLLLLLL
}}1{}]xJUXYLr||{{]{{}[}1hLcr}]]vux]tzfcftJLUZCJZXUXUOQZOX0O0OYOXOXZaOoahaooaaaQhdd+>i><>>ctfCCCCtLLt
{}[{{}jCYYYtc?|{{}[{}}[)hjzj[}]rr]zjj[fffjUZtZXtYZYUUXXXOZQXZXJZUYQOohaQooohhCZapI_<!+iilLjtCCCCLLCC
{{{{1[zCUYYLr|({}}}{[}})Qjxzv1}r{jntrftztfjxUYJUUXJYJLYYXOYYXXCJaZokQ0QQQoohYLZJaii>><>lxffCCCCCCLCL
{{{11]zCYYYLc?){{{{}}{}|0Cfzc}r[zxfujjffzLztLJXLJYZtJJYUJUXYUL0U0aa0QQoooaahCZtXh_>i+>i{LjLCCLLLLCLL
{{{{{rjCYYYLu?){{{{}{}1)O0Utu[1zjzcffxxxjxJLfJLnfjCCJLUCJtLtYZOaoZZQ0oaaaakjUYrQ+_i!l>!fftCCtLtLLLCL
{{}{}]zCUYYLv(|{}{{{{{{|h0Cnfcuvx[jjzcxjnCcfn]}zC]]tLvjt)LjQCaoQQQQooahaahht|{Xd_>><i>}tjCCLLCLLLtLL
}}}}[cjCXXXCv||{{}1{{}1(bOJuCcvu]xnzvnnnxttv]|ijLx}((l!>_lYYao0o00QaoaahhJ1))fb<+>!>>>ffLCCCLLCLLLtt
}}{1[rzJYYYCu((1{}[{}{}|k0Zuxc]]vnxvxxuzjuuv[cI|rl(l?>_~tzOho0QaQQQoakh[)(){Col+>>>>+{LjCJCCLCLCtLLt
]]{{}rfCYYYCu(|{{}{{{{}(doOnxj{cnxnvxvjzvx}l|li[r!)|>+-YrOOZQ0Z0OQQoQnn{)}fY0+>>i<>I>xfCJCCCCLCLLtLt
]]}{}cjCUYULv||1}1}}1}})nhhtjjrjn}nnufxtr|?i>++c>I?>+[Xco0XOZ0O00QoZ{]zLUUUQ<+ii><><)fzJJJJCCLCLtLLt
[[[[}]jJYYYCn?|}}{{1}[]}(koXC{vj[ucxjzv|illi!_|!!!In>ZvoQXZXXU0OOQIrv]nLYXQ!_<i>>>!>njtJJUJCCCLLLLLt
[rr{}]zCUXYLu||1{[{{{}[[{doUYunrnnnuc[](l!+r~i{lil_-uxo0OXZXJXO0XO)|{j}tYoi>+>!!<>>lznJUJCCJCCCCCLLL
]][{{}zJUUXCn||){{}1{[]][|aO]vuvnvccr)|(}!>{>+>({{l[c0OOXXXXXZO0oQz(]lztC><+ii!i<i<|nxUUCJJCCCCCLLLL
][[{{{zCXUXLn?|{{}}}}[]rr)baxucvrjn1)(!i?)ii{+<<+l)UXX0YJXXXUooooY}>}xfv<+>!i!i+<icuLJJUCJJCCCLLCLL
[[[{{1nLUYYJn|?11}}{{rrrr1bb}crrunui!l+i)|+_r1_-<rZzZ0OJCZJYOQaoaaO{+i?C)<<i!Il<><|jzttJUJCCJJCLLLtL
{[[{{1xCUYYCz(|1{[{}}}jcr}hXrr{x]ll1)+l!>??!l+_-jJCxC0YUJjJOQQoahhaci+|C|>i!!l>>!<vxLLtttCCCCJCCCLLt
[[}{1{nCUYYCz|({[}{}}rvccc]ocruxu(>_(<<_!ll|([~tjUXjUQJJfCU0Qohahkhzl-IY{>ll!l>>>!znJJttfftfLCCLLLLf
}[}}11nCYYYJx?(1{}[[}[rccvcZrxnxIi[>+>(!<>i|-xCuCZ0tZYLtCYXZQoaakkkL)<<u{iiI!!>i+{jjJCCJLtLfffjftLtL
}[}{[1xCUYXCu(|1}{][[]rvvvvjcvv]i-1+l]i<_>!]+c}CUUZXYtUJJXXXQQQaakkX{+_|jI?!li>><xntJCCCUUCtffjjjjtf
}}{}{1xCYXYJn(|1}}{}[]rvcal_xur<)_]lii+<<++<!{ZXCJQOl}uzUZXXOQQQahaCI<_>Ci>!?l><?zuCCCCLCCCLLtffjzzz
[]{[{1nCYYXCx|(1{]{[}[rcufi+Lr1l<lu-l<+-_l!-j0YatOUjLLu|luUYXQOQQY])(<+!JI|l>><i[xzCCLCLCCCCtLLtfjzx
{{}{1)nLXXYCz|11{[{[}{rxCfYj(!In|{)>>+]<i+-JXUoXO[}?|(<]nczJY0Q0jv!lI?_?X)!Ii>lixuLLtLLtLLttLLtLfffz
{{{1((nLYXXCz|?(1[}{1vbjfL(|_c]{I|1)!__r++I}U0oQ)l>><I?<1xntUah]!>li!>+IO)!!>!iijxCLtLCLtCttCtttjfzz
{}{1|?nCXXXLn|({{{}{(CZftf?_)><|1i<<j+<<i?jaZ0ou|i{1+Io}]cnzXbL>+[Ui<<+lY(li!><]zfCLttLtttttjfjjzxn
}}{)|?nLXZZCu)(][[}]!lZdv>_)Il?)(i<_l><++!uYQoCcxr}rlaatcrvjXd}j)rU>>++iUI1iii<njCLtLttttttffjjxxnzL
[}]1)luCXXXJz(1}]]+!CLt<_{[(+<l{{v<+!><__zOOajczfjurvfLYfuuzZpYxxL}i!>><L1l>+!vzxLLtftLfLttfzjxxxJZO
}[}1(lvLYZOUx()}][xjJ?}iirz]ilc[zf+<<><-rrX0C{njffffzjtttnnnYqYfnu}{I_>ituI+>_{xfCtLfttfttfjzznCZ000
}))|?luLXXZUx))}r]r]?|<<I[>|_l>jIr>>>i><[X0o)rnfCUUJJUJUzjnuYpOCLnv]l+I>Lz|>><uxCLLtttffjffzzxYQQ000
1}(|llvCYXZJz(1{[jLt|(i>|(u_++}]x+|+<><frOh<}vujtJXYXJXCfjnuCbQYfxncl<><CLi<!<jnCLLLttttfjzzxUQQQ00Q
|(||IivLXXXUx1)}]u}{)Ilrc?<<<>><_1]<<i<ch0+irvxftCXZYYUCffznLpoCCjxni<!ijtil+|zfCtLfftftjjznCQQQQQQo
l||?I!ctUYXYx?)1[rncr??i?uli<+<i<()r_-jvQZ+I[vzzttJXXUUCjjnnthkJtjucli!+rn_l>vxXLttftjfjfjnnQQQQQohh
|?|lI>vLYYXJx?|{xc+[[li[Ll>l+_>|+[ru-_cok-I![unxztCYUUJLxxnuzodUfznr+il_1u|!_zxJCLLttffjzznJQQ0Qahha
||Ili>vLUXXCx||]cxir(>)}}{-<<~1{L(f<~urh+<(?}vnxzttLCLCjnvvczXqLLzn1_>l+|L]+lzjLLLtLtfjjzxxOQ0Qahhhh
(|(I!+vLUXXUz|)j[r|I|!{<{zx>l_lC]}><zncY_<)l{unnzjtLCCLz]xzunZwjfxv>>!?!<Lz_rfCtLftLtfffjuJoQQahaahh
)|?ll<vtUYXUz}|{]{I{)-[irX[>!+>ix_+-UXUi_+1|}vnnzjttLLCfcx!rvChtfnu<<li<+Yj+zjCtLfttjfjjznOQQoaahhhh
|?|?llnLYXXYx((}r|lIln||nr(~>L+-(>!}fUi_<>}|[cunxtfttLLfxuxurLYtzn1!<!l++Jf>fjCLfttjttjjxj0Qohhhhkhh
??|?l1zCJYXUj)({[n>I|!|(}(r1ljl)|cY}uC>!li{(]vuxxzjjLCLLtfzZYQYtzu><<I|><fjlffLLttfffjfjxLQQahhhhhhh
lll?l1zCUYZUx)|{j[!nl>?l))jv)?++XUY)o-<>|i?[rvunjjzzttttttfLaZoJjx}><!(ll+czrftLtttfjjfzjnUQQhhhhhhhk
IllIl)zCUXZXj(ln[(?{!l!(I1lcj~[]jQ(U+?><l>>[{uunjzzxjjjjzLJboQJzx+<<I||!+]rxfLttfjffjjjxnYoahhhhhkkk
lill!(zJYXZYj(|1}l<[I>!r}lcrc)rQk1c+<I<il!>{1rnunxfjrvnnnuvxxr{j)<i>()|i+lvtjLLttfftjfjzvXakhkhhkkkk
>illI)jCYXXUz|](jli>liln{r(r)xxpC(t<>><!l|i(|]vunzjjnv]runujLnzj<><!!?)ii>jLfLLttffffjjzn0hkkkkkkkk
i>!ll{xCYYZUxu?r[Ij_<llL<r{i[juh]X>ii>!>I?i)l[rcvnxjfncnnfJJCjfl<l<!<?)Ii>CLtCLLLtffjjjnuQbkkkkkkhh
>i!il{xCXXZYz|?nriz>(i(rc1)j{UrjLl!!?!i!l?iIl)[rcnxzjxnncnzvfzx++>>l<l(!l<CCtLLLLttfjjznvabkkkbkhhah
<i>>]1ntYXZYj||{|>I>)l]f]cvv)z[J|>i!li!>l!>!l))]ruxxzxzxzjfLfj<>>iI>+l{I|iUttLLLLCtffjzuchdbbbbhhhkk
>>i>[{ntYXXUz)?rI[!_{ll}]r{?c}}C>i>iiii<i><I?lI1[vnzxjttLJUCfzi>il?><l|lIiXjfLLLttftjxzuchbbkkkkkhhh
<i<>(}xtYYXYj(|{l?>+)I1[lxr]vur!!Iil!ii!!!!!Il>>)]uxjjtUJJUJLili>I><i|?(<lXjCLtttfttjjxnnddkkkbkaaaa
<>>!?[nCYYXYj(l)<[lI!i>xl)rtuCxl>>>l>!>ii<!il!li>Il)cnjtJJtjz+!i!!!>l|?{i!YjnjjffffftjzxcJdbkkkhaZu>
>><<l(uLYUXUx}L(+{!l!<+vliuclLlUI+<l<I<i+><l!iI({cxfzjjCCJUXOr>>i!!i!())llJc[rcvnxfjzxnrQdhhhhhOx>+>
lI>_!IrfUUYJz](t+(IlI!lun<]L}r{?n>i|>l<<liill!l)cnunxzjfjjtXQahUl!i>))(??ljunvrr]]rcunurokhhha0C!<!(
xr|il>)fUUYaz?1l_<(-i|1ixlz){nrr|v>?<(<<!>>!l!>)uvuuzxzzzjCZQooahZ__|{|II>Lzxnnvr}}1{}](Qhahhat<>l||
zzv)?>)jJUYUjcQ+++(~llI-)r{|u{nvC>?l<{>>>l>i!!<)rcucxjjzjtJO00Qoahh<i{??l!tjfzzxnuc}(crlQhkkox<l|(1|
vjfn1>)jYYYJzo)>>>!?I||u+!?l(uxf-tt><)>!!l>il{>(cxrzxjzzjCJYOO0oakkk>(Il!?tffzjzxnuv]uv}abkhX>I)({}1
[nffu>)fYYXUj}?+>IIi>?><<clIr|zuzvi><{l>>liil]>?nrnxzjjztCCUYZOQohhbd><l?rjfjfjjzxxuvnutkkbanl{11r})
+}fJz!(fYXXYjr!+<[li<l?_I{})|{{jX<nv_(li>li>?c!|rnnxzzjjttLUYZO0oahkd0<ilLzfffjjzzznvxzZdbkQ|(){c[{[
-ijUtlljYXXYfh<<l>i<>I(<>Il(rl|vJ?n0+)|>>|>i{ri?rnxxxzjftLCJYXZOoahkdd+>-tzjjjjzxzzvvjXabbhJ|{}rr){[
-izYLiIjXXZYfn+_!+<>]iI+>-!l]1(jU>))_(1l>l<i}cl}unxzzxzjfftCUYZO0ohkbdb+~fztfjzznxuvxoakbbQ])1]c11}{
-_uYC!>fXXZUfr>l>++-[i_>i+>I}lIzLjYX-li>>i>|ru|)vxxxxzzjjjLCCYYO0Qhhkbp~~zCfjjjzxxvvOdkkbkJ{){r[1}]{
--vYC?>fXXXYt[I<>i!|{!>|>+I}>vvzlxz_ii+i<<I}nul)nnzjzzzzzffLCUXZ0Qahbkd]-zjfjzjjxnuvQbabbox[{}c[1[}{
~-]UJ)!jXYXUjn+>+i<>1lIi>+_r_ljx|-+[run!!>lrun![nxxzjxnzjfftLUUXOQoahbdh+jjfjzxzxuuvakadkUc[}[[1{[{(
<_[CJ{ljYXZXj]<+++?<I?_+<<+><|v]c}+-l>l(>>)vuu>[nzzjzzxzzjjLCCJYOQokhkkbljtfjzjjxnnnXaodhjr]{[({[{{1
<_|CCrlzXXXYzx(++{>+I>!i1i(!}|]r!]|-!I><<I}vu](uxjzzxxxzxjjtLCJYZQQhhkkb[jnxjjjjxxzzxCadon]{{1|r{(){
_-|LCrlxUXXYff!_>!+>Ii?!>+><v{>{<ln<+!<_>?rvul)nnzjjzjxzjjfttCJUX0Qahkbk0[)}cnjzzzjtzfhbXr{1{|}[11{)
_-iYXn!nUYXXtr{+!<>iI!|>_+>c[r?l<|li><+<l1crr!cnxjjzfjxxzzjftLCUY00ahkkbd|}l|1{[rnLLxJdhf{|)?)[})}{?
<+>JOj!nJYYXLc|<_+!+il?l+!<!]xt}{+!>>_<i1]cc_1xxzzzjjjjxzzjjttLJUX0ohhkbd|111(li>i}QJQbZ1!(|1]1(1[{l
l(|tOLInUXXXjx>i><<<II?l<>i1>?z{l>>++_+|[vciluznjzjjffjzzzzjjLLLUYZQohkkbt{]{}|I++rhthknl!|l]1((}}?l
))[LZJInJXXZxvi<<>>>><(><+!!<!l}v>+++<!)rc>icnnxzjzzjzzjzzjzjttCJUO0ahhkkk)]]}{)l>{o00jiiIl1{I|1r}|l
I}rtZC|nUXZXfc+<>I>i<I1l|+|li>|ri><+-+|{)-(]unnxzjzjjfjjjzzzzfLLCUXOQohkkb|}}{})|1ch0]l!?l)}|?)]}1??
<|{xXU1nJXZXL1>>ll>1>||?}>l>1I}C><l<>>>>(]cuunnxnzxjjjjfjjjjzjfLCJXZ0oahkb{){][{}][J}(?|l1r(?l]c}|II
>>)uYUcuCZXXC?>l<l![<!!i?-1|!vlnr<<+i?)}ccvuxunxxnjzjzjjzfjjfjfLtJUO0Qohkb0){[]]r]c}1{{||[()I1c]}?i!
i>lvoonuJXOZ}l<i<!l>+!>>lu<u-l>|l><il{rcrvuunnnxxzzzzjzffffffffCJYXZOoahhh({][cvcr][r{()}??)rnc|!!i
>i!nooYvCXXXcii!<l(Il]!l_x+u<lI|Iv-l)}]ccvuunnxzzzjxzjjjjfttfftCJCXZOQahhh?([}cvvrcuc1{1((?rn](!>!?
```



3 Language Reference Manual

3.1 Program Definition

The structure of an EZ-ASCII program source file consists of expression statements and functions. A `main()` function may be optionally specified to denote the main entry point of the program.

```
< global statements >
< function declarations >
fun main() {
   <main program code>
}
```

3.2 Lexical Conventions

3.2.1 Tokens

There are six types of tokens: identifiers, keywords, constants, string literals, operators, and other separators. Blanks, horizontal, and vertical tabs, newlines, formfeeds, and comments as described below (collectively, “white space”) are ignored except as they separate tokens. Some white space is required to separate otherwise adjacent identifiers, keywords, and constants.

3.2.2 Comments

When a `//` symbol is encountered, the `//` symbol and the rest of the line is considered a comment and is ignored by the compiler.

```
// This is a comment line
img[x1, y1] <- 1; // This is another comment
```

3.2.3 Identifiers

An identifier is a sequence of letters and digits. The first character must be a letter; the underscore _ counts as a letter. Upper and lower case letters are different.

3.2.4 Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

Table 3.1 Keywords

`blank`	`load`	`SHIFT_LEFT`
`else`	`main`	`SHIFT_RIGHT`
`false`	`map`	`SHIFT_UP`
`Fun`	`out`	`true`
`for`	`return`	
`if`	`shift`	
`include`	`SHIFT_DOWN`	

3.2.5 Constants

There are four types of constants in EZ-ASCII and they are listed as follows:

- Boolean Constants: A boolean constant is either `true` or `false` (case-sensitive).
- Integer Constants: An integer constant consists of a sequence of digits.

  ```
  i <- 213
  ```

  There are 4 labeled integer constants to define shifting directions
  `SHIFT_UP, SHIFT_LEFT, SHIFT_DOWN, SHIFT_RIGHT`
  Their values are 0, 1, 2, 3 respectively.

- String Constants**:** A string constant consists of a sequence of characters enclosed in double quotes `“”`. The following characters may be used with escape sequences:

Table 3.2 Characters For Escape Sequence

Character	**Escape Sequence**
newline	\n
horizontal tab	\t
single quote	\’

double quote	\”
backslash	\\

- Mapping Constants: An intensity mapping consists of a table mapping intensities to characters. A custom mapping can be defined using the keyword **map**:

```
map <- {I0:”C0”, I1:”C1”,...,IN:”CN”}
```

where each I is an intensity, and the corresponding C is the character mapped to that intensity. Any reference to the intensity mapping will refer to the most recent assignment of `MAP` or the default if none has been assigned.

The default mapping is a map of all printable ASCII characters ordered in ascending order based on how many pixels each character takes up in each character space.

3.2.6 Granularity and Intensity

A mapping must have at least two values and the granularity must be at least 2. The minimum intensity will be the least intense item in the map and the maximum will be the most intense. For intensities between `1` and `n - 1` where `n` is the size of the mapping the distance between each intensity is as close to even as possible. The formula for this is defined as follows:

```
diff = (n - 2) / ((g - 2) + 1)
```

where *n* is again the size of the map, and *g* is the granularity.

3.3 Meaning of Identifiers

Identifiers may refer to objects (locations in storage) or functions. A function and an object may not be referred to using the same identifier – the following is a syntax error:

```
foo <- 3
fun foo() {
      <function-body>
}
```

3.3.1 Types

There are four types:

- Boolean: A boolean stores one bit of information and may have the value `true` or `false`.

- Integer: Integers can store 32-bits of data and are signed.
- String: Strings are sequences of characters, and are bounded only by available memory.
- Canvas: A canvas is the primary storage type in EZ-ASCII. All of the image modification happens on this type. Internally, it is represented as a two-dimensional array of integers referred to as intensities. This canvas can be loaded from an existing image file or it can be created manually. Additionally, a canvas has the following readable attributes: width and height in number of characters, and granularity.

  There are two methods of creating a canvas in EZ-ASCII. The first is to load an existing image using the `load` built-in function, and the second is to use the `blank` built-in function (see built-in functions). In the case of loading an external image file, a custom intensity mapping may be specified to specify the granularity of the image, or the default will be used. Various operations may be performed on canvases, including selection, movement, and masking.

3.4 Expressions

3.4.1 Unary Minus Operator

The operand of the unary `-` operator must have arithmetic type, and the result is the negative of its operand.

