Crayon

by **Naman** Agrawal, **Vaidehi** Dalmia, **Ganesh** Ravichandran, **David** Smart

Table of Contents

1 Introduction

Crayon is a raster-graphics creation language that simplifies the digital painting of images through code. Raster graphics are images created from two-dimensional arrays of hex codes that represent the rectangular grid of pixels we see for many computer graphics. Inspired by this paradigm, the Crayon programming language provides a way for the user to create any sort of pixelated image (in color) that they can imagine - like the beautiful creations often made on Microsoft Paint.

The crux of Crayon's design is the assigning of pixel color values to an array-like Canvas. In the same way that a painter assigns different colors to sections of a physical canvas, so can a programmer to Crayon's digital canvas. The main objective for the programmer should be the artistic or systematic implementation of color.

1.1 Background

Graphics is often done in one of two manners: rasterization and vectorization. Rasterized images are colored pixel by pixel and thus allow more color blending and granular texture. Vectorized images are colored object by object - where objects are outlines of pieces of an image. Although vector images are more scalable, we found the raster approach to be more interesting, and wanted to explore the programming of pixelated images.

Specifically, we wanted to explore the construction of raster images through a bitmap model of assigning color values to cells in a rectangular area. In our language, we call this a Canvas, and it is represented by a two-dimensional array. In essence, we make the programmer's job as simple as possible : paint colors on the Canvas. This approach reduces the mysticality accompanies drawing on a computer program's UI. With our language, developers should be able to manipulate color values on an image with ease, and design unique programs for graphics creation.

1.2 Goals

1.2.1 Transparency

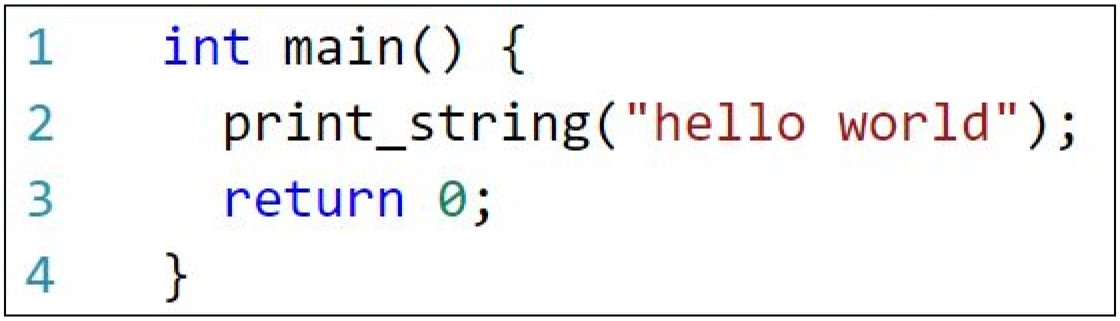
By offering the Canvas type to programmers, we hope to allow the programmer to achieve exactly the image they desire, pixel for pixel. Although this requires somewhat of a learning curve (eased by standard library functions), the unrestricted access to the mapping of pixels will allow for robust programs and libraries to be formed.

1.2.2 Familiarity

The Canvas type is as simple to understand as a two-dimensional array - because that's what it is. As long as programmers are familiar with how access and assignment to a two dimensional array works, painting colors into positions is just that easy. Furthermore, the syntax of our is close to C, reducing the syntactical learning curve for the average programmer.

2 Tutorial

2.1 Writing “Hello World”



```
1   int main() {
2     print_string("hello world");
3     return 0;
4   }
```

A compilable “Hello World”.