```
i <- -(1 + 4) // i assigned -5
```

3.4.2 Multiplicative Operators

The multiplicative operators `*`, `/`, and `%` group left-to-right and require their operands to be of the same primitive types.

If the operands are of integer type, then the result of the `*` operator is the product of the operands. The result of the `/` operator is the quotient of the operands and the result of the `%` is the remainder after integer division on the operands. The `/` operator results in an integer (fractions truncated).

If the operands are of any other type, a syntax error will occur.

```
i <- 2 * 3   // i assigned 6
i <- 3 / 2   // i assigned 1
i <- 2 / 3   // i assigned 0
i <- 3 % 6   // i assigned 0
i <- 5 % 3   // i assigned 2
```

3.4.3 Additive Operators

The additive operators + and − group left-to-right and require their operands to be of the same primitive types. The grammar is as follows:
If the operands are of integer type, then the result of the + operator is the sum of the operands, and the − operator is the difference of the operands.

If the operands are of string type, then the result of the + operator is the concatenation of the operands, and the − operator will result in a syntax error.

If the operands are of type canvas, then the result of the + operator is a new canvas where each intensity is the result of adding the two corresponding intensities from the operand canvases, truncated to the maximum mapped intensity. The result of the − operator is a new canvas where each intensity is the result of the difference between the two corresponding intensities from the operand canvases, truncated to the minimum intensity of 0.

If the operands are of boolean type, a syntax error will occur.

```
i <- 1 + 2 * 3 + 4                // i assigned 11
j <- 5 - 3                        // j assigned 2
k <- "hello " + "world!"          // k assigned "hello world!"
k <- "hello " - "world!"          // syntax error
img3 <- img1[3:8, 2:4] + img2[,]  // img3 assigned additive layering
img4 <- img1[,] - img3            // img4 assigned difference layering
m <- k + img4                     // syntax error
n <- true + img4                  // syntax error
```

3.4.4 Relational Operators

The relational operators group left to right, i.e. `a < b < c` is parsed as `(a < b) < c`. The operators < (less), > (greater), <= (less than or equal), and >= (greater than or equal) all yield a boolean `true` or `false`. The two variables on either side of a relational operator must be of the same type.

3.4.5 Logical Negation Operator

The operand of the ~ operator must have boolean type, and the result is true if the value of its operand compares equal to `false`, and `false` otherwise.

```
b <- ~(3 > 2) // b assigned false
```

3.4.6 Equality Operators

The = (equal to) and ~= (not equal to) operators are analogous to the relational operators except for their lower precedence. For example, `a<b = c<d` is parsed as `(a<b) = (c<d)` and evaluates to `true` if `a<b` and `c<d` have the same truth-value.

3.4.7 Logical AND Operator

The `&&` operator groups left-to-right, returning `true` if both its operands compare unequal to `false`, and `false` otherwise. Both operands must be of boolean type, except in the case of boolean expressions used in a selection operator, in which case both operands must be of a boolean expression type that satisfies the selection operator (see selection operator).

3.4.8 Logical OR Operator

The `||` operator groups left-to-right, returning `true` if either of its operands compares unequal to `false`, and `false` otherwise. Both operands must be of boolean type, except in the case of boolean expressions used in a selection operator, in which case both operands must be of a boolean expression type that satisfies the selection operator (see selection operator).

3.4.9 Comma Operator

A pair of expressions separated by a comma "`,`" is evaluated left to right.

3.4.10 Selection Operator

The selection operator `[]` denotes a selection on the canvas that it is applied to. When the selection operator is used on a canvas, the return value is a canvas of equal size, which contains only the points of interest (rest are blank). There are multiple types of selections possible depending on different integer parameters for the selection operator, as follows:

- Selection of a single point

  identifier[x, y] – x and y are integer types which denote the x and y coordinates of a single point.
- Selection of rectangles/slices

  identifier[x1:x2, y1:y2] – x1:x2 denotes a range of rows (inclusive), and y1:y2 denotes a range of columns (inclusive).
  identifier[x, y1:y2] – A horizontal slice in row x from columns y1 to y2 (inclusive).
  identifier[x1:x2, y] – A vertical slice in column y from rows x1 to x2 (inclusive).
  identifier[,] – Returns a new copy of the canvas (all rows and columns).

- Selection by boolean expression

  identifier[boolean expression] – selects elements with intensity that satisfy the boolean expression. Boolean expressions for the selection operator must be of the format *[condition][intensity]*, where *[condition]* may be either a relational or equality operator (`<, >, <=, >=, ~=, =`), and *[intensity]* is an integer value. Boolean expressions may be chained by a logical AND operator (`&&`) or logical OR operator (`||`).

3.4.11 Mask Operator

The `&` operator groups left-to-right, operating on canvas types. It returns a new canvas where at any given position, the intensity is 0 if the corresponding intensity in the second operand is 0, and the intensity is the corresponding intensity in the first operand if the corresponding intensity in the second operand is greater than 0. In other words, it returns the first canvas operand, but where the corresponding areas in the second canvas are 0, the corresponding areas in the first canvas are "masked" out. Any operand type other than a canvas type is a syntax error.

```
img3 <- img1 & img2  // img3 is img1 with img2 applied as a mask
i <- 2 & 3           // syntax error
```

3.4.12 Arrow Operator

There are two arrow operators <- (left) and -> (right), which are used for assignment and output, respectively.

• Assignment

The <- left arrow operator assigns the value of the expression to its right to the variable to its left. If the variable is undefined, it is created. If the variable is already in memory, its contents are overwritten with the new value.

identifier `<-` *`expression`*

Examples:
```
canvas <- load("pic.jpg", 10); // the variable canvas holds image data
canvas <- 2;                   // the variable canvas holds an integer
for i <- 2 | i < 10 | i <- i + 1 {
...
}
```

• Output

The -> right arrow operator outputs the value of the variable to its left to either a file specified by a filepath string to its right, or to standard output, specified by the keyword out. The left-hand-side variable must have been assigned previously, otherwise a compiler error will result.

```
d <- "output string";
d -> out;              // outputs "output string" to standard out
```

If the left operand is a canvas, an intensity map may be optionally supplied to dynamically change the intensity mapping.

```
canvas -> out, render;          // outputs image canvas to standard out
canvas(map) -> "test2.txt", render; // outputs image canvas to file
                                    // with new mapping
```

`render` must be a boolean value that specifies whether or not the intensities should be converted to their corresponding characters before being printed. If render is false and it is printed to a file, the file is post pended with the extension .i. If render is true, then the file will map will be applied and the actual image will be printed.

3.4.13 Canvas Attribute Accessor (read-only)

The `$` operator may be appended to a canvas identifier along with one of [`w, h, g`] for width, height, and granularity, respectively, to read the attribute of interest from an existing canvas object as follows:

```
canvas <- load("test.jpg", map)
canvas$w -> [width-integer]
canvas$h -> [height-integer]
canvas$g -> [granularity-integer]
```

3.4.14 Function Calls

A function call moves program execution to the target function. The syntax of a function call is:

function-name (*identifier-list*$_{opt}$)
where *identifier-list* is defined as:
    identifier
    identifier-list , identifier

A function must be declared before the function call.

3.4.15 Include

The `include` keyword allows you to add functionality from another EZ-ASCII code file to the one you are currently working on and has the following syntax:

```
include [filepath];
```

filepath must be the location of another EZ-ASCII file. At compilation time the code included in the desired file will be copied into the file being compiled. Note that identically named global variables or function names in both files will cause compilation errors.

3.5 Declarations

3.5.1 Function Declarations

A function is declared as:

Fun function-name (*identifier-list*$_{opt}$) { <*function-body*> }
where *identifier-list* is defined as:

identifier
identifier-list , *identifier*

Functions act as blocks of code that can be called when desired. Functions can be optionally passed a list of input parameters which are passed by value, and the parameters will be copies of the inputs for the function body. Functions can also optionally return some value at the end of their execution. A function may also call itself recursively in its body.

```
fun foo(img) {
      img[>3] <- 5;
      img[4:8, 3:6] <- 6;
      return img;
}

// recursive factorial
fun factorial(x) {
      if(x = 1) return 1;
      else return x * factorial(x - 1);
}
```

3.5.2 Variable Declarations

Variable declarations are declared as:

Variable-name <– *expression*

Type declarations are not required - variable types are inferred from the declaration. A variable may be set to a different value with a different type even if previously declared, e.g. the following will not result in an error:

```
i <- 3;                    // i holds 3
i <- load("test.jpg", map); // i now holds a canvas
```

3.6 Statements

Except as described, statements in EZ-ASCII are executed in sequence. Statements are executed for their effect and do not have return values. They fall into several groups.

Statement:

assignment-statement

output-statement

conditional-statement

for-statement

selection-statement

return-statement

include-statement

3.6.1 Assignment/Output Statements

Most statements in EZ-ASCII are assignment/output statements. The former is any assignment operation, and the latter is any output operation.

```
img(map) -> "test2.txt";
img[x1, y1] <- 1;
img2 <- shift(img, SHIFT_UP, 5);
```

3.6.2 Conditional Statement

Conditional statements allow for one of several flows of control. An `if` statement may be used with or without an `else` clause. The grammar is as follows:

`if` (*expression*) *statement*
`if` (expression) *statement* `else` *statement*

The expression in the `if` statement must be of boolean type, and if it evaluates to `true`, the first sub-statement is executed. In the second form, the second sub-statement is executed if the expression evaluates to `false`.

```
if(1 > 0) "true case" -> out; // "true case" is output to standard out

if(true) {
...
}

if(3 > 4) "three is greater than four" -> out;
else "the world is sane" -> out;
```

The `else` ambiguity is resolved by connecting an `else` with the last encountered `else`-less `if` at the same block nesting level.

```
if(2 ~= 2)
if(3 > 2)
else "this else binds to the second if" -> out;
```

3.6.3 For Statement

The `for` statement specifies looping.

`for` *expression*$_{opt}$ | *expression*$_{opt}$ | *expression*$_{opt}$ *statement*

In the `for` statement, the first expression is evaluated once, and thus specifies initialization for the loop. There is no restriction on its type. The second expression must be a boolean expression; it is evaluated before each iteration, and if it becomes `false`, the `for` is terminated. The third expression is evaluated after each iteration, and thus specifies a re-initialization for the loop. There is no restriction on its type. Any of the three expressions may be dropped. A missing second expression makes the implied test equivalent to testing a `true` constant.

```
for i <- 2 | i < 10 | i <- i + 1
{
      img[i, i] <- 3;
}
```

3.6.4 Return Statement

`return` *expression*$_{opt}$;

A function returns to its caller by the `return` statement. When return is followed by an expression, the value is returned to the caller of the function. A function without a return statement returns the integer 0 by default.

```
fun a(x) {
      return x + 1;
}

// boo has no explicit return value
fun boo() {
      <body>
}
```

3.7 Scope and Linkage

3.7.1 Lexical Scope

- Variable Scope

Variables declared outside of functions have global scope and are accessible from anywhere in the program. Within a function, scoping rules proceed in the following manner. First, when execution enters a function, all of the function parameters are allocated as local variables for the duration of the function. Second, if a variable does not match a formal parameter, it is checked with the list of globally allocated variables, and may be modified/read. Finally, if a variable does not match any function parameter or global, it is allocated as a new local variable for the duration of the function call. This implies that new globals cannot be created from within functions, though they can be modified.

```
i <- 1;

fun foo(p) {
      p -> out; // p is a local with value passed in as a parameter
      i -> out; // i refers to the global i, outputs 1
      a <- 4;   // a is a new local
      return a;
}
i <- foo(i);    // global i is overridden with the return value of foo
i -> out;       // outputs 4
```

- Function Scope

Functions have global scope. A function may not be referred to unless it has been previously declared.

```
Fun foo() {
      <body>
}
runfunc <- a();
runfunc <- b(); // error – b undefined
```

3.8 System Functions

3.8.1 Blank

`blank` ([width], [height], [granularity])

`Blank` takes three integer input parameters (width and height in number of characters, and a granularity level), and outputs an empty canvas with attributes set accordingly. An empty canvas in EZ-ASCII is one such that all of the intensities are 0.

3.8.2 Load

```
load ( [filepath], [granularity] )
```

`load` takes a string filepath to an existing image file and an integer granularity level as inputs. . If the file name contains the extension .i, it is assumed to be an EZ-ASCII intensity file. In this case, it will load the image directly without any processing. This case will throw an error if the intensities found in file are not compatible with the granularity specified. Otherwise, it uses the Python Imaging Library to convert the image into an EZ-ASCII intensity file. PIL is used to convert the image into 8-bit gray scale and shrink it down to at most 100 by 100 pixels. The 8-bit gray scale values are then normalized according to the granularity and those values written to a file in EZ-ASCII intensity format. This new file is then read in into a canvas which is returned. The side effect of creating a new file is useful because it takes up at most 20K and can be used to quickly reload the canvas on bigger images.

3.8.3 Shift