The above code is a fully compilable program:

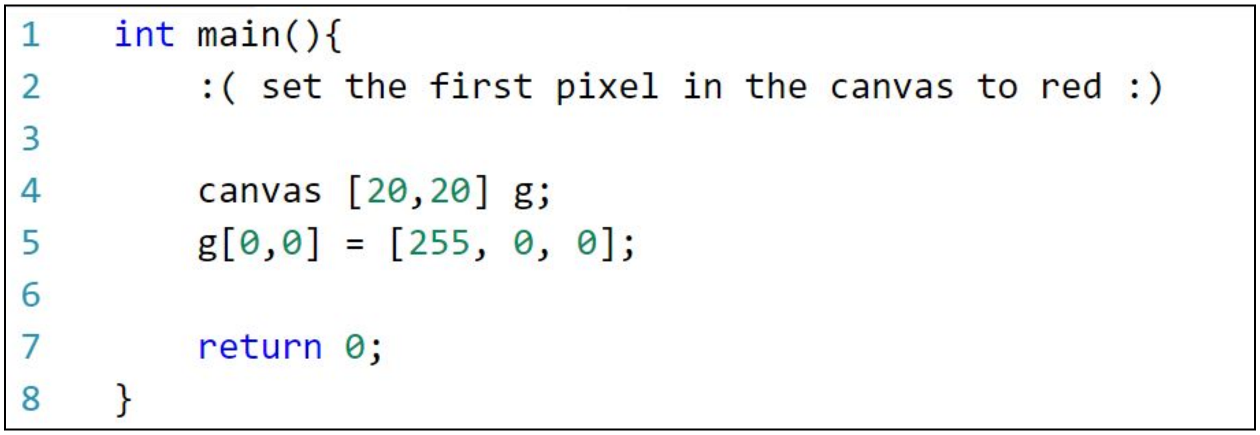
- **Line 1:** defines the main() function that is run automatically in any Crayon file
- **Line 2**: print_string(), with a *String* type as a parameter, will print to the output
- **Line 3:** the return statement, in this case zero, tells the main function to end

2.2 Compiling and running hello_world.py

Please follow the following steps:

- In the home folder, run **make** to compile and link the codebase.
- To compile and run hello_world.cry, outputting to a file, run:
    - **./crayon.native < hello_world.py > out.s**
- To output the result of out.s run **lli out.s**
- That’s it!

2.2 Creating a Canvas Type Variable



```
1   int main(){
2       :( set the first pixel in the canvas to red :)
3
4       canvas [20,20] g;
5       g[0,0] = [255, 0, 0];
6
7       return 0;
8   }
```

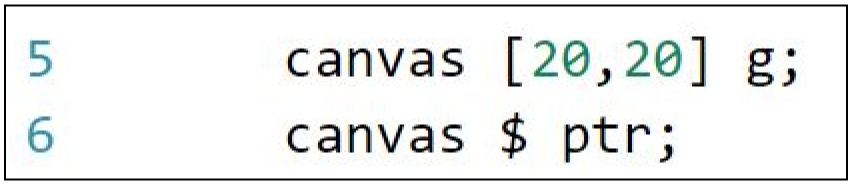
Creating a canvas.

The above code shows how to get started using the Canvas to paint an image:

- **Line 2:** comments are blocked off from the rest of the code by **:(** *string goes here* **:)**
- **Line 4:** the keyword 'canvas' signifies defining a Canvas type variable **g** of size 20x20
    - Within brackets **[x, y]**, the height (x) and width (y) of the Canvas can be defined.
- **Line 5:** accessing a pixel in the canvas is done by using brackets **[x, y]**, where x is the latitudinal index and y is the longitudinal index
    - This value on the right of the equals sign is an RGB color value, represented as an integer array [R, G, B].
- **IMPORTANT**: declare <u>all variables</u> at the top of the method.

2.3 Using Pointers to Change Canvas Pixel Values

Throughout this tutorial, we will refer to the three-dimensional arrays that fill a Canvas as *Pixels*. We'll use this next part to teach you how to use pointers in Crayon. Don't worry - It's easy! **Note:** the steps below can also be applied to array types.



```
    canvas [20,20] g;
    canvas $ ptr;
```

Declaring the pointer.

Firstly, declare a *pointer*, as seen in Line 6. The name of our pointer is **ptr** and is denoted by the pointer type symbol **$**. This contains the memory address of the beginning of the Canvas. In other words, the first value (R of RGB) of the first Pixel.

```
7           ptr = &g;
```

Defining a pointer.