```
shift( [canvas], [shift_dir], [dist] )
```

`shift` takes a valid canvas identifier canvas, a integer value `shift_dir` representing which direction to shift in and a distance to shift, `dist.` For simplicity, `shift_dir` will accept only 4 possible values: `SHIFT_UP`, `SHIFT_DOWN`, `SHIFT_LEFT`, `SHIFT_RIGHT`. These values are described in section 2.5.2. The purpose of shift is to take all of the characters in on a canvas and shift them in the shift_dir direction, dist spaces.

The direction itself is implied by the names of the variables (e.g. `SHIFT_LEFT` means left). Dist must be greater than 0 and less than the width of the canvas if shifting left or right or less than the height if shifting up or down. The result of this function will be to return a representation of the canvas with everything shifted.

Please note, if the movement causes a character to go beyond the border of the canvas it will disappear. Shifting one column past the right edge will cause the right most edge to disappear and the second to right most column will take its place. Furthermore in this case, the left most column will be padded with intensities of -1. The analogous situation is true for all other shifting directions.

4 Project Plan

4.1 Project Process

4.1.1 Proposal phase

During the early phase of the project, the team met on every Monday after class for two hours to brainstorm ideas for implementing a new language. This continued until the proposal was drafted.

4.1.2 LRM phase

Once receiving positive feedback from TA on Oct. 3rd, the team started working on the actual details of our language, specifying syntax/symantics and grammar rules. The syntax of the language was built primarily around making canvas manipulations simple and intuitive for the end-user. We also began working on the basic structure of the scanner and parser to investigate any potential ambiguities of our language. During this time we also set up a Git repository to track our source code.

4.1.3 Code Design and Implementation

We initially concentrated our efforts in getting most of our language implemented in the scanner, parser, and ast before moving forward. Once we had most of the language parsing working correctly, we started working on a basic interpreter that supported our entire language except for the canvas functionality in order to demonstrate a proof of concept. During this time, we created a top-level file along with a Makefile to trivialize the build process, and also started creating a suite of unit tests. Once we felt confident that the interpreter was working well, we started working on the compiler, bytecode, and bytecode-executor. When the compiler was mostly finished, we added a static semantic analysis module which interfaced between the parser and the compiler to check for compile-time errors.

4.2 Programming Style Guide

We adhered to the following general and OCaml coding conventions:

1. Descriptive comments occur immediately before the block of code they are related to.

2. Scoping is indicated by single tab of 4 spaces (every let block should be properly indented, and nested let blocks should be additionally indented accordingly).

3. Global variables are used sparingly, if at all (functional style).

4. Similar code should be separated into separate modules for modularity and scalability

purposes. Similarly, place any generic code in separate modules as a re-use library.

5. Each module should have a block commented header describing the module's purpose.

6. Significant functions should be accompanied by a detailed comment as to the function's purpose, inputs, outputs, and any other information which would be helpful for other developers.

7. Side effects should be used sparingly, and only when the benefit of the side effect outweighs the alternative. In the case that the use of the side effect may confuse other developers, a descriptive comment should be added.

8. Since OCaml is a functional language, prefer functional programming style over iterative-style programming, when possible.

9. Variables and functions should have intuitive names when possible for maximal clarity, e.g. a variable that stores accumulated data within a recursive function should have something like "acc" or "accum" somewhere in its name. The names "i, j, k" are typically reserved for loop counters.

10. Unless it is absolutely obvious, when accessing functions and types from other modules, prepend the module name – e.g. when pattern matching against a type from the Ast module, the pattern should be `Ast.*` (where * is the Ast type of interest).

11. When pattern matching, always have a catch all case of some form so that the end-user does not see an OCaml matching error. This ensures that in the case that a developer extends the pattern but forgets to add the new patterns to the match list, the catch-all case code is executed.

12. Define descriptive custom exceptions to help other developers during debugging.

13. Wrap any debug code with an appropriate debug flag – remove it once a unit test provides coverage.

4.3 Project Timeline

Table 4.1 Project Timeline

Date	Milestone
09-26-2012	Language proposal submitted.
10-25-2012	Basic scanner, parser, interpreter started to test LRM.
10-31-2012	The language reference manual complete.
11-08-2012	Basic AST implemented

11-26-2012	Test harness added. Interpreter/Ast/Scanner/Parser completed for language (no canvas functionality yet). Start work on Compiler/Bytecode/Execute, and canvas functionality.
12-12-2012	Compiler/Bytecode/Execute complete. Added simple Preprocessor. Canvas load functionality implemented via python script utilizing PIL library. Static semantic analysis module started. Final report work started.
12-15-2012	Canvas functionality and static semantic analysis module completed.
12-17-2012	Project Final Report completed.

4.4 Member Roles and Responsibilities

Table 4.2 Member Roles

Team Member	Code Responsibility	Documentation Responsibilities
Dmitriy (leader)	AST, Parser, Canvas functionality, Preprocessor, Compiler, Bytecode execution, unit testing	Project Proposal, LRM, Final Report (sections 1-8)
Feifei	Scanner, Parser	Project Proposal, LRM, Final Report(sections 2, 6, 7)
Yilei	AST, Scanner, Parser	Project Proposal, LRM, Final Report (sections 1, 3, 4, 5, 7)
Xin	AST, Scanner, Parser	Project Proposal, LRM, Final Report (sections 4, 7)
Joe	AST, Scanner, Parser, Static Semantic Analysis, Compiler, Bytecode execution, unit testing	Project Proposal, LRM, Final Report (sections 1-8)

4.4 Development Tools and Environment

The EZ-ASCII compiler source files were written primarily in OCaml (v4.00.0), with the exception of a python script used to load image files with PIL (Python Image Library). We used `make` to build our executable, and a bash script to run our unit tests and output the test run summaries. Team members used various text editors of choice, from Vim (with omlet.vim OCaml plugin) to Sublime 2 and Notepad++.

We used Git version control to manage our project source, and took advantage of free Github hosting.

4.5 Project Log

Table 4.3 Project Log

Date	Work Done
09/10/2012	Team formed, first meeting, brainstormed ideas for project language.
09/17/2012	Second meeting, solidifying idea of ASCII-art manipulation language
09/24/2012	Third meeting, project proposal drafted, Git version control setup
09/28/2012	Project proposal submitted.
10/08/2012	Fourth meeting, discuss language implementation details.
10/15/2012	Fifth meeting, finalize language details.
10/22/2012	Sixth meeting, divide up portions of LRM.
10/25/2012	Basic scanner, parser, and interpreter started.
10/29/2012	Seventh meeting, final version of LRM submitted.
11/2/2012	Roadmapped work ahead, divided up portions of scanner, parser, and AST. Start image manipulation implementation investigation.
11/8/2012	Basic AST completed.
11/9/2012	Began integrating AST and parser. Image manipulation implementation investigation continues.
11/26/2012	Scanner, Parser, Ast, Interpreter, and top-level Ezac modules working except for Canvas functionality. Test harness added for running unit tests.
12/04/2012	Started adding image supporting features and Preprocessor.
12/12/2012	Compiler, Bytecode, Execute modules finished. Canvas load function implemented with Python script calling into PIL library functions. Final report started.
12/15/2012	Semantic analysis module added, rest of canvas functions finished.
12/16/2012	New development stopped on compiler apart from final bug fixes and unit testing. final report and presentation slides finished
12/17/2012	Final report and presentation slides completed.

5 Architectural Design

This section presents the compiling and executing process of EZ-ASCII program.

5.1 Compiling and Executing

EZ-ASCII compiler takes EZ-ASCII source code files (*.eza) as input. The first step is a preprocessing step where the compiler recursively finds all `include` statements and prepends all dependent modules' source code into one accumulated source program. The accumulated source program is then given to the scanner, which outputs a stream of tokens if successful. The parser then runs on the tokens and constructs them into an abstract syntax tree (AST) according to production rules. The static semantic analyzer then takes in the AST and runs through the entire program, stepping through function calls, in order to find compile-time errors such as type errors, undefined variable/function errors, invalid canvas selection operators, etc. If any errors occur, the errors are output and compilation stops. Otherwise, the compiler performs a second pass through the Ast program to generate the bytecode. Finally, if compilation is successful, the execute module takes the bytecode program and executes it using a stack. The stack-type is of integer, which can be literal values or addresses. If an address is encountered with a local/global load/store bytecode, a lookup/update is performed on a hashtable, which stores all non-integer types (booleans, strings, canvases). The compilation and execution process is illustrated in Figure 5.1.

The following is a list of our source modules and contributors:

Makefile	Joe, Dmitriy
runtests.sh, unit testing	Joe, Dmitriy
scanner.mll	Joe, Dmitriy, Xin, Feifei, Yilei
parser.mly	Joe, Dmitriy, Xin, Feifei
ast.ml	Joe, Dmitriy, Yilei, Xin
interpreter.ml	Joe
compiler.ml	Joe, Dmitriy
execute.ml	Joe, Dmitriy
bytecode.ml	Joe, Dmitriy
ezac.ml (top-level)	Joe, Dmitriy
canvas.ml	Dmitriy
load_img.py	Dmitriy
preprocess.ml	Dmitriy
sast.ml	Joe
hashtypes.ml	Joe, Dmitriy
reuse.ml	Joe

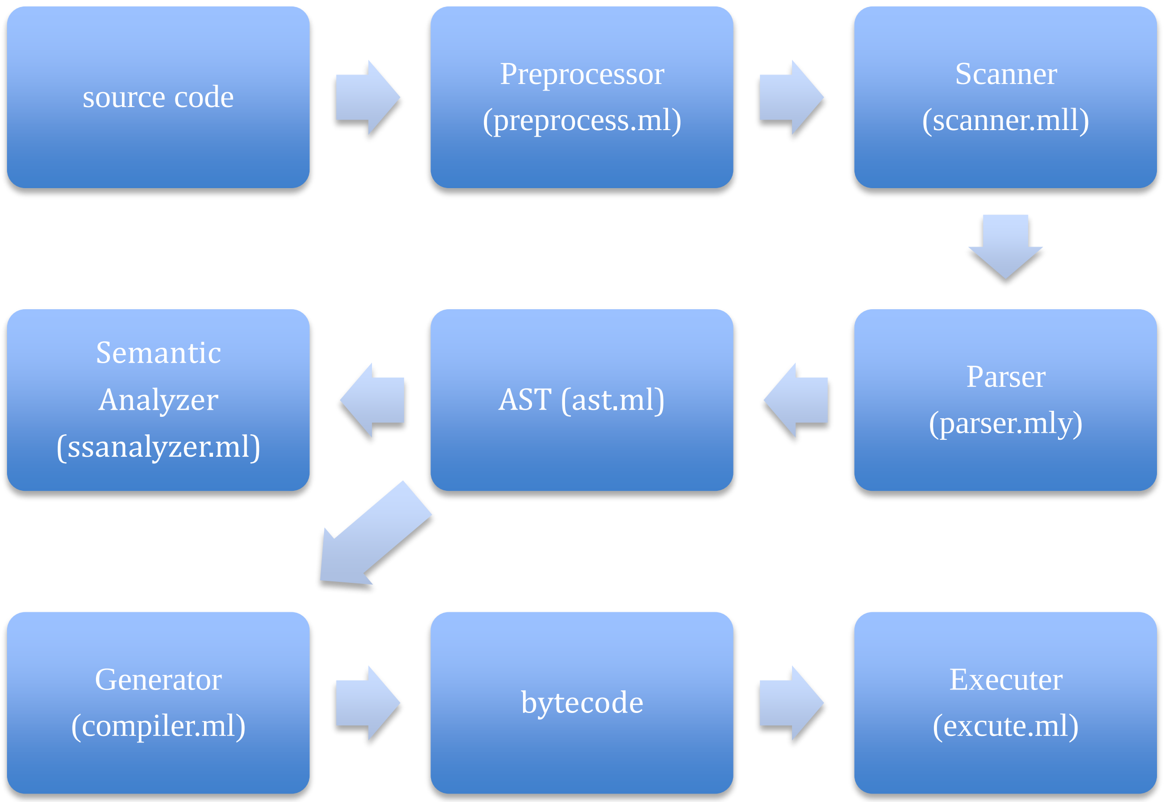

source code
Preprocessor (preprocess.ml)
Scanner (scanner.mll)
Parser (parser.mly)
AST (ast.ml)
Semantic Analyzer (ssanalyzer.ml)
Generator (compiler.ml)
bytecode
Executer (excute.ml)

Figure 5.1

6 Test Plan

It is important to have robust testing when writing a complex program like a compiler. During initial development, we tested basic operations manually. Once more features were added to the interpreter, we created a test harness shell script to begin automating our test cases and comparing actual outputs with the expected outputs. When we designed our test cases, we wrote one test for each feature specified in the Language Reference Manual (LRM); meanwhile, we built sequences of tests that start with the simplest version, and gradually evolve into more complex versions.

6.1 Representative Source Program and Target Program

6.1.1 Example 1

```
// Loops through various granularities and shows the result.

min_gran <- 2;
max_gran <- 65;

for i <- min_gran | i < max_gran | i <- i + 1
{
      c <- load ("../image/joconde.jpg", i);
      c -> out , true;

}
```

```
-- Byte code down -> right             Lct 1
Lit 2                                  Lod 3
Str 0                                  Jsr -1
Drp 1                                  Drp 1
Lit 65                                 Drp 1
Str 1                                  Lod 2
Drp 1                                  Lit 1
Lod 0                                  Bin +
Str 2                                  Str 2
Drp 1                                  Drp 1
Bra 16                                 Lod 2
Lct 0                                  Lod 1
Lod 2                                  Bin <
Jsr -3                                 Bne -18
Str 3                                  Hlt
Drp 1
```

6.1.2 Example 2

```
// Demo Library
```

```
Fun load_image(path, gran)
{
      c <- load(path, gran);
      return c;
}

Fun invert(c)
{
      z <- blank(c$h, c$w, c$g);

      for i <- 0 | i < c$g | i <- i+1
      {
            d <- c[=i];
            d[=i] <- c$g-i-1;
            z <- z & d;
      }

      return z;
}

-----

// Inverts Apple Image

include "../demo/demo_lib.eza";

a <- load_image("../image/apple.jpeg", 10);
i <- invert(a);
a -> out, true;
i -> out, true;



-- Bytecode Down -> Right
Lit 10                    Lct 2                     Rts 2
Lct 0                     Lod 1                     Ent 0
Jsr 20                    Jsr -1                    Lfp -2
Str 0                     Drp 1                     Catr $H
Drp 1                     Drp 1                     Lfp -2
Lod 0                     Hlt                       Catr $W
Jsr 29                    Ent 0                     Lfp -2
Str 1                     Lfp -2                    Catr $G
Drp 1                     Lfp -3                    Jsr -4
Lct 1                     Jsr -3                    Sfp 2
Lod 0                     Sfp 3                     Lit 0
Jsr -1                    Lfp 3                     Str 1
Drp 1                     Rts 2                     Drp 1
Drp 1                     Lit 0                     Bra 28
```

```
Lod 1
Lit 0
Lit 8
Lfp -2
Jsr -6
Sfp 3
Lod 1
Lit 0
Lit 8
Lfp -2
Catr $G
Lod 1
Bin -
Lit 1
Bin -
Lfp 3
Jsr -7
Lfp 2
Lfp 3
Bin MASK
Sfp 2
Drp 1
Lod 1
Lit 1
Bin +
Str 1
Drp 1
Lod 1
Lfp -2
Catr $G
Bin <
Bne -31
Lfp 2
Rts 1
Lit 0
Rts 1
```

6.2 Test Cases and Test Suits

To maintain confidence that new code does not break old code, we maintained a suite of unit tests. As we implemented each feature of our language, a basic unit test was added to ensure correct functionality. Intermediate and more difficult unit tests were also added for each feature to cover possible corner cases as more of the compiler was developed. Our test cases can be divided into two categories: functional tests and semantic tests. Functional tests are used to ensure the features of our compiler perform correctly. All functionality tests need to pass. The semantic analysis ensures that all the illegal semantic errors be caught, thus all semantic analysis tests need to fail. A complete list of test cases is as follows:

Table 6.1 Test Cases

assign	include	select2	typecheck5
blank	include2	selectall	typecheck6
canvas	loadcanvas	selectbinop1	typecheck7
canvasattr	mainfunction1	selecthslice	typecheck8
canvasset	mainfunction2	selecthsliceall	typecheck9
for1	mainfunction3	selectpoint1	
for2	mainfunction4	selectrect1	
fun1	mask	selectvslice	
fun2	minus1	selectvsliceall	
fun3	minus2	shift	
fun4	minus3	string	
fun5	outToFile	string2	
fun6	recursion1	string3	
if1	recursion2	typecheck1	
if2	recursion3	typecheck10	
if3	relops	typecheck11	
if4	scope1	typecheck12	
ifelse1	scope2	typecheck2	

ifelse2	scope3	typecheck3
ifelse3	select	typecheck4

6.3 Test Automation

A set of unit tests was written, each a .eza source file. Each unit test also has a corresponding .gs (gold standard) file which has the correct expected output of the program (or an error message for semantic tests). A shell script, `runtests.sh`, was written which when run, performs the following for each unit test: 1) Compile the test source file with the built executable and execute the bytecode (-c option), 2) pipe the output to a .out file, 3) compare the .out file with the corresponding .gs file, 4) if the contents are different, then output the difference to a .diff file using the diff utility, and finally 5) output the summary of the test run. It is important to note that each time a test is run, it is compiled and executed.

6.4 Test Responsibility

During initial development, each team member was responsible for informally testing his code by examining standard output. Once significant work was done on the interpreter, Joe added the runtests.sh script and began populating the unit test suite. As development proceeded, Joe and Dmitriy maintained the unit tests and added more as development proceeded.

7 Lessons Learned

7.1 Dmitriy Gromov

This project was a learning experience that was both very informative and eye opening. I had never been exposed to a functional language programming language before and so the very first lesson learned was how to more effectively read documentation of written languages. This was dually important, as OCaml does not have a huge user base and there are fewer examples of OCaml code as opposed to C or Java code.

Having been selected as the lead on this project, I also learned about some of the difficulties in managing a group. When work assignments weren't specified properly or there was confusion about what needed to be done, work was delayed or done incorrectly. Dealing with these issues taught me about the different kind of problems that can occur when managing groups of people and how to deal with them in an effective way. Towards the end of working on this project, it became easier to tell what could practically be done in parallel, who should be assigned what work, and when a democratic approach to decision making was not an appropriate thing to do.

Finally, aside from learning OCaml and a lot about compilers in general, I became much more familiar with Git. I had worked in industry for two years where we used source control but I never did anything past checking in and reverting my code. Over the course of this project we went through several broken branches and spent about a week fixing the repository. I became very familiar with what Git was capable of and gained a better understanding of how to use it. I was very grateful for this because it helped me with similar issues in other classes and I am sure it will help me in the future.

7.2 Feifei Zhong

It's my first time to do a complex a project like a complier. There are several things I learned from this project. Firstly, GitHub, a version control system, is the important tool I learned for doing a team project. Secondly, it's much harder than I thought to start writing the simple interpreter. The reason is that it took me almost one and half days to figure out the error in AST while I'm implementing the for loop. In addition, regression test suite we used is another important tool that I learned. I also learned some ways to wrap C/C++ code in Ocaml when I'm trying to figure out the canvas part of our project; however, my code didn't work, and we gave up using the Cimg library. Finally, effective teamwork is very important. It's better for team members to know each other well.

7.3 Yilei Wang

I first would like to thank my teammates. I really learned of a great deal from them, both knowledge and commitment to hard work.

Version control System is the first lesson I learned in this project. It’s so helpful in keeping track of progress when multiple people work on the same project. The most important thing learned is of course building a language from ground up. Starting with trivial constructs, I learned how to add to the output little by little and detect error in the process of building AST, scanner and parser.

7.4 Xin Ye

This is the first time I have participated in the compiler project. Through the course of it, there are many lessons I learned. First, since in this project, I mainly focused on the AST, scanner and parser, I realized a good design in them can save a lot of efforts in the coding process. What I said about a good design simply means the simpler the better. In order to reach this goal, it’s better to finish scanner, parser, and define the AST before finalizing the LRM, and then continuously modify them to make them simpler while integrating with other parts. Second, while coding and integrating codes with other team members, I came across and understood better about the issues that Prof. Edwards talked about in class, i.e. scoping. This is a really good learning process for me. Also Github is a very useful and powerful tool in order to make coding done neatly while working as a team. Finally, planning ahead, compromising and communicating with each other as much as possible are very important factors when working with other team members. Also it is important to help each other and look into the problems together if someone is not able to accomplish job on time. In summary, this project introduces me to the area of programming language design, which I have never had experience before and teaches me a lot.

7.5 Joe Lee

Overall, I enjoyed the work on this compiler project. I have had some industry experience in software engineering, so I was able to focus purely on implementation and not have to worry about learning version control, or good software engineering practices (one-click builds, unit testing). I enjoyed learning Git, which I had not used before. I did not enjoy learning OCaml at first, but looking back, I fully agree that it is a powerful language (particularly the pattern matching features), and am glad I had the opportunity to learn it. I am really glad that we started the initial compiler work (scanner, parser, interpreter) relatively early, as it paid off to have more of our language defined, so that we could focus on implementing the bytecode generation. If I had to pinpoint the most valuable lesson learned, I would say that I learned that the project work itself, though it was a lot, was not really the most difficult part; rather, it is the teamwork aspect. No one on the team knew each other, so it was difficult for everyone to work well together – it may have proved useful at the beginning of the semester to set expectations of each team member.

7.6 Advice For Future Teams

An obvious piece of advice would be to work together as early as possible. None of the members on our team knew each other, so the team had to spend some time adjusting to each team member's working styles, and getting everyone up to speed on Git version control. We are glad that we started our scanner and parser modules while working on the LRM; this gave us the advantage of mostly dealing with implementation after the LRM deadline. Another word of advice would be to spend time at the beginning of the semester setting expectations for each team member.

8 Appendix

8.1 preprocess.ml

```
(* FILENAME :  preprocess.ml
 * AUTHOR(S):  Dmitriy Gromov (dg2720)
 * PURPOSE  :
 *)

open Str

let read_file fname =
  let ic = open_in fname in
  let n = in_channel_length ic in
  let s = String.create n in
    really_input ic s 0 n;
    close_in ic;
    (s) ;;

let run fname =
  let prog_text = read_file fname in
  let white_sp = "[\r\n\t ]" in
  let inc_regex = Str.regexp (
    "include" (* include *)
    ^ white_sp ^ "+"  (* atleast 1 space btwn include and filename *)
    ^ "\"\\(.+\\)\""  (* "sometext" - Only the part between the quotes
goes in *)
    ^ white_sp ^ "*" ^ ";" ^ white_sp ^ "*";
  ) in (* Possible space between end of string and semi-colon *)

(*   let comm_regex = Str.regexp (
    "//.*" (* Two slashes followed by anything until new line. *)
  ) in
  let remove_comments text =
    let q = Str.global_replace comm_regex "" text in
      (q)
  in  *)

  let rec replace_include text =
    let find_include text =
      try
        ignore (Str.search_forward inc_regex text 0);
        true
      with Not_found ->
        false
    in
      if find_include text then
        let q = Str.global_substitute inc_regex (
          fun m ->
            let inc_name = Str.matched_group 1 m in
              read_file inc_name
        ) text
        in
          replace_include ( q )
          else
            text
  (* in replace_include (remove_comments prog_text) *)
in replace_include prog_text
```

8.2 scanner.mll

```
(* FILENAME :  scanner.mll
 * AUTHOR(S):  Joe Lee (jyl2157), Dmitriy Gromov (dg2720),
 *             Yilei Wang (yw2493), Peter Ye (xy2190), Feifei Zhong
(fz2185)
 * PURPOSE  :  Scanner definition for EZ-ASCII.
 *)

{
        open Parser
        exception Eof
}

let letter    = ['a'-'z' 'A'-'Z']
let digit     = ['0'-'9']
let dblquote  = '"'

(* printable ASCII chars, excluding double quote and forward slash *)
let printable = ['!' '#'-'.' '0'-'~']

(* escape sequences: newline, horiz tab, single/double quote, back/forw
slash *)
let esc_char  = "\\n" | "\\t" | "\\\"" | "\\\'" | "\\" | "/"
let comment = "//" _* ['\r' '\n']

(* allowable characters for strings *)
let strchar   = printable | ' ' | '\t' | esc_char

rule token = parse
        [' ' '\t']                              { token lexbuf }
      | ['\n' '\r']                             { token lexbuf }
      | "//"                                    { comment lexbuf }
      | ","                                     { COMMA }
      | ";"                                     { SEMICOLON }

      (* arithmetic operators *)
      | "+"                                     { PLUS }
      | "-"                                     { MINUS }
      | "*"                                     { TIMES }
      | "/"                                     { DIVIDE }
      | "%"                                     { MOD }

      (* relational operators *)
      | "&&"                                    { AND }
      | "||"                                    { OR }

      (* boolean operators/keywords *)
      | "<"                                     { LT }
      | ">"                                     { GT }
      | "="                                     { EQ }
      | "<="                                    { LEQ }
      | ">="                                    { GEQ }
      | "~="                                    { NEQ }
      | "~"                                     { NEGATE }
      | "true"                                  { BOOLLITERAL(true) }
      | "false"                                 { BOOLLITERAL(false) }

      (* canvas operators/keywords/constants *)
      | "&"                                     { MASK }
      | "["                                     { LBRACKET }
      | "]"                                     { RBRACKET }
      | ":"                                     { COLON }
      | "out"                                   { STDOUT }
      | "SHIFT_UP"                              { INTLITERAL(0) }
      | "SHIFT_LEFT"                            { INTLITERAL(1) }
      | "SHIFT_DOWN"                            { INTLITERAL(2) }
      | "SHIFT_RIGHT"                           { INTLITERAL(3) }
      | "$w"                                    { ATTR_W }
      | "$h"                                    { ATTR_H }
      | "$g"                                    { ATTR_G }