The above assigns the memory address of Canvas g to **ptr.**

```
8           g[0,0] = [255, 0, 0];
9           ~ptr = 0;
```

Dereferencing a pointer.

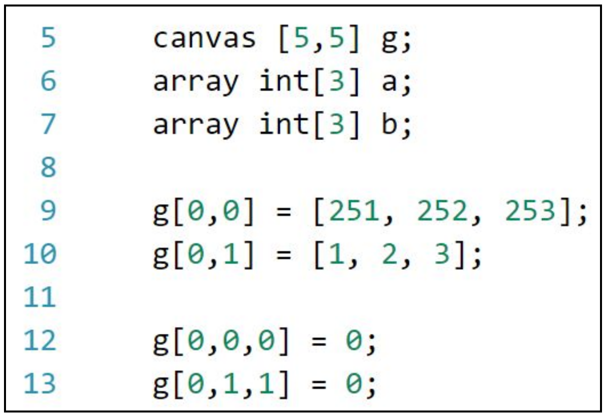
Next, you can *dereference* the pointer to assign values to the memory address represented by the pointer. The ~ symbol achieves this goal. Here, we assign the value of zero to the R value of the pixel - turning a red (255, 0, 0) to a black (0, 0, 0).

```
ptr = @2@ ptr;
~ptr = 255;
```

Pointer arithmetic.

Finally, if you want to access the G and B values, you can use *pointer arithmetic* - in a simple form - to move from one RGB sub-color of each Pixel to the next. This means to go from one R to another Pixel's R to another, you must increment by 3. To do this arithmetic, use the **@ @** symbols with an integer expression in between - even negatives to go backwards!

2.3 Change Canvas Pixel Values the Easy Way

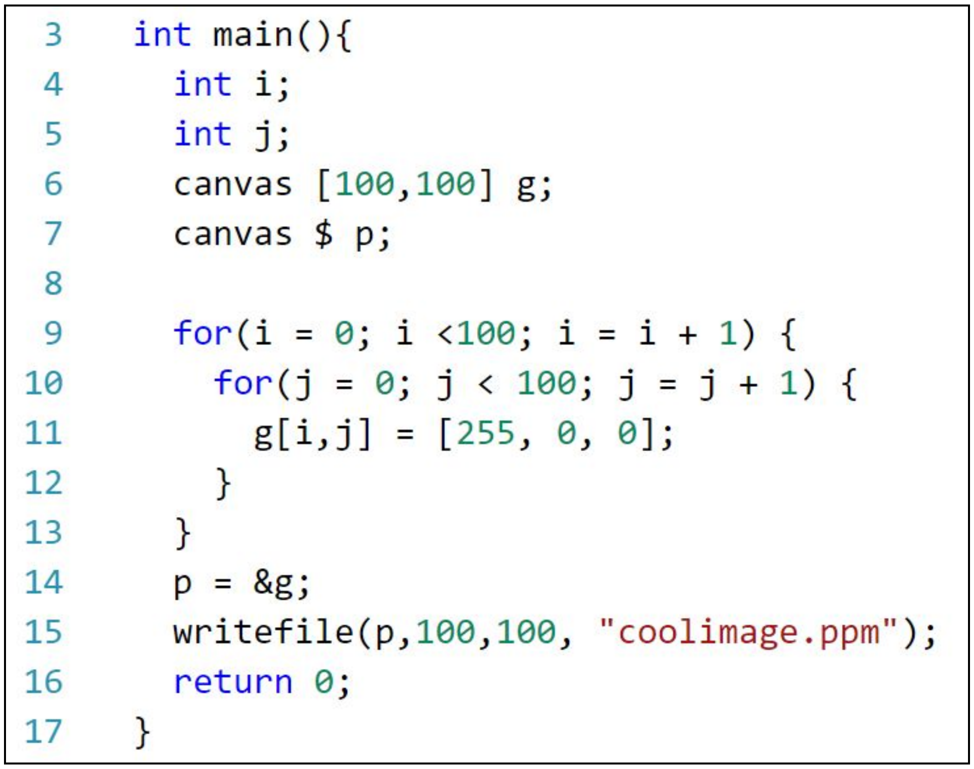


```
    canvas [5,5] g;
    array int[3] a;
    array int[3] b;

    g[0,0] = [251, 252, 253];
    g[0,1] = [1, 2, 3];

    g[0,0,0] = 0;
    g[0,1,1] = 0;
```

Changing pixel values using index.