      (* statement operators/keywords *)
      | "if"                                  { IF }
      | "else"                                { ELSE }
      | "for"                                 { FOR }
      | "|"                                   { FOR_SEP }
      | "Fun"                                 { FXN }
      | "include"                             { INCLUDE }
      | "return"                              { RETURN }
        (* remove leading/trailing newlines
         * for braces *)
      | "{"                                   { LBRACE }
      | "}"                                   { RBRACE }
      | "("                                   { LPAREN }
      | ")"                                   { RPAREN }
      | "<-"                                  { ASSIGN }
      | "->"                                  { OUTPUT }

      (* built-in functions *)
      | "main"                                { MAIN }
      | "blank"                               { BLANK }
      | "load"                                { LOAD }
      | "map"                                 { MAP }
      | "shift"                               { SHIFT }

      | letter (letter | digit | '_')* as id    { ID(id) }
      | digit+ as lit                         {
INTLITERAL(int_of_string lit) }

      | dblquote strchar* dblquote as str       { STR(String.sub str 1
((String.length str) - 2)) }
      | eof                                   { EOF } (* raise Eof }
*)

and comment = parse
      (* end of line marks end of comment *)
        ['\n' '\r']                           { token lexbuf }

      (* ignore everything else *)
      | _                                     { comment lexbuf }
```

8.3 parser.mly

```
/* FILENAME :  parser.mly
 * AUTHOR(S):  Joe Lee (jyl2157), Dmitriy Gromov (dg2720),
 *             Yilei Wang (yw2493), Peter Ye (xy2190), Feifei Zhong
(fz2185)
 * PURPOSE  :  Parser definition for EZ-ASCII.
 */
```

```
%{ open Ast %}

%token <int> INTLITERAL
%token <bool> BOOLLITERAL
%token <string> ID
%token <string> CMP
%token <string> STR
%token TRUE, FALSE
%token AND, OR, COMMA, SEMICOLON, COLON, LBRACKET, RBRACKET, LPAREN,
RPAREN, EOF
%token LT, GT, EQ, LEQ, GEQ, NEQ, NEGATE
%token ATTR, MASK, IF, ELSE, FOR, FOR_SEP, INCLUDE, RETURN, LBRACE,
RBRACE, FXN
%token PLUS, MINUS, TIMES, DIVIDE, MOD
%token ASSIGN, OUTPUT, ATTR_W, ATTR_H, ATTR_G, STDOUT
%token MAIN, BLANK, LOAD, INCLUDE, MAP, SHIFT
%token CANVAS

%nonassoc NOELSE
%nonassoc ELSE
%left MASK, PLUS, MINUS        /* lowest precedence */
%left TIMES, DIVIDE, MOD
%left EQ, NEQ
%left LT, GT, GEQ, LEQ
%left AND, OR
%left LPAREN, RPAREN
%right ASSIGN
%nonassoc UMINUS         /* highest precedence */

%start program           /* the entry point */
%type <Ast.program> program

%%
program:
      /* nothing */                      { [], [] }
      | program stmt                     { List.rev($2 :: List.rev
(fst $1)), snd $1 }
      | program funcdecl                 { (fst $1), List.rev ($2 ::
List.rev (snd $1)) }

funcdecl:
        FXN ID LPAREN param_list RPAREN LBRACE stmt_list RBRACE
        { { fname = $2; params = List.rev $4; body = List.rev $7 } }
      | FXN MAIN LPAREN RPAREN LBRACE stmt_list RBRACE
        { { fname = "main"; params = []; body = List.rev $6 } }

param_list:
      /* nothing */                                  { [] }
      | ID                                             { [$1] }
      | param_list COMMA ID                            { $3 :: $1 }

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

stmt:
    ID ASSIGN expr SEMICOLON                  { Assign($1, $3) }
  | ID OUTPUT STDOUT SEMICOLON                { OutputC(Id($1),
BoolLiteral(false)) }
  | ID OUTPUT STDOUT COMMA expr SEMICOLON     { OutputC(Id($1), $5)}
  | ID OUTPUT expr SEMICOLON                   { OutputF(Id($1), $3,
BoolLiteral(false)) }
  | ID OUTPUT expr COMMA expr SEMICOLON        { OutputF(Id($1), $3,
$5) }
  | IF LPAREN expr RPAREN cond_body %prec NOELSE { If($3, $5) }
  | IF LPAREN expr RPAREN cond_body ELSE cond_body { If_else($3, $5,
$7) }
  | FOR stmt_in_for FOR_SEP expr FOR_SEP stmt_in_for LBRACE
  stmt_list RBRACE   { For($2, $4, $6, List.rev $8) }
  | RETURN expr SEMICOLON             { Return($2) }
  | INCLUDE STR SEMICOLON             { Include($2) }
  | ID LBRACKET select_expr RBRACKET ASSIGN expr SEMICOLON
{CanSet(Id($1), $3, $6)}

stmt_in_for:
  ID ASSIGN expr          { Assign($1, $3) }


select_bool_expr:
    LT expr                           { Select_Binop (Lt, $2) }
  | GT expr                           { Select_Binop (Gt, $2) }
  | EQ expr                           { Select_Binop (Eq, $2) }
  | LEQ expr                          { Select_Binop (Leq, $2) }
  | GEQ expr                          { Select_Binop (Geq, $2) }
  | NEQ expr                          { Select_Binop (Neq, $2) }
/*   | bool_expr AND bool_expr           { $1, $3 }
  | bool_expr OR bool_expr            { $1, $3 } */


select_expr:
    expr COMMA expr                           { Select_Point($1, $3) }
  | expr COLON expr COMMA expr COLON expr { Select_Rect($1, $3, $5, $7)
}
  | expr COMMA expr COLON expr            { Select_VSlice($1, $3, $5) }
  | expr COLON expr COMMA expr            { Select_HSlice($1, $3, $5) }
  | expr COMMA                            { Select_VSliceAll($1) }
  | COMMA expr                            { Select_HSliceAll($2) }
  | COMMA                               { Select_All }
  | select_bool_expr                    { Select_Bool($1) }

cond_body:
        /* If no braces are supplied in a
         * conditional body, we expect one statement;
         * Otherwise, we expect a statement list.
         */
        stmt                            { [$1] }
      | LBRACE stmt_list RBRACE         { List.rev $2 }

expr_list:
      /* nothing */                     { [] }
      | expr                            { [$1] }
      | expr_list COMMA expr            { $3 :: $1 }

expr:
        INTLITERAL                      { IntLiteral($1) }
      | MINUS INTLITERAL %prec UMINUS   { IntLiteral(- $2) }
      | BOOLLITERAL                     { BoolLiteral($1) }
      | STR                             { StrLiteral($1) }
      | ID                              { Id($1) }
      | expr PLUS expr                  { Binop($1, Plus, $3) }
      | expr MINUS expr                 { Binop($1, Minus, $3) }
      | expr TIMES expr                 { Binop($1, Times, $3) }
      | expr DIVIDE expr                { Binop($1, Divide, $3) }
      | expr MOD expr                   { Binop($1, Mod, $3) }
      | expr EQ expr                    { Binop($1, Eq, $3) }
      | expr NEQ expr                   { Binop($1, Neq, $3) }
      | expr LT expr                    { Binop($1, Lt, $3) }
      | expr GT expr                    { Binop($1, Gt, $3) }
      | expr LEQ expr                   { Binop($1, Leq, $3) }
      | expr GEQ expr                   { Binop($1, Geq, $3) }
      | expr OR expr                    { Binop($1, Or, $3) }
      | expr AND expr                   { Binop($1, And, $3) }
      | expr MASK expr                  { Binop($1, Mask, $3) }
      | ID LPAREN expr_list RPAREN      { Call($1, List.rev $3) }
      | LPAREN expr RPAREN              { $2 }
      | ID LBRACKET select_expr RBRACKET  { Select(Id($1), $3) }
      | ID ATTR_W                       { GetAttr (Id($1), W)}
      | ID ATTR_H                       { GetAttr (Id($1), H)}
      | ID ATTR_G                       { GetAttr (Id($1), G)}
      | LOAD LPAREN expr COMMA expr RPAREN { Load($3, $5) }
      | BLANK LPAREN expr COMMA expr COMMA expr RPAREN { Blank ($3, $5,
$7 ) }
      | SHIFT LPAREN ID COMMA expr COMMA expr RPAREN { Shift (Id($3),
$5, $7 ) }
```

8.4 ast.ml

```
(* FILENAME :  ast.ml
 * AUTHOR(S):  Joe Lee (jyl2157), Dmitriy Gromov (dg2720),
 *             Yilei Wang (yw2493), Peter Ye (xy2190)
 * PURPOSE  :  Define abstract syntax tree for EZ-ASCII.
 *)

type op = Plus | Minus | Times | Divide  | Mod
        | And | Or
        | Lt  | Gt  | Eq | Leq | Geq | Neq
        | Mask

let op_id op =
  match op with
    | Eq     -> 0
    | Neq    -> 1
    | Lt     -> 2
    | Leq    -> 3
    | Gt     -> 4
    | Geq    -> 5


let string_of_op op =
  match op with
      Plus   -> "+"
    | Minus  -> "-"
    | Times  -> "*"
    | Divide -> "/"
    | Mod    -> "%"
    | And    -> "&&"
    | Or     -> "||"
    | Eq     -> "="
    | Neq    -> "~="
    | Lt     -> "<"
    | Leq    -> "<="
    | Gt     -> ">"
    | Geq    -> ">="
    | Mask   -> "MASK"

type attr = W | H | G

type seltype =  POINT | RECT | VSLICE
              | HSLICE | VSLICE_ALL | HSLICE_ALL | ALL


type expr =
    IntLiteral of int                            (* 42 *)
  | StrLiteral of string                          (* "this is a string"
*)
  | BoolLiteral of bool                           (* true *)
  | Id of string                                  (* foo *)
  | Binop of expr * op * expr                     (* a + b *)
  | Call of string * expr list                    (* foo(1, 25) *)
  | Load of expr * expr                           (* load("filename",
10) *)
  | Blank of expr * expr * expr                   (* blank(x, y, g) *)
  | Shift of expr * expr * expr
  | Select_Point  of expr * expr                  (* [1,2] *)
  | Select_Rect   of expr * expr * expr * expr    (* [1:2, 3:4] *)
  | Select_VSlice of expr * expr * expr           (* [1, 3:4] *)
  | Select_HSlice of expr * expr * expr           (* [1:2, 3] *)
  | Select_VSliceAll of expr                      (* [3, ] *)
  | Select_HSliceAll of expr                      (* [, 3] *)
  | Select_All                                    (* [,] *)
  | Select of expr * expr                       (* canv[...] *)
  | Select_Binop of op * expr                     (* canv[<5] *)
  | Select_Bool of expr                           (* <5 *)
  | GetAttr of expr * attr                         (* canv$w *)

type stmt =                                       (* Statements *)
    Assign of string * expr                       (* foo <- 42 *)
  | OutputC of expr * expr                              (* canvas -> out
*)
  | OutputF of expr * expr * expr                      (* canvas ->
"C:\test.png" *)
  | If of expr * stmt list                        (* if (foo = 42) {} *)
  | If_else of expr * stmt list * stmt list       (* if (foo = 42) {}
else {} *)
  | For of stmt * expr * stmt * stmt list         (* for i <- 0 | i < 10
| i <- i + 1 { ... } *)
  | Return of expr                                (* return 42; *)
  | Include of string                             (* include
super_awesome.eza *)
  | CanSet of expr * expr * expr                (* can[..] <- 1 *)

type func_decl = {
  fname : string;                                 (* Name of the
function *)
  params : string list;                           (* Formal argument
names *)
  body : stmt list;
}

type program = stmt list * func_decl list         (* global vars, fxn
declarations *)
```

```
let string_of_attr = function
  W -> "$W"
| H -> "$H"
| G -> "$G"

let rec string_of_expr = function
    IntLiteral(l) -> string_of_int l
  | StrLiteral(l) -> "\"" ^ l ^ "\""
  | BoolLiteral(l) -> if l == true then "true" else "false"
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ (string_of_op o) ^ " " ^ string_of_expr
e2
  | Call(f, el)  ->  f ^ "(" ^ String.concat ", " (List.map
string_of_expr el) ^ ")"
  | Load(e1, gran) -> "Load(" ^ string_of_expr e1 ^ ", " ^
string_of_expr gran ^ ")"
  | Blank(e1, e2, e3) -> "Blank(" ^ string_of_expr e1 ^ ", " ^
string_of_expr e2 ^ ", " ^ string_of_expr e3 ^ ")"
  | Shift(e1, dir, e3) -> "Shift(" ^ string_of_expr e1 ^ ", " ^
string_of_expr dir ^ ", " ^ string_of_expr e3 ^ ")"

  | Select_Point (x, y) -> "[" ^ string_of_expr x ^ ", " ^
string_of_expr y ^ "] -- point select"
  | Select_Rect (x1, x2, y1, y2) ->  "[" ^ string_of_expr x1 ^  ":" ^
string_of_expr x2 ^ ", "
                                          ^ string_of_expr y1 ^  ":" ^
string_of_expr y2 ^ "] -- rect select"
  | Select_VSlice (x1, y1, y2)  ->  "[" ^ string_of_expr x1 ^ ", "
                                         ^ string_of_expr y1 ^ ":" ^
string_of_expr y2 ^ "] -- vslice"
  | Select_HSlice (x1, x2, y1) ->  "[" ^ string_of_expr x1 ^ ":" ^
string_of_expr x2
                                        ^ ", " ^ string_of_expr y1 ^ "]
-hslice"
  | Select_VSliceAll x1 -> "[" ^ string_of_expr x1 ^ ",] - vslice_all"
  | Select_HSliceAll y1 -> "[," ^ string_of_expr y1 ^ "] - hslice all"
  | Select_All -> "[,] - select all"
  | Select_Binop(o, e2) ->
      (
        match o with
          | And  -> "&&"
          | Or   -> "||"
          | Eq   -> "="
          | Neq  -> "~="
          | Lt   -> "<"
          | Leq  -> "<="
          | Gt   -> ">"
          | Geq  -> ">="
          | _    -> "error"
      )
    ^ " " ^ string_of_expr e2
  | Select_Bool(e1) -> "[" ^ string_of_expr e1  ^ "] "
  | Select (canv, selection) -> string_of_expr canv ^ string_of_expr
selection
  | GetAttr(canv, attr) -> string_of_expr canv ^ " -> " ^
string_of_attr attr

let rec string_of_stmt = function
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";"
  | If(e, sl1) -> "if (" ^ string_of_expr e ^  ")
                  {\n"  ^ String.concat "\n" (List.map string_of_stmt
sl1)  ^ "\n}"
  | If_else(e, sl1, sl2) ->
      "if (" ^ string_of_expr e
      ^ "){\n" ^ String.concat "\n" (List.map string_of_stmt sl1)
      ^ "\n}\nelse{\n"
      ^ String.concat "\n" (List.map string_of_stmt sl2)  ^ "\n}"
  | For(s1, e2, s3, sl4) ->
      "for (" ^ string_of_stmt s1  ^ " | " ^ string_of_expr e2 ^ " | "
^ string_of_stmt s3  ^ ")\n
      {\n" ^ String.concat "\n" (List.map string_of_stmt sl4)  ^ "\n}"
  | OutputC(e, render_expr) ->
      string_of_expr e ^ ", " ^ string_of_expr render_expr ^ " -> out"
  | OutputF(e, fname, render_expr) ->
      string_of_expr e ^ ", " ^ string_of_expr render_expr ^ " -> " ^
string_of_expr fname
  | Assign(v, e) ->
      v ^ " <- " ^ string_of_expr e
  | CanSet(can, sel, exp) -> string_of_expr can ^ string_of_expr sel ^
" <- " ^ string_of_expr exp
  | Include(str) ->
     "include " ^ str

let string_of_fdecl fdecl =
  fdecl.fname ^ "(" ^ String.concat ", " fdecl.params ^ ")\n{\n"
              ^ String.concat "\n" (List.map string_of_stmt fdecl.body)
              ^ "\n}\n"

let string_of_program (vars, funcs) =
  String.concat "\n" (List.map string_of_stmt vars) ^ "\n\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)
```

8.5 ssanalyzer.ml

```
(* FILENAME :  ssanalyzer.ml
 * AUTHOR(s):  Joe Lee (jyl2157)
 * PURPOSE  :  Checks for type errors, undefined var/fxn errors,
converts ast to sast.
 *)

open Ast
open Sast

type t =
    Void
  | Int
  | Bool
  | Char
  | String
  | RelOp
  | Canvas

let string_of_t = function
    Void -> "Void"
  | Int -> "Int"
  | Bool -> "Bool"
  | Char -> "Char"
  | String -> "String"
  | RelOp -> "&&, ||, =, ~=, <, <=. >, >="
  | Canvas -> "Canvas"

module StringMap = Map.Make(String)

type fxn_env = {
  mutable local_env     : (Sast.expr_detail * t) StringMap.t;
  mutable ret_type      : (Sast.expr_detail * t);
  fxn_name              : string;
  fxn_params            : string list;
  fxn_body              : Ast.stmt list;
}

(* Translation environment *)
type env = {
  mutable global_env    : (Sast.expr_detail * t) StringMap.t;
  mutable fxn_envs      : fxn_env StringMap.t;
}

exception TypeException of Ast.expr * Ast.expr * t * t
exception BinopException of Ast.op * Ast.expr * t
exception UndefinedVarException of Ast.expr
exception UndefinedFxnException of string * Ast.expr

(* takes Ast program and runs static semantic analysis (type errors,
etc..) *)
```

```
let semantic_checker (stmt_lst, func_decls) =

  (* Check an expr *)
  let rec expr env scope = function
      Ast.IntLiteral(i) ->
        Sast.IntLiteral(i), Int
    | Ast.StrLiteral(s) ->
        Sast.StrLiteral(s), String
    | Ast.BoolLiteral(b) ->
        Sast.BoolLiteral(b), Bool
    | Ast.Id(s) ->
        if scope <> "*global*"
        then
          (try
             let search_local = (StringMap.find s (StringMap.find scope
env.fxn_envs).local_env) in
               search_local
           with Not_found ->
             (try
                let search_global = StringMap.find s env.global_env in
                  search_global
              with Not_found ->
                raise (UndefinedVarException(Ast.Id(s)))))
        else
          (try
             let search_global = StringMap.find s env.global_env in
               search_global
           with Not_found ->
             raise (UndefinedVarException (Ast.Id(s))))
    | Ast.Binop(e1, op, e2) ->
        let (v1, t1) = expr env scope e1
        and (v2, t2) = expr env scope e2 in
          (match op with
               Ast.Plus ->
                 (match (t1, t2) with
                      (Int, Int) ->
                        Sast.Binop(e1, op, e2), Int
                    | (String, String) ->
                        Sast.Binop(e1, op, e2), String
                    | (_, _) ->
                        raise(TypeException(e2, Ast.Binop(e1, op, e2),
t1, t2))
                )
             | Ast.Minus | Ast.Times | Ast.Divide | Ast.Mod ->
                 (match (t1, t2) with
                      (Int, Int) ->
                        Sast.Binop(e1, op, e2), Int
                    | (_, _) ->
```