Yup that's it! In the form **[i, j, k]** where **i** = row, **j** = column, and **k** = the sub-color (R, G, or B) as the index 0, 1, or 2. *:(Sorry about the pointer stuff, but you had to learn somehow :)*

2.4 Exporting a Color-Filled Canvas to a ppm file



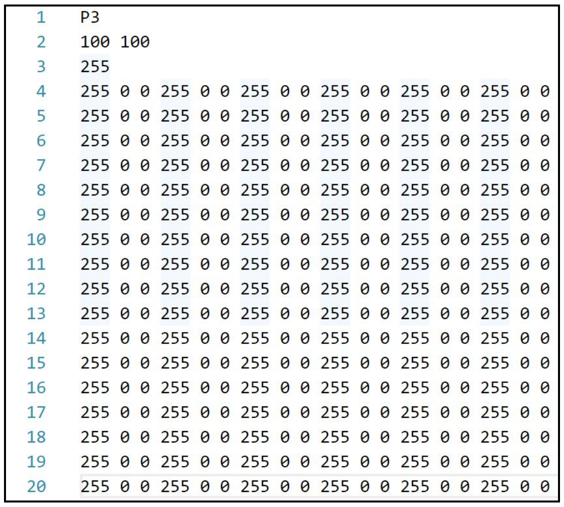
```
int main(){
  int i;
  int j;
  canvas [100,100] g;
  canvas $ p;

  for(i = 0; i <100; i = i + 1) {
    for(j = 0; j < 100; j = j + 1) {
      g[i,j] = [255, 0, 0];
    }
  }
  p = &g;
  writefile(p,100,100, "coolimage.ppm");
  return 0;
}
```

Using *writefile* to visualize a Canvas.

This may look like a big jump from the last section, but it's actually pretty simple. Here's the breakdown:

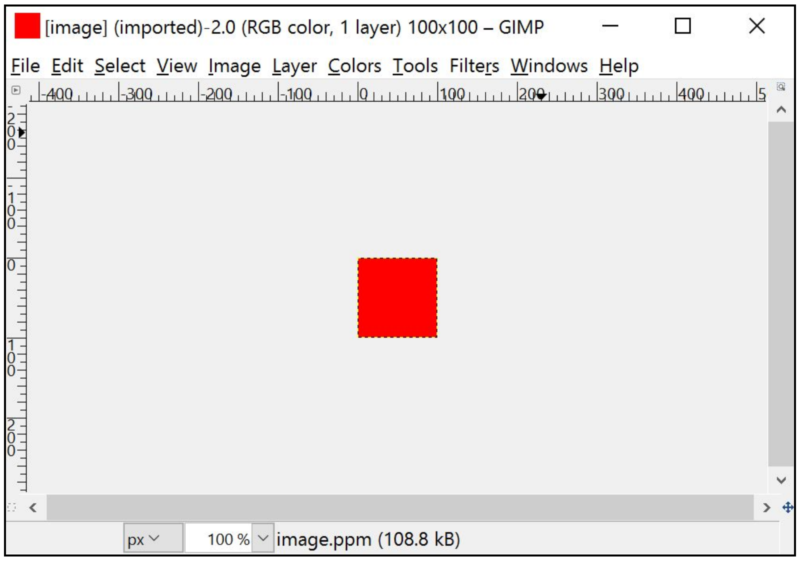
- **Lines 9-12**: use nested for-loops to assign the color red - recall [255, 0, 0] is the Pixel representation for red - to each Pixel in the 100x100 Canvas.
- **Line 15:** the *writefile* function takes a Canvas and outputs a ppm file representation of it. It takes four parameters:
    - *writefile*(canvas $ **canvas_pointer**, int **num_rows**, int **num_cols,** string **filename**)
    - Simply put, it takes a pointer to the Canvas, the width and length of the Canvas, and the output file name.