```
                        raise(TypeException(e2, Ast.Binop(e1, op, e2),
Int, t2))
                  )
              | Ast.Eq | Ast.Neq | Ast.Lt | Ast.Gt | Ast.Leq | Ast.Geq -
>
                  (match (t1, t2) with
                       (Int, Int) ->
                        Sast.Binop(e1, op, e2), Bool
                     | (_, _) ->
                        raise(TypeException(e2, Ast.Binop(e1, op, e2),
Int, t2))
                  )
              | Ast.Or | Ast.And ->
                  (match (t1, t2) with
                       (Bool, Bool) ->
                        Sast.Binop(e1, op, e2), Bool
                     | (_, _) ->
                        raise(TypeException(e2, Ast.Binop(e1, op, e2),
Bool, t2))
                  )
              | Ast.Mask ->
                  Sast.Binop(e1, op, e2), Canvas (* need to do *)
           );
    | Ast.Call(fname, actuals) ->
        (* no need to execute recursive calls *)
        if (scope <> "*global*") && (fname = scope)
        then
          (try
             let fxn_env_lookup = (StringMap.find fname env.fxn_envs)
in
               fxn_env_lookup.ret_type
           with Not_found ->
             raise (UndefinedFxnException (fname, Ast.Call(fname,
actuals))))
        else
          (try
             (* first evaluate the actuals *)
             let res = (List.map (expr env scope) (List.rev actuals))
in
             let fxn_env_lookup = (StringMap.find fname env.fxn_envs)
in
             let bindings = List.combine (fxn_env_lookup.fxn_params)
res in
             let rec init_loc_env accum_env = function
                 [] -> accum_env
               | (param_name, (sast_elem, typ)) :: tail ->
                   init_loc_env (StringMap.add param_name (sast_elem,
typ) accum_env) tail
             in
```

```
               (* Side effect: Initialize a local environment with the
new parameter values *)
               fxn_env_lookup.local_env <- init_loc_env StringMap.empty
bindings;
               (* execute the function_body, which will eventually
                * update the return type *)
               let _ = List.map (stmt env fxn_env_lookup.fxn_name)
fxn_env_lookup.fxn_body
               in
                 (* finally, return the possibly updated return type
                  *  (it is already initialized to (IntLiteral(0),
Int)) *)
                 fxn_env_lookup.ret_type
           with Not_found ->
             raise (UndefinedFxnException (fname, Ast.Call(fname,
actuals))))
    | Ast.Load(filepath_expr, gran_expr) ->
        let (v1, t1) = (expr env scope) filepath_expr
        and (v2, t2) = (expr env scope) gran_expr in
          if not (t1 = String)
          then
            raise(TypeException(filepath_expr, Ast.Load(filepath_expr,
gran_expr), String, t1))
          else
            if not (t2 = Int)
            then
              raise(TypeException(gran_expr, Ast.Load(filepath_expr,
gran_expr), Int, t2))
            else
              Sast.Load(filepath_expr, gran_expr), Canvas
    | Ast.Blank(height, width, granularity) ->
        let (v1, t1) = (expr env scope) height
        and (v2, t2) = (expr env scope) width
        and (v3, t3) = (expr env scope) granularity
        in
          (match (t1, t2, t3) with
               (Int, Int, Int) ->
                 Sast.Canvas, Canvas
             | (_, Int, Int) ->
                 raise(TypeException(height, Ast.Blank(height, width,
granularity), Int, t1))
             | (Int, _, Int) ->
                 raise(TypeException(width, Ast.Blank(height, width,
granularity), Int, t2))
             | (Int, Int, _) ->
                 raise(TypeException(granularity, Ast.Blank(height,
width, granularity), Int, t3))
             | (_, _, _) ->
```

```
                raise(TypeException(height, Ast.Blank(height, width,
granularity), Int, t1))
          )
      | Ast.Select_Point (x, y) ->
          let (v1, t1) = (expr env scope) x
          and (v2, t2) = (expr env scope) y
          in
            (match (t1, t2) with
                (Int, Int) ->
                  Sast.Canvas, Canvas
              | (Int, _) ->
                  raise(TypeException(y, Ast.Select_Point(x, y), Int,
t2))
              | (_, _) ->
                  raise(TypeException(x, Ast.Select_Point(x, y), Int,
t1))
            )
      | Ast.Select_Rect (x1, x2, y1, y2) ->
          let (v1, t1) = (expr env scope) x1
          and (v2, t2) = (expr env scope) x2
          and (v3, t3) = (expr env scope) y1
          and (v4, t4) = (expr env scope) y2
          in
            (match (t1, t2, t3, t4) with
                (Int, Int, Int, Int) ->
                  Sast.Canvas, Canvas
              | (Int, Int, Int, _) ->
                  raise(TypeException(y2, Ast.Select_Rect(x1, x2, y1,
y2), Int, t4))
              | (Int, Int, _, Int) ->
                  raise(TypeException(y1, Ast.Select_Rect(x1, x2, y1,
y2), Int, t3))
              | (Int, _, Int, Int) ->
                  raise(TypeException(x2, Ast.Select_Rect(x1, x2, y1,
y2), Int, t2))
              | (_, _, _, _) ->
                  raise(TypeException(x1, Ast.Select_Rect(x1, x2, y1,
y2), Int, t1))
            )
      | Ast.Select_VSlice (x1, y1, y2)  ->
          let (v1, t1) = (expr env scope) x1
          and (v2, t2) = (expr env scope) y1
          and (v3, t3) = (expr env scope) y2
          in
            (match (t1, t2, t3) with
                (Int, Int, Int) ->
                  Sast.Canvas, Canvas
              | (Int, Int, _) ->
```

```
                raise(TypeException(y2, Ast.Select_VSlice(x1, y1, y2),
Int, t3))
            | (Int, _, Int) ->
                raise(TypeException(y1, Ast.Select_VSlice(x1, y1, y2),
Int, t2))
            | (_, _, _) ->
                raise(TypeException(x1, Ast.Select_VSlice(x1, y1, y2),
Int, t1))
          )
    | Ast.Select_HSlice (x1, x2, y1) ->
        let (v1, t1) = (expr env scope) x1
        and (v2, t2) = (expr env scope) x2
        and (v3, t3) = (expr env scope) y1
        in
          (match (t1, t2, t3) with
               (Int, Int, Int) ->
                 Sast.Canvas, Canvas
            | (Int, Int, _) ->
                raise(TypeException(y1, Ast.Select_HSlice(x1, x2, y1),
Int, t3))
            | (Int, _, Int) ->
                raise(TypeException(x2, Ast.Select_HSlice(x1, x2, y1),
Int, t2))
            | (_, _, _) ->
                raise(TypeException(x1, Ast.Select_HSlice(x1, x2, y1),
Int, t1))
          )
    | Ast.Select_VSliceAll x ->
        let (v1, t1) = (expr env scope) x
        in
          (match t1 with
               Int ->
                 Sast.Canvas, Canvas
            | _ ->
                raise(TypeException(x, Ast.Select_VSliceAll(x), Int,
t1))
          )
    | Ast.Select_HSliceAll y ->
        let (v1, t1) = (expr env scope) y
        in
          (match t1 with
               Int ->
                 Sast.Canvas, Canvas
            | _ ->
                raise(TypeException(y, Ast.Select_HSliceAll(y), Int,
t1))
          )
    | Ast.Select_All ->
        Sast.Canvas, Canvas
    | Ast.Select (canv, selection) ->
        let (v1, t1) = (expr env scope) canv
        and (v2, t2) = (expr env scope) selection
        in
          (match (t1, t2) with
               (Canvas, Canvas) ->
                 Sast.Canvas, Canvas
             | (Canvas, _) ->
                 raise(TypeException(selection, Ast.Select(canv,
selection), Canvas, t2))
             | (_, _) ->
                 raise(TypeException(canv, Ast.Select(canv, selection),
Canvas, t1))
          )
    | Ast.Select_Binop(op, e) ->
        let (v1, t1) = (expr env scope) e
        in
          (match (op, t1) with
               (* op must be a relational operator *)
               (Ast.Eq, Int | Ast.Neq, Int | Ast.Lt, Int | Ast.Gt, Int
                | Ast.Leq, Int | Ast.Geq, Int) ->
                 Sast.Canvas, Canvas
(*
             | (Ast.And, Bool | Ast.Or, Bool) ->
                 Sast.Canvas, Canvas
 *)
             | (Ast.Eq, _ | Ast.Neq, _ | Ast.Lt, _ | Ast.Gt, _ |
Ast.Leq, _ | Ast.Geq, _) ->
                 raise(TypeException(e, Ast.Select_Binop(op, e), Int,
t1))
(*
             | (Ast.And, _ | Ast.Or, _) ->
                 raise(TypeException(e, Ast.Select_Binop(op, e), Bool,
t1))
 *)
             | (_, _) ->
                 raise(BinopException(op, Ast.Select_Binop(op, e),
RelOp))
          )
    | Ast.Select_Bool(e) ->
        (* e here is select_bool_expr which ultimately has type Canvas
*)
        let (v1, t1) = (expr env scope) e
        in
          (match t1 with
               Canvas ->
                 Sast.Canvas, Canvas
             | _ ->
```

```
                raise(TypeException(e, Ast.Select_Bool(e), Canvas,
t1))
          )
    | Ast.Shift(canv, dir, count) ->
        let (v1, t1) = (expr env scope) canv
        and (v2, t2) = (expr env scope) dir
        and (v3, t3) = (expr env scope) count
        in
          (match (t1, t2, t3) with
               (Canvas, Int, Int) ->
                 Sast.Canvas, Canvas
             | (Canvas, Int, _) ->
                 raise(TypeException(count, Ast.Shift(canv, dir,
count), Int, t3))
             | (Canvas, _, Int) ->
                 raise(TypeException(dir, Ast.Shift(canv, dir, count),
Int, t2))
             | (_, _, _) ->
                 raise(TypeException(canv, Ast.Shift(canv, dir, count),
Canvas, t1))
          )
    | Ast.GetAttr(canv, attr) ->
        let (v1, t1) = (expr env scope) canv in
          (match t1 with
               Canvas ->
                 (match attr with
                      Ast.W | Ast.H | Ast.G ->
                          Sast.Canvas, Canvas )
             | _ ->
                 raise(TypeException(canv, Ast.GetAttr(canv, attr),
Canvas, t1))
          )

  (* execute statement *)
  and stmt env scope = function
      Ast.Assign(var, e) ->
        let ev = (expr env scope e) in
          if scope <> "*global*"
          then
            (*
             * if we are in a function, variable lookup proceeds as:
             * 1) Check if the variable is a formal (parameter)
             * 2) Check if the variable is declared globally
             * 3) Finally if both 1 and 2 don't hold, create a new
local
             *)
            (
              let f_env = (StringMap.find scope env.fxn_envs)
              in
```

```
                if (StringMap.mem var f_env.local_env)
                then
                  f_env.local_env <- (StringMap.add var ev
f_env.local_env)
                else
                  if (StringMap.mem var env.global_env)
                  then
                    env.global_env <- (StringMap.add var ev
env.global_env)
                  else
                    f_env.local_env <- StringMap.add var ev
f_env.local_env
            )
          else
              env.global_env <- (StringMap.add var ev env.global_env)
    | Ast.OutputC(var, var_rend) ->
        let (var_val, var_typ) = expr env scope var
        and (var_rend_val, var_rend_typ) = expr env scope var_rend
        in
        (match (var_typ, var_rend_typ) with
               (Canvas, Bool) ->
                 ();
             | (_, Bool ) ->
                  ( match var_rend with
                      Ast.BoolLiteral(b) -> if b
                                          then
raise(TypeException(var_rend, var_rend, Bool, var_rend_typ))
                                          else ()
                    | _ -> raise(TypeException(var_rend, var_rend,
Bool, var_rend_typ)) ) ;
             | (_, _) ->
                 raise(TypeException(var_rend, var_rend, Bool,
var_rend_typ))
          );

    | Ast.OutputF(var, var_fname, var_rend) ->
        let (var_val, var_typ) = expr env scope var
        and (var_rend_val, var_rend_typ) = expr env scope var_rend
        (* and (var_fname_val, var_fname_typ) = expr env scope
var_fname *)
        in
        (match (var_typ, var_rend_typ) with
               (Canvas, Bool) ->
                 ();
             | (_, Bool ) ->
                  ( match var_rend with
                      Ast.BoolLiteral(b) -> if b
                                          then
raise(TypeException(var_rend, var_rend, Bool, var_rend_typ))
```

```
                                        else ()
                    | _ -> raise(TypeException(var_rend, var_rend,
Bool, var_rend_typ)) ) ;
             | (_, _) ->
                 raise(TypeException(var_rend, var_rend, Bool,
var_rend_typ))
          );

    | Ast.If(cond, stmt_lst) ->
        let (cond_val, cond_typ) = expr env scope cond in
          (match cond_typ with
               Bool ->
                 ();
             | _ ->
                 raise(TypeException(cond, cond, Bool, cond_typ))
          );
          (* regardless of the condition, check the statements *)
          List.iter (stmt env scope) stmt_lst;
          ();
    | Ast.If_else(cond, stmt_lst1, stmt_lst2) ->
        let (cond_val, cond_typ) = expr env scope cond in
          (match cond_typ with
               Bool -> ();
             | _ -> raise(TypeException(cond, cond, Bool, cond_typ))
          );
          (* regardless of the condition, check both blocks *)
          List.iter (stmt env scope) stmt_lst1;
          List.iter (stmt env scope) stmt_lst2;
          ();
    | Ast.For(s1, e1, s2, stmt_lst) ->
        (stmt env scope s1);
        let (e1_val, e1_typ) = (expr env scope e1)
        in
          (match e1_typ with
               Bool -> ();
             | _ -> raise(TypeException(e1, e1, Bool, e1_typ))
          );
          (* we only need to check the statement body once *)
          List.iter (stmt env scope) stmt_lst;
          stmt env scope s2;
          ();
    | Ast.Return(e) ->
        let (v, typ) = expr env scope e
        and fxn_env_lookup = StringMap.find scope env.fxn_envs
        in
          fxn_env_lookup.ret_type <- (v, typ);
    | Ast.Include(str) ->
        (); (* no type checking needed since we know it's already a
string *)
```

```
    | Ast.CanSet (canv, select_expr, set_expr) ->
        let (v1, t1) = (expr env scope) canv
        and (v2, t2) = (expr env scope) select_expr
        and (v3, t3) = (expr env scope) set_expr
        in
         (match (t1, t2, t3) with
              (Canvas, Canvas, Int) ->
                ();
            | (Canvas, Canvas, _) ->
                raise(TypeException(set_expr, set_expr, Int, t3))
            | (Canvas, _, Int) ->
                raise(TypeException(select_expr, select_expr, Canvas,
t2))
            | (_, _, _) ->
                raise(TypeException(canv, canv, Canvas, t1))
         )

  (************************
   * start main code here
   ************************)

  in let env = {
    global_env    = StringMap.empty;
    fxn_envs      = StringMap.empty;
  } in

  let rec add_fxn accum_env = function
      [] -> accum_env
    | fxn_decl :: rest ->
        let f_env =
          {
            local_env  = StringMap.empty;
            ret_type   = (Sast.IntLiteral(0), Int); (* return 0 by
default *)
            fxn_name   = fxn_decl.fname;
            fxn_params = fxn_decl.params;
            fxn_body   = fxn_decl.body;
          }
        in
          env.fxn_envs <- StringMap.add fxn_decl.fname f_env
env.fxn_envs;
          add_fxn env rest
  in
  (* add func_decls to env.fxn_envs *)
  let env = add_fxn env func_decls
  in
    (* execute the global statements *)
    List.iter (stmt env "*global*") stmt_lst;
    (* return the ast program unchanged for now - need to return
sast.program
     * later *)
    (stmt_lst, func_decls)
```

8.6 sast.ml

```
(* FILENAME :  sast.ml
 * AUTHOR(s):  Joe Lee (jyl2157)
 * PURPOSE  :  Defines sast.
 *)

type expr_detail =
    IntLiteral of int
  | StrLiteral of string
  | BoolLiteral of bool
  | Canvas (* of Ast.expr * Ast.expr * Ast.expr *)
  | Binop of Ast.expr * Ast.op * Ast.expr
  | Load of Ast.expr * Ast.expr
```

8.7 hashtypes.ml

```
(* FILENAME :  hashtypes.ml
 * AUTHOR(s):  Joe Lee (jyl2157)
 * PURPOSE  :  Define custom types for hash map values to support EZ-
ASCII's
 * types.
 *)

open Canvas

type ct =
    (* Note: The compiler will not add any int types to the hash map
     * but the bytecode executor might during binop operations *)
    Int of int
  | String of string
  | Bool of bool
  | Canvas of Canvas.canvas

let string_of_ct render = function
    Int(i) -> string_of_int i
  | String(s) -> ( Scanf.unescaped s )
  | Bool(b) -> string_of_bool b
  | Canvas(c) -> (Canvas.string_of_canvas c Canvas.default_map render)
```

8.8 interpret.ml

```
(* FILENAME :  interpret.ml
 * AUTHOR(S):  Joe Lee (jyl2157), Dmitriy Gromov (dg2720)
 * PURPOSE  :  Interpreter for EZ-ASCII.
 *)

open Parser
open Scanner
open Ast

module NameMap =
  Map.Make(struct
             type t = string
             let compare x y = Pervasives.compare x y
           end)

exception ReturnException of string * string NameMap.t NameMap.t


(* given a NameMap, print its bindings
 *)
let env_to_str m =
  let bindings = NameMap.bindings m in
  let rec print_map_helper s = function
      [] -> s
    | hd :: tl -> print_map_helper ((fst hd) ^ " = " ^ (string_of_int
(snd hd)) ^ "\n" ^ s) tl
  in print_map_helper "" bindings

(* helper function *)
let bool_of_int i =
  if i > 0 then true
  else false



(* ==============================================================
 * Interpreter main method. Its called in ezac with -i switch.
 *
 *
 * ============================================================== *)
let run lexbuf =
    let (stmt_lst, fxns_lst) =
      try
        (Parser.program Scanner.token lexbuf)
      with Parsing.Parse_error ->
        let curr = lexbuf.Lexing.lex_curr_p in
        let line = curr.Lexing.pos_lnum in
        let cnum = curr.Lexing.pos_cnum - curr.Lexing.pos_bol in
        let tok = Lexing.lexeme lexbuf in
          print_endline (">>> Parse error at line " ^ (string_of_int
line) ^ ", character " ^ (string_of_int cnum) ^ ": '" ^ tok ^ "'");
          exit 0;
    in
      (* ======================================================
       * eval function
       *
       * Takes Ast.expr type and evaluates it, returning a
       * string value and updated environment pair.
       * ====================================================== *)
    let rec eval env scope = function
          Ast.IntLiteral(e1)      -> string_of_int e1, env
        | Ast.StrLiteral(e1)      -> e1, env
        | Ast.BoolLiteral(e1)     ->
            if e1 then
              "1", env
            else
              "0", env
        | Ast.Id(var)             ->
            let local_decls = (NameMap.find scope env)
            in
              (* Look for the variable in the function's local scope *)
              if NameMap.mem var local_decls
              then (NameMap.find var local_decls), env
              (* If variable not found in local scope, look in global
scope *)
              else
                let global_decls = (NameMap.find "*global*" env)
                in
                  if NameMap.mem var global_decls
                  then (NameMap.find var global_decls), env
                  (* Otherwise, error. *)
                  else raise (Failure (">>> Undefined identifier: " ^
var))
        | Ast.Binop(e1, op, e2)   ->
            let v1, env = eval env scope e1 in
            let v2, env = eval env scope e2 in
            let boolean i = if i then 1 else 0 in
              string_of_int (
                match op with
                    Plus   -> (int_of_string v1) + (int_of_string v2)
                  | Minus  -> (int_of_string v1) - (int_of_string v2)
                  | Times  -> (int_of_string v1) * (int_of_string v2)
                  | Divide -> (int_of_string v1) / (int_of_string v2)
                  | Mod    -> (int_of_string v1) mod (int_of_string v2)
                  | Eq     -> boolean((int_of_string v1) ==
(int_of_string v2))
```

```
                  | Neq    -> boolean((int_of_string v1) !=
(int_of_string v2))
                  | Lt     -> boolean((int_of_string v1) <
(int_of_string v2))
                  | Gt     -> boolean((int_of_string v1) >
(int_of_string v2))
                  | Leq    -> boolean((int_of_string v1) <=
(int_of_string v2))
                  | Geq    -> boolean((int_of_string v1) >=
(int_of_string v2))
                  | Or     -> boolean((bool_of_int (int_of_string v1))
|| (bool_of_int (int_of_string v2)))
                  | And    -> boolean((bool_of_int (int_of_string v1))
&& (bool_of_int (int_of_string v2)))
                  | Mask   -> 1 (* NEED TO DO *)
              ), env

        | Ast.Call(fxn_name, param_exprs)   ->
            (* Find the target function from the list of function
declarations.
             * Target function is as defined in the Ast:
             *   {
             *     fname :  string;
             *     params : string list;
             *     body :   stmt list;
             *   }
             *
             * Note: Need to add error handling for when user tries to
call
             * an undefined function. *)
            let target_fxn = List.find (fun s -> s.fname = fxn_name)
fxns_lst
            in
            let fxn_env = NameMap.find fxn_name env
            in
            (* For every parameter in the function environment,
             * initialize it to the VALUE of the parameter expression
             * that the user passed in.
             * (e.g. if the function declaration is foo(a, b, c),
             * and the user calls with foo(1+2, 3+4, 5+6),
             * evaluate each parameter expression and update the
             * function environment accordingly.
             *
             * Note: error handling needs to be added for the case
where
             * the user supplies the incorrect number of arguments. *)
            let rec setparams fxn_env'  = function
                [] -> fxn_env'
              | hd :: tail ->
```

```
                  (* make sure we evaluate the parameter
                   * expressions in their CURRENT environments *)
                  let (param_expr_val, _) = eval env scope (snd hd)
                  in
                    setparams (NameMap.add (fst hd) param_expr_val
fxn_env') tail
            in let fxn_env =
              setparams fxn_env (List.combine target_fxn.params
param_exprs)
            in
            (* Update the global env with the fxn_env
             * before executing the function body. *)
            let update_env = (NameMap.add target_fxn.fname fxn_env env)
            in
              try
                let (update_env', fxn_name) = List.fold_left (exec)
                                                         (update_env,
target_fxn.fname)

target_fxn.body
                (* by default, return 0 if no return statement given *)
                in "", update_env'
              with ReturnException(ret_val, eval_env) -> ret_val,
eval_env

    and
      (* ======================================================
       * exec function
       *
       * Takes Ast.stmt type and executes it, returning an
       * updated environment.
       * ====================================================== *)
       exec (env, scope) = function

          Assign(var, e) ->
            (* update the environment for the expression first *)
            let e_val, e_env = eval env scope e in
            (* print_endline (">>> " ^ var ^ " assigned " ^ e_val); *)
            let update_global = function
                (var, var_val) ->
                  let updated_submap = (NameMap.add var var_val
(NameMap.find "*global*" e_env))
                  in (NameMap.add "*global*" updated_submap e_env)
            in
              if scope = "*global*"
              then
                (* If global scope, then just update the global env *)
                (update_global (var, e_val)), scope
              else
```

```
                let local_decls = (NameMap.find scope e_env)
                in
                  (* Look for variable in local scope *)
                  if NameMap.mem var local_decls
                  then
                    (* If variable found in local scope, assign to
local version;
                     * otherwise, assign it in global scope. *)
                    let updated_env = (NameMap.add var e_val
local_decls)
                    in (NameMap.add scope updated_env e_env), scope
                    else (update_global (var, e_val)), scope

        | OutputC(var) ->
            let e_val, e_env = eval env scope var
            in
              (* print_endline(e_val); *)
              (* Printf.printf( "%s\n", e_val); *)
              (* Printf.printf "%s\n" (Scanf.unescaped e_val); *)
              e_env, scope;

        | OutputF(var, f) ->
            (* No-op, NEED TO DO) *)
            env, scope;

        | If(cond, stmt_lst) ->
            let c1, c_env = eval env scope cond in
              if (bool_of_int (int_of_string c1)) then
                List.fold_left (exec) (c_env, scope) stmt_lst
              else env, scope

        | If_else(cond, stmt_lst1, stmt_lst2) ->
            let c1, c_env = eval env scope cond in
              if (bool_of_int (int_of_string c1)) then
                List.fold_left (exec) (c_env, scope) stmt_lst1
              else
                List.fold_left (exec) (c_env, scope) stmt_lst2

        | For(s1, e1, s2, stmt_lst) ->
            let (env, scope) = (exec (env, scope)) s1 in
            let rec loop (env, scope) =
              let v, env = eval env scope e1 in
                if (bool_of_int (int_of_string v)) then
                  let (body_env, body_scope) = List.fold_left (exec)
(env, scope) stmt_lst in
                    loop (exec (body_env, body_scope) s2)
                    else env, scope
            in loop (env, scope)
        | Return(exp) ->
            let exp_val, exp_env = eval env scope exp in
              raise (ReturnException(exp_val, exp_env))


      in
      let rec parse_stmts env scope = function
          [] -> env
        | hd :: tail ->
            (* execute statements and return updated environments *)
            let (updated_env, scope) = exec (env, scope) hd
            in
              (parse_stmts updated_env scope) tail
      in
        try
          let init_env =
            (* The initial global NameMap env consists of one key-value
             * pair, *global* : NameMap.empty.  Then for every
             * function declaration, add a key-value pair where
             * the key is the function name, and the value is
             * a NameMap initialized with the function parameters
             * as keys, and empty strings for values.  *)
            (List.fold_left
               (fun new_env fxn_decl ->
                 let fxn_env = (List.fold_left
                                 (fun tmp_env param_id -> NameMap.add
param_id "" tmp_env)
                                 NameMap.empty
                                 fxn_decl.params)
                 in NameMap.add fxn_decl.fname fxn_env new_env)
               (NameMap.add "*global*" NameMap.empty NameMap.empty)
               fxns_lst)
          in (parse_stmts init_env "*global*") stmt_lst
        with
          | Failure(s) ->
              print_endline s;
              exit 0;
```

8.9 compiler.ml

```
(* FILENAME :  compiler.ml
 * AUTHOR(S):  Joe Lee (jyl2157), Dmitriy Gromov (dg2720)
 * PURPOSE  :  Translate abstract syntax tree to bytecode.
 *)

open Ast
open Bytecode
```

```
(*open Ezatypes*)
open Hashtypes
open Canvas

(* global hash table
 * keys are absolute integer addresses
 * values are of type Hashtypes.ct *)
let glob_ht = Hashtbl.create 2048

(* initialize hash counter to keep track of next available key *)
let hash_counter = ref 0;

module StringMap = Map.Make(String)

(* Translation environment *)
type env = {
  function_idx          : int StringMap.t;   (* Index for each function
*)
  mutable global_idx    : int StringMap.t;   (* "Address" for global
vars *)
  mutable local_idx     : int StringMap.t;   (* FP offset for args,
locals *)
  num_formals           : int;               (* Number of parameters *)
}

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

(* 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

(* Translate a program in AST form into a bytecode program.
 * Throw an exception if something is wrong, (e.g. reference
 * to an unknown var or function.
 *)
let translate (stmt_lst, func_decls) =

  let built_in_functions =
    let rec bif_helper map counter = function
        [] -> map
      | hd :: tl ->
          (bif_helper (StringMap.add hd (counter) map) (counter-1)) tl
    (* add built-in functions here *)
    (* reserve -1 for printing *)
    (* reserve -2 for printing to file *)
    in (bif_helper StringMap.empty (-3)) ["load"; "blank"; "shift"]
  in

  let function_indexes = string_map_pairs built_in_functions
                           (* start built-in functions at 2, reserve 1
for
                            * top-level statements pseudofunction *)
                           (enum 1 2 (List.map (fun f -> f.fname)
func_decls)) in

  (* Translate an expr *)
  let rec expr env = function
      Ast.IntLiteral(i) -> [Lit i]
    | Ast.StrLiteral(s) ->
        Hashtbl.add glob_ht !hash_counter (Hashtypes.String s);
        let ret_val = [Lct !hash_counter] in
          hash_counter := !hash_counter+1; (* incr value of
hash_counter ref *)
          ret_val
    | Ast.BoolLiteral(b) ->
        Hashtbl.add glob_ht !hash_counter (Hashtypes.Bool b);
        let ret_val = [Lct !hash_counter] in
          hash_counter := !hash_counter+1; (* incr hash_counter in-
place *)
          ret_val
    | Ast.Id(s) ->
        (try
           let search_local = (StringMap.find s env.local_idx) in
             [Lfp search_local]
         with Not_found ->
           (try
              let search_global = StringMap.find s env.global_idx in
                [Lod search_global]
            with Not_found ->
              raise (Failure ("Undeclared variable " ^ s))))
    | Ast.Binop(e1, op, e2) ->
        let ev1 = (expr env) e1
        and ev2 = (expr env) e2 in
        ev1 @ ev2 @ [Bin op]

    | Ast.Call(fname, actuals) ->
        (try
           (* first evaluate the actuals *)
           let res = (List.map (expr env) (List.rev actuals))
           in
             (List.concat res) @ [Jsr (StringMap.find fname
env.function_idx)]
         with Not_found ->
           raise (Failure ("Undefined function: " ^ fname)))
    | Ast.Load(filepath_expr, gran_expr) ->
        let ev1_val = (expr env) filepath_expr
        and ev2_val = (expr env) gran_expr in
          ev1_val @ ev2_val @ [Jsr (-3)]

    | Ast.Blank(height, width, granularity) ->
        let ev1_val = (expr env) height
        and ev2_val = (expr env) width
        and ev3_val = (expr env) granularity in
          ev1_val @ ev2_val @ ev3_val @ [Jsr (-4)]

    | Ast.Select_Point (x, y) ->
        let ev1_val = (expr env) x
        and ev2_val = (expr env) y in
          ev1_val @ ev2_val @ [Lit (Canvas.select_type (Canvas.POINT))]

    | Ast.Select_Rect (x1, x2, y1, y2) ->
        let ev1_val = (expr env) x1
        and ev2_val = (expr env) x2
        and ev3_val = (expr env) y1
        and ev4_val = (expr env) y2 in
          ev1_val @ ev2_val  @ ev3_val @ ev4_val  @  [Lit
(Canvas.select_type (Canvas.RECT))]

    | Ast.Select_VSlice (x1, y1, y2)  ->
        let ev1_val = (expr env) x1
        and ev2_val = (expr env) y1
        and ev3_val = (expr env) y2 in
          ev1_val @ ev2_val  @ ev3_val @  [Lit (Canvas.select_type
(Canvas.VSLICE))]

    | Ast.Select_HSlice (x1, x2, y1) ->
        let ev1_val = (expr env) x1
        and ev2_val = (expr env) x2
        and ev3_val = (expr env) y1 in
          ev1_val @ ev2_val  @ ev3_val @ [Lit (Canvas.select_type
(Canvas.HSLICE))]

    | Ast.Select_VSliceAll x ->
        let ev1_val = (expr env) x in
          ev1_val @ [Lit (Canvas.select_type (Canvas.VSLICE_ALL))]

    | Ast.Select_HSliceAll y ->
        let ev1_val = (expr env) y in
          ev1_val @ [Lit (Canvas.select_type (Canvas.HSLICE_ALL))]

    | Ast.Select_All ->
       [Lit (Canvas.select_type (Canvas.ALL))]
    | Ast.Select (canv, selection) ->
        let ev1_val = (expr env) canv
        in
          (expr env) selection @ ev1_val @ [Jsr (-6)]
    | Ast.Select_Bool(e) ->
        let expr_val = (expr env) e in
        expr_val @ [Lit (Canvas.select_type (Canvas.BOOL))]

    | Ast.Select_Binop (op, e) ->
      let expr_val = (expr env) e
      and op_id = Ast.op_id op in
      expr_val @ [Lit op_id]

    | Ast.Shift(canv, dir, count) ->
        let canv_val = (expr env) canv
        and dir_val = (expr env) dir
        and count_val = (expr env) count in
        count_val @ dir_val  @ canv_val @ [Jsr (-5)]

    | Ast.GetAttr(canv, attr) ->
        let canv_val = (expr env) canv
        in
          canv_val @ [CAtr attr]
  (* *)
  and  stmt env scope = function
      (* need to update assign later *)
      Ast.Assign(var, e) ->
        let ev = (expr env e) in
          ev @
          if scope = "*local*"
          then
            (*
             * if we are in a function, variable lookup proceeds as:
             * 1) Check if the variable is a formal (parameter)
             * 2) Check if the variable is declared globally
             * 3) Finally if both 1 and 2 don't hold, create a new
local
             *)
            if (StringMap.mem var env.local_idx)
            then
              let exis_local_idx = StringMap.find var env.local_idx in
                (* side effect: update env.local_idx *)
                env.local_idx <- (StringMap.add var exis_local_idx
env.local_idx);
                [Sfp exis_local_idx]
            else
              if (StringMap.mem var env.global_idx)
                then
                let exis_global_idx = StringMap.find var env.global_idx
in
                  (* side effect: update env.global_idx *)
                  env.global_idx <- (StringMap.add var exis_global_idx
env.global_idx);
                  [Str exis_global_idx]
              else
                (* note the +1 for the next available local idx *)
                let new_local_idx = (List.length (StringMap.bindings
env.local_idx)) + 1
                in
                  (* side effect: modify env.local_idx *)
                  env.local_idx <- (StringMap.add var new_local_idx
env.local_idx);
                  [Sfp new_local_idx]
          else
            [Str
               (if (StringMap.mem var env.global_idx)
                then
                  let exis_global_idx = StringMap.find var
env.global_idx in
                    env.global_idx <- (StringMap.add var
exis_global_idx env.global_idx);
                    exis_global_idx
                    else
                      let new_global_idx = (List.length
(StringMap.bindings env.global_idx))
                      in
                        (* side effect: modify env.global_idx *)
                        env.global_idx <- (StringMap.add var
new_global_idx env.global_idx);
                        new_global_idx)]


    | Ast.OutputC(var, rend) ->
        let var_val = (expr env var) in
        let rend_val = (expr env rend) in
        rend_val @ var_val @ [Jsr (-1)]

    | Ast.OutputF(var, fn, rend) ->
        let var_val = (expr env var) in
        let fn_val = (expr env fn) in
        let rend_val = (expr env rend) in
        rend_val @ fn_val @ var_val @ [Jsr (-2)]

    | Ast.If(cond, stmt_lst) ->
        let t_stmts = (List.concat (List.map (stmt env scope)
stmt_lst))
        in
```

```
          (expr env cond) @
          [Beq (1 + List.length t_stmts)] @
          t_stmts
    | Ast.If_else(cond, stmt_lst1, stmt_lst2) ->
        let t_stmts = (List.concat (List.map (stmt env scope)
stmt_lst1))
        and f_stmts = (List.concat (List.map (stmt env scope)
stmt_lst2))
        in
          (expr env cond) @
          [Beq (2 + List.length t_stmts)] @
          t_stmts @
          [Bra (1 + List.length f_stmts)] @
          f_stmts
    | Ast.For(s1, e1, s2, stmt_lst) ->
        (* note: order of executing statements and evaluating
expressions here
         * matters since the environment can be updated on each
         * execution/evaluation
         *)
        let s1' = (stmt env scope s1)
        and e1' = (expr env e1)
        and for_body_stmts = (List.concat (List.map (stmt env scope)
stmt_lst)) @ (stmt env scope s2)
        in
         let
          for_body_length = (List.length for_body_stmts) in
           s1' @
           [Bra (1 + for_body_length)] @
           for_body_stmts @
           e1' @
           [Bne (-(for_body_length + List.length e1'))]
    | Ast.Return(e) ->
        (expr env e) @ [Rts env.num_formals]
    | Ast.CanSet(can, select_exp, inten)->
        let int_exp = (expr env inten)
        and sel_exp = (expr env select_exp)
        and can_exp = (expr env can) in
        sel_exp @ int_exp @ can_exp @ [Jsr (-7)]
    | Ast.Include(str) ->
        []


  (*
   * Translates a function
   *)
  in let translate env fdecl =
    (* Bookkeeping: FP offsets for locals and args *)
    let num_formals = List.length fdecl.params
    (* we don't currently have locals...*)
    and num_locals = 0 (* List.length *)

    and formal_offsets = (enum (-1) (-2) fdecl.params)
    in
    let formal_offsets' = (List.map (fun (i, s) -> (i, s))
formal_offsets)
    in
    let env = { env with local_idx = string_map_pairs StringMap.empty
formal_offsets';
                         num_formals = num_formals }

    in
      [Ent num_locals] @                                  (* Entry:
allocate space for locals *)
       (List.concat (List.map (stmt env "*local*") fdecl.body)) @
(* Body *)
       [Lit 0; Rts num_formals]                              (* Default
- return 0 *)

  in let env = {
    function_idx  = function_indexes;
    global_idx    = StringMap.empty; (* global_indexes; *)
    local_idx     = StringMap.empty;
    num_formals   = 0
  } in

  (* Compile the global statement list *)
  let glob_stmts = (List.concat (List.map (stmt env "*global*")
stmt_lst)) in
  let main_func_call =
    try
      [Jsr (StringMap.find "main" function_indexes)]
    with Not_found -> []
  in
  (* Compile the functions, and prepend compiled global statements and
Hlt *)
  let func_bodies = (glob_stmts @ main_func_call) :: [Hlt] :: List.map
(translate env) func_decls 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
```

```
    {
      num_globals = List.length (StringMap.bindings env.global_idx);
      (* 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));
      glob_hash = glob_ht;
      glob_hash_counter = hash_counter;
    }
```

8.10 bytecode.ml

```
(* FILENAME :  bytecode.ml
 * AUTHOR(S):  Joe Lee (jyl2157), Dmitriy Gromov (dg2720)
 * PURPOSE  :  Specify assembly operators, type definition of prog
 *             which is returned by Compiler.translate, and a string
 *             representation (string_of_prog).
 *)

open Ast
open Hashtypes


type bstmt =
    Lit of int          (* Push a literal *)
  | Drp                 (* Discard a value *)
  | Bin of Ast.op       (* Perform arithmetic on top of stack *)
  | Lod of int          (* Fetch global variable *)
  | Str of int          (* Store global variable *)
  | 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          (* PushFP, FP->SP, SP+=i *)
  | Rts of int          (* Restore FP, SP, consume formals, push result
*)
  | Beq of int          (* Branch relative if top-of-stack is zero *)
  | Bne of int          (* Branch relative if top-of-stack is non-zero
*)
  | Bra of int          (* Branch relative *)
  | Lct of int          (* Load complex type by absolute address *)
  | CAtr of Ast.attr    (* Get Canvas Attribute*)
  | Hlt                 (* Terminate *)
 *)


type prog = {
  num_globals : int;    (* Number of global variables *)
  text : bstmt array;   (* Code for all the functions *)
  (* global hash table initially populated by compiler *)
  glob_hash : (int, Hashtypes.ct) Hashtbl.t;
  glob_hash_counter : int ref;
}


let string_of_prog prog =
  let rec string_of_prog_helper s = function
      []          -> s
    | hd :: tail  ->
        let s =
          match hd with
              Lit(i)      -> s ^ "Lit " ^ string_of_int i ^ "\n"
            | Drp         -> s ^ "Drp\n"
            | Bin(op)     -> s ^ "Bin\n"
            | Lod(i)      -> s ^ "Lod " ^ string_of_int i ^ "\n"
            | Str(i)      -> s ^ "Str " ^ string_of_int i ^ "\n"
            | Lfp(i)      -> s ^ "Lfp " ^ string_of_int i ^ "\n"
            | Sfp(i)      -> s ^ "Sfp " ^ string_of_int i ^ "\n"
            | Jsr(i)      -> s ^ "Jsr " ^ string_of_int i ^ "\n"
            | Ent(i)      -> s ^ "Ent " ^ string_of_int i ^ "\n"
            | Rts(i)      -> s ^ "Rts " ^ string_of_int i ^ "\n"
            | Beq(i)      -> s ^ "Beq " ^ string_of_int i ^ "\n"
            | Bne(i)      -> s ^ "Bne " ^ string_of_int i ^ "\n"
            | Bra(i)      -> s ^ "Bra " ^ string_of_int i ^ "\n"
            | Lct(i)      -> s ^ "Lct " ^ string_of_int i ^ "\n"
            | CAtr(a)     -> s ^ "Catr " ^ Ast.string_of_attr a ^ "\n"
            | Hlt         -> s ^ "Hlt\n"
        in string_of_prog_helper s tail
  in string_of_prog_helper "" (Array.to_list prog.text)
```

8.11 execute.ml

```
(* FILENAME :  execute.ml
 * AUTHOR(S):  Joe Lee (jyl2157), Dmitriy Gromov (dg2720)
 * PURPOSE  :  Execute bytecode returned from Compile.translate
 *)

open Ast
open Bytecode
open Hashtypes
open Canvas

(* stack type can be either Int (value) or Address (pointer) *)
type stack_t =
    IntValue of int
  | Address of int


let execute_prog prog debug_flag =
  (* wrapper functions around extracting types from the stack *)
  let pop_int = function
      IntValue(i) -> i
    | Address(i)  -> raise (Failure ("Expected an int but popped an
address."))
  and pop_address_val = function
      IntValue(i) -> raise (Failure ("Expected an address but popped an
int."))
    | Address(i) -> (Hashtbl.find prog.glob_hash i)
  in
  let stack = Array.make 1024 (IntValue 0)
  and globals = Array.make prog.num_globals (IntValue 0)
  in
  let get_pnts sel_type soff canv =
      (* This match should be on some sort of enum *)
      let h = (Canvas.height canv -1)
      and w = (Canvas.width canv -1) in
      ( match sel_type with
          1 ->
            let x = pop_int stack.(soff - 1)
            and y = pop_int stack.(soff) in
            Canvas.select_point x y
        | 2 ->
            let x1 = pop_int stack.(soff - 3)
            and x2 = pop_int stack.(soff - 2)
            and y1 = pop_int stack.(soff - 1)
            and y2 = pop_int stack.(soff) in
            Canvas.select_rect x1 x2 y1 y2
        | 3 ->
            let x = pop_int stack.(soff - 2)
            and y1 = pop_int stack.(soff - 1)
            and y2 = pop_int stack.(soff) in
            Canvas.select_hslice x y1 y2
        | 4 ->
            let x1 = pop_int stack.(soff - 2)
            and x2 = pop_int stack.(soff - 1)
            and y  = pop_int stack.(soff) in
            Canvas.select_vslice x1 x2 y
        | 5 ->
            let x = pop_int stack.(soff) in
            Canvas.select_hslice_all x w
        | 6 ->
            let y = pop_int stack.(soff) in
            Canvas.select_vslice_all y h
        | 7 ->
            Canvas.select_all h w
        | 8 ->
            let op_id = pop_int stack.(soff)
            and limit = pop_int stack.(soff - 1) in
            (match op_id with
              0 ->
                 let eq x y =
                    Canvas.get x y canv == limit in
                  Canvas.fetch_match 0 h w eq [];
             | 1 ->
                  let neq x y =
                    Canvas.get x y canv != limit in
                  Canvas.fetch_match 0 h w neq [];
             | 2 ->
                  let less x y =
                    Canvas.get x y canv < limit in
                  Canvas.fetch_match 0 h w less [];
             | 3 ->
                  let leq x y =
                    Canvas.get x y canv <= limit in
                  Canvas.fetch_match 0 h w leq [];
             | 4 ->
                  let gt x y =
                    Canvas.get x y canv > limit in
                  Canvas.fetch_match 0 h w gt [];
             | 5 ->
                  let gte x y =
                    Canvas.get x y canv >= limit in
                  Canvas.fetch_match 0 h w gte [];

            | _ -> raise (Failure("Invalid Select: SS should catch
this")))
        | _ ->
          raise (Failure("Invalid Select: SS should catch this")) )
  and debug s =
    if debug_flag then print_string s
  in
    debug ("DEBUG: num_globals is " ^ string_of_int prog.num_globals ^
"\n");
    try
      let rec exec fp sp pc =
        debug ("DEBUG: fp=" ^ (string_of_int fp) ^ ", sp=" ^
(string_of_int sp) ^ ", pc=" ^ (string_of_int pc) ^ ":  ");
        match prog.text.(pc) with
```

```
          Lit i ->
            stack.(sp) <- IntValue i;
            debug ("Lit " ^ string_of_int i ^ "\n");
            exec fp (sp+1) (pc+1)
        | Lct i ->
            stack.(sp) <- Address i;
            debug ("Lct " ^ string_of_int i ^ "\n");
            exec fp (sp+1) (pc+1)
        | Drp ->
            debug ("Drp " ^ "\n");
            exec fp (sp-1) (pc+1)
        | Bin op ->
            let op1 =
              (match stack.(sp-2) with
                   IntValue(i)     -> Hashtypes.Int(i)
                 | Address(i) -> (Hashtbl.find prog.glob_hash i) (* add
error handling *)
              )
            and op2 =
              (match stack.(sp-1) with
                   IntValue(i)     -> Hashtypes.Int(i)
                 | Address(i) -> (Hashtbl.find prog.glob_hash i) (* add
error handling *)
              )
            in
              (stack.(sp-2) <-
               (let boolean b = if b then 1 else 0
                and bool_of_int i = if i > 0 then true else false
                in
                  (match (op1, op2) with
                       (Hashtypes.Int(i), Hashtypes.Int(j)) ->
                         (match op with
                              Plus ->
                                debug("Bin +: i=" ^ string_of_int i ^ "
j=" ^ string_of_int j ^ "\n");
                                IntValue (i + j)
                            | Minus ->
                                debug("Bin -: i=" ^ string_of_int i ^ "
j=" ^ string_of_int j ^ "\n");
                                IntValue (i - j)
                            | Times ->
                                debug("Bin *: i=" ^ string_of_int i ^ "
j=" ^ string_of_int j ^ "\n");
                                IntValue (i * j)
                            | Divide ->
                                debug("Bin /: i=" ^ string_of_int i ^ "
j=" ^ string_of_int j ^ "\n");
                                IntValue (i / j)
                            | Mod ->
```

```
                                debug("Bin mod: i=" ^ string_of_int i ^
" j=" ^ string_of_int j ^ "\n");
                                IntValue (i mod j)
                            | Eq ->
                                debug("Bin eq: i=" ^ string_of_int i ^
" j=" ^ string_of_int j ^ "\n");
                                IntValue (boolean (i = j))
                            | Neq        ->
                                debug("Bin neq: i=" ^ string_of_int i ^
" j=" ^ string_of_int j ^ "\n");
                                IntValue (boolean (i != j))
                            | Lt         ->
                                debug("Bin <: i=" ^ string_of_int i ^ "
j=" ^ string_of_int j ^ "\n");
                                IntValue (boolean (i < j))
                            | Gt         ->
                                debug("Bin >: i=" ^ string_of_int i ^ "
j=" ^ string_of_int j ^ "\n");
                                IntValue (boolean (i > j))
                            | Leq        ->
                                debug("Bin <=: i=" ^ string_of_int i ^
" j=" ^ string_of_int j ^ "\n");
                                IntValue (boolean (i <= j))
                            | Geq        ->
                                debug("Bin >=: i=" ^ string_of_int i ^
" j=" ^ string_of_int j ^ "\n");
                                IntValue (boolean (i >= j))
                            | Or         ->
                                debug("Bin ||: i=" ^ string_of_int i ^
" j=" ^ string_of_int j ^ "\n");
                                IntValue (boolean ((bool_of_int i) ||
(bool_of_int j)))
                            | And        ->
                                debug("Bin &&: i=" ^ string_of_int i ^
" j=" ^ string_of_int j ^ "\n");
                                IntValue (boolean ((bool_of_int i) &&
(bool_of_int j)))
                            | Mask ->
                                raise( Failure("Mask is not valid for
bools. SS should catch this"))
                          )
                      | (Hashtypes.Bool(b1), Hashtypes.Bool(b2)) ->
                          (match op with
                              Eq ->
                                debug("Bin eq: b1=" ^ string_of_bool b1
^ " b2=" ^ string_of_bool b2 ^ "\n");
                                Hashtbl.add prog.glob_hash
!(prog.glob_hash_counter) (Hashtypes.Bool (b1 = b2));
```

```
                                let ret_val = Address
!(prog.glob_hash_counter) in
                                  prog.glob_hash_counter :=
!(prog.glob_hash_counter)+1;
                                  ret_val
                            | Neq ->
                                debug("Bin neq: b1=" ^ string_of_bool
b1 ^ " b2=" ^ string_of_bool b2 ^ "\n");
                                Hashtbl.add prog.glob_hash
!(prog.glob_hash_counter) (Hashtypes.Bool (b1 != b2));
                                let ret_val = Address
!(prog.glob_hash_counter) in
                                  prog.glob_hash_counter :=
!(prog.glob_hash_counter)+1;
                                  ret_val
                            | Or ->
                                debug("Bin ||: b1=" ^ string_of_bool b1
^ " b2=" ^ string_of_bool b2 ^ "\n");
                                Hashtbl.add prog.glob_hash
!(prog.glob_hash_counter) (Hashtypes.Bool (b1 || b2));
                                let ret_val = Address
!(prog.glob_hash_counter) in
                                  prog.glob_hash_counter :=
!(prog.glob_hash_counter)+1;
                                  ret_val
                            | And ->
                                debug("Bin &&: b1=" ^ string_of_bool b1
^ " b2=" ^ string_of_bool b2 ^ "\n");
                                Hashtbl.add prog.glob_hash
!(prog.glob_hash_counter) (Hashtypes.Bool (b1 && b2));
                                let ret_val = Address
!(prog.glob_hash_counter) in
                                  prog.glob_hash_counter :=
!(prog.glob_hash_counter)+1;
                                  ret_val
                            | _ ->
                                raise (Failure ("Binop not supported
for boolean types."))
                          )
                     | (Hashtypes.String(s1), Hashtypes.String(s2)) ->
                         (match op with
                              Plus ->
                                (* + operator for string operands is a
                                 * concatenation *)
                                debug("Bin +: string1=" ^ s1 ^ "
string2=" ^ s2 ^ "\n");
                                Hashtbl.add prog.glob_hash
!(prog.glob_hash_counter) (Hashtypes.String (s1 ^ s2));
```

```
                                let ret_val = Address
!(prog.glob_hash_counter) in
                                  prog.glob_hash_counter :=
!(prog.glob_hash_counter) + 1;
                                  ret_val
                           | _ ->
                                raise (Failure ("Binop not supported
for string types."))
                         )
                     | (Hashtypes.Canvas(c1), Hashtypes.Canvas(c2)) ->
                       ( match op with
                         Mask ->
                           debug("Canvas 1\n: "
^(Hashtypes.string_of_ct true (Hashtypes.Canvas(c1))   ) ^ "\n");
                           debug("Canvas 1\n: "
^(Hashtypes.string_of_ct true (Hashtypes.Canvas(c2)) ) ^ "\n");
                           Hashtbl.add prog.glob_hash
!(prog.glob_hash_counter) (Hashtypes.Canvas (Canvas.mask c1 c2));
                           let ret_val = Address
!(prog.glob_hash_counter) in
                                 prog.glob_hash_counter :=
!(prog.glob_hash_counter)+1;
                                 ret_val
                         | _ ->
                              raise (Failure ("Binop not supported for
canvas types."))
                        )

                    | (_, _) ->
                         (* This shouldn't happened if the SS gets to
it first *)
                         raise (Failure ("Binop not supported on those
operand types."))
                  )));
                  exec fp (sp-1) (pc+1)
        | Lod i ->
            stack.(sp) <- globals.(i);
            debug ("Lod " ^ string_of_int i ^ " Global=" ^
                   (match globals.(i) with
                        IntValue(j) -> "Int value " ^ string_of_int j
                      | Address(j) -> "Pointer to address " ^
string_of_int j
                   ) ^ "\n");
            exec fp (sp+1) (pc+1)
        | Str i ->
(*
            (match (stack.(sp-1), globals.(i)) with
                 Address(j), Address(k) ->
                   if j != k then
```

```
                    (* if assigning a different pointer to a hash
pair, no
                     * longer need the old hash pair, so remove it *)
                    (); (*Hashtbl.remove prog.glob_hash k; *)
                  globals.(i) <- stack.(sp-1)
              | _ ->
                  globals.(i) <- stack.(sp-1));
 *)
            globals.(i) <- stack.(sp-1);
            debug ("Str " ^ string_of_int i ^ "\n");
            exec fp sp (pc+1)
        | Lfp i ->
(*
            (match (stack.(fp+i), stack.(sp)) with
                 Address(j), Address(k) ->
                   if j != k then
                     (* if over-writing a local, remove hash for
                      * the local being overwritten *)
                     (* if assigning a different pointer to a hash
pair, no
                      * longer need the old hash pair, so remove it *)
                     Hashtbl.remove prog.glob_hash k;
                   stack.(sp) <- stack.(fp+i)
               | _ ->
                   stack.(sp) <- stack.(fp+i));
 *)
            stack.(sp) <- stack.(fp + i);
            debug ("Lfp " ^ string_of_int i ^ "\n");
            exec fp (sp+1) (pc+1)
        | Sfp i ->
            stack.(fp+i) <- stack.(sp-1);
            debug ("Sfp " ^ string_of_int i ^ "\n");
            exec fp (sp+1) (pc+1)
        (* here Jsr -1, refers to OutputC functionality *)
        | CAtr atr ->
            debug ("CAtr ");
            let canv_id = stack.(sp-1) in
              let canv = (match pop_address_val canv_id with
                Hashtypes.Canvas(c) -> c
              | _ -> raise (Failure ("Catr needs to be given a
canvas"))) in

            let result =
              (
                match atr with
                  Ast.W -> Canvas.width canv
                | Ast.H -> Canvas.height canv
                | Ast.G -> Canvas.granularity canv
              ) in
```

```
             stack.(sp-1) <- IntValue result;
             exec fp sp (pc+1)

        | Jsr(-1) ->
             debug ("Jsr -1" ^ "\n");
             let lookup =
               (match stack.(sp-1) with
                    IntValue(i) -> Hashtypes.Int(i)
                  | Address(i) ->
                      try (Hashtbl.find prog.glob_hash i)
                      with Not_found ->
                        (* add error handling *)
                        raise(Failure("Jsr -1: No value found at address
" ^ string_of_int i))
               ) in
             let render =
               (match stack.(sp-2) with
                   IntValue(i) -> raise (Failure ("Jsr -1: Render should
be a boolean."))
                 | Address(i) ->
                     match (Hashtbl.find prog.glob_hash i) with
                         Hashtypes.Bool(b) -> b
                       | _ -> raise(Failure ("Jsr -1: Render should be a
boolean."))
               ) in
             print_endline (Hashtypes.string_of_ct render lookup);
             exec fp sp (pc+1)
        | Jsr(-2) ->
             debug ("Jsr -2");
             let lookup =
               (match stack.(sp-1) with
                    IntValue(i) -> Hashtypes.Int(i)
                  | Address(i) -> (Hashtbl.find prog.glob_hash i) (* add
error handling *)
               ) in
             let render =
               (match stack.(sp-3) with
                   IntValue(i) -> raise (Failure ("Render should be a
boolean"))
                 | Address(i) -> match (Hashtbl.find prog.glob_hash i)
                   with
                     Hashtypes.Bool(b) -> b (* add error handling *)
                   | _ -> raise (Failure("Jsr -2 expected a boolean
render but got a different type."))
               ) in
              let filename =
               ( match (pop_address_val stack.(sp-2)) with
                   Hashtypes.String(s) ->
                                        (match lookup with
                                          Hashtypes.Canvas(c) ->
Canvas.make_name s render
                                        | _ -> s )

                | _ ->
                    raise (Failure("Jsr -2 expected a string filepath
but got a different type."))
              ) in

            let oc = open_out filename in
              output_string oc ( (Hashtypes.string_of_ct render lookup)
^ "\n" );
            exec fp sp (pc+1)
        | Jsr(-3) ->
            (* CANVAS LOADING *)
            debug ("Jsr -2" ^ "\n");
            let gran_val = pop_int(stack.(sp-1))
            and path =
              match (pop_address_val stack.(sp-2)) with
                  Hashtypes.String(s) -> s
                | _ ->
                    raise (Failure("Jsr -2 expected a string filepath
but got a different type."))
            in
            let granularity = string_of_int gran_val
            in
              let filename_parts = Str.split (Str.regexp "/") path in
                let filename =
                  match Str.string_match (Str.regexp ".+.i")  (List.hd
(List.rev filename_parts)) 0 with
                    false ->
                      debug ("Trying to open: " ^ "../tmp/" ^ List.hd
(List.rev filename_parts) ^ ".i" ^ "\n");
                      let comm = "python util/load_img.py " ^ path ^ "
" ^ granularity in
                      let _ = Sys.command (comm)
                      in ();
                      "../tmp/" ^ List.hd (List.rev filename_parts) ^
".i"
                  | true ->
                      debug ("Trying to open raw: " ^ path ^ "\n");
                      path
              in
              Hashtbl.add prog.glob_hash !(prog.glob_hash_counter)
                (Hashtypes.Canvas (Canvas.load_canvas filename
gran_val));
              let ret_val = Address !(prog.glob_hash_counter) in
                prog.glob_hash_counter := !(prog.glob_hash_counter)+1;
                stack.(sp-1) <- ret_val;
                exec fp sp (pc+1)
        | Jsr(-4) ->
            (* BLANK *)
            debug ("Jsr -4" ^ "\n");
            let h_val = (pop_int stack.(sp-3))
            and w_val = (pop_int stack.(sp-2))
            and g_val = (pop_int stack.(sp-1))
            in
              Hashtbl.add prog.glob_hash !(prog.glob_hash_counter)
                (Hashtypes.Canvas (Canvas.blank h_val w_val g_val 0));
              let ret_val = Address !(prog.glob_hash_counter) in
                prog.glob_hash_counter := !(prog.glob_hash_counter)+1;
                stack.(sp-1) <- ret_val;
                exec fp sp (pc+1)
       | Jsr (-5) ->
            (* SHIFT *)
            debug ("Jsr -5" ^ "\n");
             let existing = match (pop_address_val stack.(sp-1)) with
                Hashtypes.Canvas(c) -> c
              | _ -> raise(Failure("Jsr -6: Expected canvas type."))
             and dir = pop_int stack.(sp-2)
             and dist = pop_int stack.(sp-3)
           in
            let shifted = (Canvas.shift existing dir dist) in
            Hashtbl.add prog.glob_hash !(prog.glob_hash_counter)
                (Hashtypes.Canvas (shifted));
              let ret_val = Address !(prog.glob_hash_counter) in
                prog.glob_hash_counter := !(prog.glob_hash_counter)+1;
                stack.(sp-1) <- ret_val;
                exec fp sp (pc+1)
       | Jsr (-6) ->
            (* SELECT *)
            debug ("Jsr -6: - Select Piece of Canvas" ^ "\n");
            let existing = match (pop_address_val stack.(sp-1)) with
                Hashtypes.Canvas(c) -> c
              | _ -> raise(Failure("Jsr -6: Expected canvas type.")) in
            let sel_type = (pop_int stack.(sp-2)) in
            let stack_offset = sp-3 in
            let pnts = get_pnts
                        sel_type stack_offset existing in
            let selected = (Canvas.select_rect_from_list pnts existing)
in

            Hashtbl.add prog.glob_hash !(prog.glob_hash_counter)
(Hashtypes.Canvas(selected));
            let ret_val = Address !(prog.glob_hash_counter) in
              prog.glob_hash_counter := !(prog.glob_hash_counter)+1;
              stack.(sp-1) <- ret_val;
              exec fp sp (pc+1)
        | Jsr (-7) ->
            (* SET POINT *)
            debug ("Jsr -7: - Set point" ^ "\n");
            let existing = match (pop_address_val stack.(sp-1)) with
                Hashtypes.Canvas(c) -> c
              | _ -> raise(Failure("Jsr -6: Expected canvas type."))
            and set_val = (pop_int stack.(sp-2))
            and sel_type = (pop_int stack.(sp-3))
            and stack_offset = sp-4 in

            let pnts = get_pnts sel_type stack_offset existing in
            Canvas.set_from_list existing set_val pnts;

            (*
               Don't actually need to set the canvas back to what it
was because it's being modified directly.
                let modified_can = (Canvas.set_from_list existing
set_val pnts) in
               Hashtbl.add prog.glob_hash !(prog.glob_hash_counter)
(Hashtypes.Canvas(modified_can));
            *)

            exec fp sp (pc + 1)
        | Jsr i ->
            stack.(sp) <- IntValue (pc + 1);
            debug ("Jsr " ^ string_of_int i ^ "\n");
            exec fp (sp+1) i
        | Ent i ->
            stack.(sp) <- IntValue (fp);
            debug ("Ent " ^ string_of_int i ^ "\n");
            exec sp (sp+i+1) (pc+1)
        | Rts i ->
            let new_fp = pop_int stack.(fp)
            and new_pc = pop_int stack.(fp-1)
            in
              stack.(fp-i-1) <- stack.(sp-1);
              debug ("Rts " ^ string_of_int i ^ "\n");
              exec new_fp (fp-i) new_pc
        | Beq i ->
            debug ("Beq " ^ string_of_int i ^ "\n");
            exec fp (sp-1)
              (pc +
               if (match stack.(sp-1) with
                       IntValue(k) -> (k = 0)
                     | Address(k) -> match (Hashtbl.find prog.glob_hash
k) with
                           Hashtypes.Bool(b) -> not b
                         | _ -> raise(Failure("Beq operation: Address
lookup resulted in a non-boolean type.")))
```

```
               then i
               else 1)
        | Bne i ->
            debug ("Bne " ^ string_of_int i ^ "\n");
            exec fp (sp-1)
              (pc +
               if (match stack.(sp-1) with
                       IntValue(k) -> (k != 0)
                     | Address(k) -> match (Hashtbl.find prog.glob_hash
k) with
                           Hashtypes.Bool(b) -> not b
                         | _ -> raise(Failure("Bne operation: Address
lookup resulted in a non-boolean type.")))
               then i
               else 1)
        | Bra i ->
            debug ("Bra " ^ string_of_int i ^ "\n");
            exec fp sp (pc+i)
        | Hlt -> ()
      in exec 0 0 0
    with e -> (* catch all exceptions *)
      Printf.eprintf "Runtime error: %s\n" (Printexc.to_string e);
```

8.12 canvas.ml

```
(* AUTHOR(S):  Dmitriy Gromov (dg2720), Joe Lee (jyl2157)
 * PURPOSE  :  Canvas functions (loading, preprocessing, blank,
 *             string_of_canvas, etc...).
 *)

module IntMap =
  Map.Make(
    struct type t = int
    let compare = compare end
  )

let make_map vals =
  let rec add_val the_map = function
     v :: vs -> (add_val (IntMap.add (IntMap.cardinal the_map) v
the_map) vs)
    | [] -> the_map
  in add_val IntMap.empty vals

(* Default mapping taken from
http://incredibleart.org/links/ascii/new_faq.html *)
let default_map =
   make_map (['.';'`';'^';':';'"';';';'~';
            '-';'_';'+';'<';'>';'i';'!'
            ;'l';'I';'l';'?';'|';
```

```
              '(';')';'1';'{';'}';'[';']';
              'r';'c';'v';'u';'n';'x';'z';'j'
              ;'f';'t';'L';'C';'J';'U';'Y';'X'
              ;'Z';'O';'0';'Q';'o';'a';'h';'k'
              ;'b';'d';'p';'q';'w';'m';'*';'W';
              'M';'B';'8';'&';'%';'$';'#';'@'])

(* If 0 or Max Granularity Then min or max, otherwise even dist *)
let get_char_for_intensity intensity granularity map =
  let max_int = (granularity - 1) in
  let card = (IntMap.cardinal map) in
  let idx =
    match intensity == 0 with
      true -> 0
    | false -> match intensity == max_int with
        true -> ((card-1))
      | false  -> let divisor = ( card - 2 ) / (granularity - 2) in
                    let x = (card - 2 ) / divisor in
                      x * intensity
  in
  Char.escaped (IntMap.find idx map);


type canvas =
{
  data: int array array;
  gran: int;
};;

type stypes =
  POINT
| RECT
| VSLICE
| HSLICE
| VSLICE_ALL
| HSLICE_ALL
| ALL
| BOOL

let select_type = function
  POINT -> 1
| RECT -> 2
| VSLICE -> 3
| HSLICE -> 4
| VSLICE_ALL -> 5
| HSLICE_ALL -> 6
| ALL -> 7
| BOOL -> 8
```

```
type dir =
   UP
 | LEFT
 | DOWN
 | RIGHT

let get_dir = function
   0 -> (UP)
 | 1 -> (LEFT)
 | 2 -> (DOWN)
 | 3 -> (RIGHT)
 | _ -> raise(Failure ("Not a valid Direction"))


(* Blank Function  *)
let blank height width granularity default=
{
    data = Array.make_matrix height width default;
    gran = granularity
}

(* Make File Name *)
let make_name name render =
  if render then
    name
  else
    String.concat "" [name; ".i"]

(* Print Row *)
let string_of_row row render the_map gran =
  match render with
      false -> String.concat " " (Array.to_list (Array.map
string_of_int row ))
    | true ->  String.concat "" (Array.to_list (Array.map (fun x ->
match x with

(-1) -> " "
                                                                      |
_ ->  (get_char_for_intensity

x gran the_map)) row))



let string_of_canvas can map render =
  String.concat "\n" (Array.to_list(Array.map (fun r -> string_of_row r
render map can.gran ) can.data))
```

```
(* CANVAS ATTRIBUTES *)
let height can =
  Array.length can.data

let width can =
  Array.length can.data.(0)

let granularity can =
  can.gran

(* END CANVAS ATTRIBUTES *)

let create_blank_from_existing existing default =
   blank (height existing) (width existing)  (granularity existing)
default


(* SELECT *)
let get x y can =
  if x < (height can) && x >= 0
     && y < (width can) && y >= 0
  then
    can.data.(x).(y)
  else
    raise (Failure("(" ^ string_of_int x ^ ", " ^ string_of_int y ^ ")
is out of bounds of canvas"))
(* MASK *)
let set x y intensity can =
  if x < (height can) && x >= 0
     && y < (width can) && y >= 0
  then
    can.data.(x).(y) <- intensity

let accept_all x y =
  true


let rec fetch_row x1 y1 y2 acc cond=
    match y1 <= y2 with
      true ->
           if (cond x1 y1) then
                (x1, y1) :: fetch_row x1 (y1+1) y2 acc cond
           else fetch_row x1 (y1+1) y2 acc cond

    | false -> []

let rec fetch_box x1 x2 y1 y2 acc =
    match x1 <= x2 with
```

```
      true -> (fetch_row x1 y1 y2 acc accept_all) @ fetch_box (x1+1) x2
y1 y2 acc
      | false -> []

let rec fetch_match x h w cond acc =
    match x <= h with
      true -> (fetch_row x 0 w acc cond) @ fetch_match (x+1) h w cond
acc
      | false -> []

let string_of_point = function
  (x, y) -> string_of_int x ^ " " ^ string_of_int y ;;

let rec print_l = function
    x :: xs -> print_endline (string_of_point x);
             print_l xs
  | [] -> ""

let rec set_point can intensity = function
    x :: xs -> (match (x) with
                (i,j) ->
                  (match intensity >= 0 with
                    true -> set i j intensity can
                    | false -> ());
                  set_point can intensity xs;
                )
  | [] -> ()
;;

(* let set_points_int points can intensity =
  let l = (fetch_match can []) in
    set_point can intensity l
 *)
let set_rect_can l old_can new_can=
  let rec set_point = function
    x :: xs -> (match x with
                  (i,j) ->
                    let selected = get i j old_can in
                      match selected >= 0 with
                        true -> set i j selected new_can
                      | false -> ()
                );
                set_point xs;
  | [] -> ()
  in
  set_point (l);
  (new_can)

let set_from_list can intensity pnts =
  set_point can intensity pnts

let select_rect_from_list l can =
   let blank_slate = create_blank_from_existing can (-1) in
    set_rect_can  l can blank_slate

(*
   The following functions simply return points representing various
boxes
   They are labels for the various select operations.
*)

let select_rect x1 x2 y1 y2 =
    fetch_box x1 x2 y1 y2 []

let select_point x y =
  fetch_box x x y y []

let select_hslice x y1 y2  =
  fetch_box x x y1 y2 []

let select_vslice x1 x2 y  =
  fetch_box x1 x2 y y []

let select_hslice_all x w=
  fetch_box x x 0 (w) []

let select_vslice_all y h =
  fetch_box 0 (h) y y []

let select_all h w=
  fetch_box 0 (h) 0 (w) []

(* END SELECT *)

let mask can1 can2 =
  let blank_slate = create_blank_from_existing can1 (-1) in
    let pl = select_all ((height can2)-1) ((width can2)-1) in
     let cp_can1 = set_rect_can pl can1 blank_slate in
     (set_rect_can  pl can2 cp_can1 )

let shift can dir steps =
  let shifted = create_blank_from_existing can (-1) in
    let rec set_point = function
      x :: xs -> (match (x) with
                 (i,j) ->
                   ( let intensity = get i j can
                     and new_point =
                       (match ( get_dir (dir) ) with
                          UP ->
                            (i - steps, j)
                        | DOWN ->
                            (i + steps, j)
                        | LEFT ->
                            (i, j - steps)
                        | RIGHT ->
                            (i, j + steps)
                        ) in

                      (
                        match new_point with
                          (a, b) ->  set a b intensity shifted
                      )
                    )
                );
                set_point xs;
    | [] -> ()
  in
  let l =
    (match ( get_dir (dir) ) with
                          UP ->
                          (fetch_box steps (((height can)-1)) 0 ((width
can)-1) [])
                        | DOWN ->
                          (fetch_box 0 (((height can)-1) - steps) 0
((width can)-1) [])
                        | LEFT ->
                          (fetch_box 0 ((height can)-1) steps ((width
can)-1) [])
                        | RIGHT ->
                          (fetch_box 0 (((height can)-1)) 0 (((width
can)-1) - steps) [])
                      ) in
    set_point (l);

  shifted

(* Loads an image from filepath fname, and returns
 *  canvas type int array array *)
let load_canvas fname granularity  =
  let ic = open_in fname in
  let n = in_channel_length ic in
  let s = String.create n in
    really_input ic s 0 n;
    close_in ic;
    let lines = Str.split (Str.regexp("\n")) s  in
    let can = {
```

```
                data = (Array.make_matrix (List.length lines)
(String.length (List.hd lines))) 0;
                gran = granularity;
              }
    in
    let x = ref 0 in
      List.iter (
        fun line ->
          let row = Str.split (Str.regexp(" ")) line in
            can.data.(!x) <- (Array.of_list (List.map int_of_string
row));
            x := !x+1
      ) lines;
      (can);;
```

8.13 ezac.ml

```
(* FILENAME :  ezac.ml
 * AUTHOR(S):  Joe Lee (jyl2157), Dmitriy Gromov (dg2720)
 * PURPOSE  :  Top-level file providing command-line interface to
 *             bytecode executor, compiler, interpreter.
 *)

open Ast
open Parser
open Scanner
open Bytecode
open Ssanalyzer
open Compiler
open Execute
open Preprocess

type action = Ast | StaticSemanticChecker | Interpret | Bytecode |
Compile

let _ =
  let (action, debug_flag, filepath) =
    let param_count = Array.length Sys.argv in
      if param_count > 2 then
        let option = (List.assoc Sys.argv.(1)
            [ ("-a",  (Ast, false));
              ("-s",  (StaticSemanticChecker, false));
              ("-i",  (Interpret, false));
              ("-b",  (Bytecode, false));
              ("-c",  (Compile, false));
              ("-cd", (Compile, true))
            ])
        in (fst option), (snd option), Sys.argv.(2)
      else
        if param_count = 2 then
          (Compile, false, Sys.argv.(1))
        else raise (Failure ("Invalid number of arguments."))
  in
  let preprocessed = Preprocess.run (filepath) in
  let lexbuf = Lexing.from_string preprocessed in
  let program = try
        (Parser.program Scanner.token lexbuf)
      with Parsing.Parse_error ->
        let curr = lexbuf.Lexing.lex_curr_p in
        (* let line = curr.Lexing.pos_lnum in *)
        let cnum = curr.Lexing.pos_cnum - curr.Lexing.pos_bol in
        let tok = Lexing.lexeme lexbuf in
          print_endline (">>> Parse error at character " ^
(string_of_int cnum) ^ ": '" ^ tok ^ "'");
          exit 0;
  in
  let run_ssanalyzer program =
    (try
       let ret = Ssanalyzer.semantic_checker program
       in ret
     with
         TypeException(astexpr1, astexpr2, expected_typ, actual_typ) ->
           print_endline("Type error at subexpression " ^
(Ast.string_of_expr astexpr1) ^ " in expression " ^ (Ast.string_of_expr
astexpr2) ^ ".  Expected type " ^ (Ssanalyzer.string_of_t expected_typ)
^ " but got type " ^ (Ssanalyzer.string_of_t actual_typ ^ "."));
           exit(0)
       | BinopException(astop, astexpr, expected_op) ->
           print_endline("Binop error in expression " ^
(Ast.string_of_expr astexpr) ^ ".  Expected binop " ^
(Ssanalyzer.string_of_t expected_op) ^ " but got binop " ^
(Ast.string_of_op astop));
           exit(0)
       | UndefinedVarException(astexpr) ->
           print_endline("Undefined variable " ^ (Ast.string_of_expr
astexpr));
           exit(0)
       | UndefinedFxnException(fxn_name, astexpr2) ->
           print_endline("Undefined function " ^ fxn_name ^ " in
expression " ^ (Ast.string_of_expr astexpr2));
           exit(0)
       | Failure(s) ->
           print_endline(s);
           exit(0)
    )
  in
```

```
    match action with
        Ast -> let listing = Ast.string_of_program program in
print_string listing
      | StaticSemanticChecker ->
          let _ = run_ssanalyzer program in
            print_endline("Static semantic checker finished with no
errors.")
      | Interpret ->
          print_string "Interpret: nada at the moment" (* ignore
(Interpret.run program) *)
      | Bytecode ->
          let listing = Bytecode.string_of_prog (Compiler.translate
program)
          in print_endline listing
      | Compile ->
          let checked_prog = run_ssanalyzer program
          in
          let program = Compiler.translate checked_prog
          in Execute.execute_prog program debug_flag
```

8.14 reuse.ml

```
(* FILENAME :  reuse.ml
 * AUTHOR(S):  Joe Lee (jyl2157)
 * PURPOSE  :  Functions for re-use.
 *)

(* given a string, splits it into a list of chars *)
let explode s =
  let rec exp i l =
    if i < 0 then l else exp (i - 1) (s.[i] :: l) in
    exp (String.length s - 1) [];

(* given a list of chars, joins them and returns a string *)
let implode lst =
  let res = String.create (List.length lst) in
  let rec imp i = function
    | [] -> res
    | c :: lst -> res.[i] <- c; imp (i + 1) lst in
    imp 0 lst;

(* debug function to inspect environment *)
let env_to_str m =
  let bindings = StringMap.bindings m in
  let rec print_map_helper s = function
      [] -> s
    | hd :: tl -> print_map_helper ((fst hd) ^ " = " ^ (string_of_int
(snd hd)) ^ "\n" ^ s) tl
  in print_map_helper "" bindings;
```

8.15 Makefile

```
# FILENAME :  Makefile
# AUTHOR(S):  Joe Lee (jyl2157), Dmitriy Gromov (dg2720)
# PURPOSE  :  Makefile for EZ-ASCII project.

OBJS = ast.cmo scanner.cmo parser.cmo sast.cmo ssanalyzer.cmo
preprocess.cmo bytecode.cmo canvas.cmo compiler.cmo  hashtypes.cmo
execute.cmo ezac.cmo

ezac : $(OBJS)
      ocamlc -o ezac str.cma $(OBJS)

scanner.ml : scanner.mll
      ocamllex scanner.mll

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

%.cmo : %.ml
      ocamlc -c $<

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

.PHONY : clean
clean :
      rm -rf *.cmo *.cmi ezac parser.mli parser.ml scanner.ml

# generated by ocamldep *.ml *.mli
ast.cmo :
ast.cmx :
bytecode.cmo : hashtypes.cmo ast.cmo
bytecode.cmx : hashtypes.cmx ast.cmx
canvas.cmo :
canvas.cmx :
compiler.cmo : canvas.cmo hashtypes.cmo bytecode.cmo ast.cmo
compiler.cmx : canvas.cmx hashtypes.cmx bytecode.cmx ast.cmx
execute.cmo : hashtypes.cmo bytecode.cmo ast.cmo
execute.cmx : hashtypes.cmx bytecode.cmx ast.cmx
ezac.cmo : ssanalyzer.cmo scanner.cmo preprocess.cmo parser.cmi
execute.cmo compiler.cmo bytecode.cmo ast.cmo
ezac.cmx : ssanalyzer.cmx scanner.cmx preprocess.cmx parser.cmx
execute.cmx compiler.cmx bytecode.cmx ast.cmx
hashtypes.cmo : canvas.cmo
hashtypes.cmx : canvas.cmx
interpret.cmo : scanner.cmo parser.cmi ast.cmo
interpret.cmx : scanner.cmx parser.cmx ast.cmx
```

```
parser.cmo : ast.cmo parser.cmi
parser.cmx : ast.cmx parser.cmi
preprocess.cmo :
preprocess.cmx :
sast.cmo :
sast.cmx :
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx
ssanalyzer.cmo :
ssanalyzer.cmx :
parser.cmi : ast.cmo
```

8.16 runtests.sh

```
#!/bin/bash

# FILENAME :  runtests.sh
# AUTHOR(S):  Joe Lee (jyl2157)
# PURPOSE  :  Shell script to run tests on executable.
#             Each test in the tests dir is run on the executable
#             with its standard out piped to a .out file.  The .out
#             file is compared with a corresponding .gs (gold standard)
#             file for each test.  If the test fails (output differs),
#             a .diff file is created for developer use.

APP=$(dirname $0)/ezac
globallog=test_ezac.log
testdir=tests
rm -f $globallog
error=0

Check() {
      basename=$(basename $1)
      basename=${basename%.*}

      ezafile=$testdir/${basename}.eza
      reffile=$testdir/${basename}.gs
      outfile=$testdir/${basename}.out
      difffile=$testdir/${basename}.diff

      echo -n "Running $basename..."

      $APP $ezafile > $outfile 2>&1 || {
            echo "Failed: ezac terminated."
            error=1;
            return 1
      }
      diff -b $reffile $outfile > $difffile 2>&1 || {
            echo "Failed: output mismatch."
            error=1;
            return 1
      }

      rm $outfile $difffile
      echo "OK."
}

for file in $testdir/test*.eza
do
        Check $file 2>> $globallog
done

exit $error
```