Below is a sample of the ppm output for the above program.



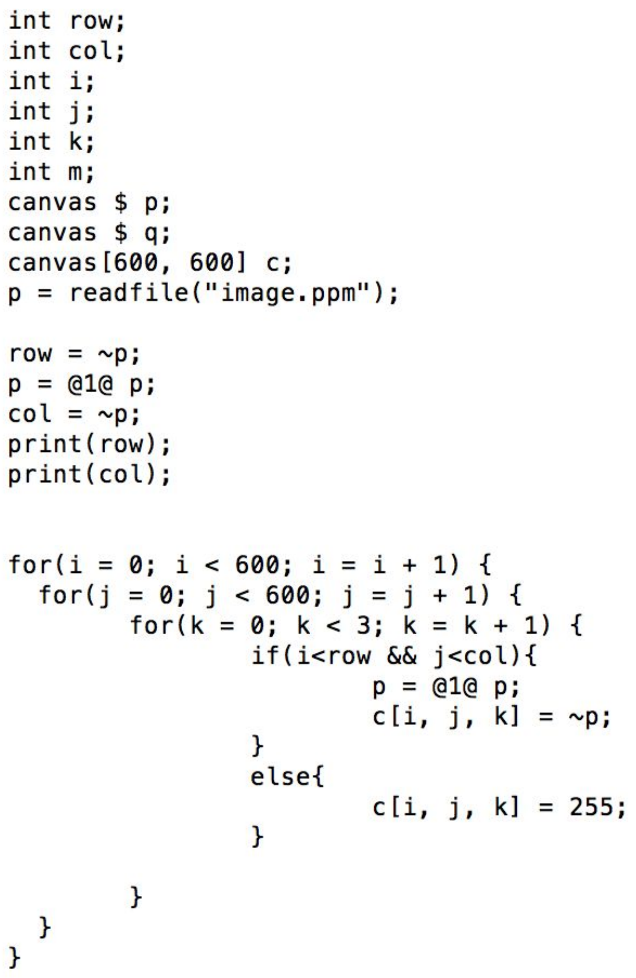
```
P3
100 100
255
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
```

The text version.


[image] (imported)-2.0 (RGB color, 1 layer) 100x100 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
-400 -300 -200 -100 0 100 200 300 400
px
100 %
image.ppm (108.8 kB)

The more human-friendly version.

2.4 Importing a ppm file into a Canvas



```
int row;
int col;
int i;
int j;
int k;
int m;
canvas $ p;
canvas $ q;
canvas[600, 600] c;
p = readfile("image.ppm");

row = ~p;
p = @1@ p;
col = ~p;
print(row);
print(col);


for(i = 0; i < 600; i = i + 1) {
  for(j = 0; j < 600; j = j + 1) {
        for(k = 0; k < 3; k = k + 1) {
                if(i<row && j<col){
                        p = @1@ p;
                        c[i, j, k] = ~p;
                }
                else{
                        c[i, j, k] = 255;
                }

        }
  }
}
```

Using *readfile* to create Canvas from a PPM file.

Now to do the opposite, we can use the counterpart function *readfile* to open a PPM file and read its values. Here's the breakdown:

- **Line 10**: *readfile* takes in the name of a file, which must be in the same directory and in PPM format, and returns a pointer to the first value of the file, which is the number of rows in the image. The second value is the number of columns, and the rest of the values after iterating the pointer are the pixel colors.
- **Line 19-30:** Three for-loops can be used along with the color accessing functionality to set the corresponding color values to a Canvas type by moving the pointer through the data that was returned by *readfile*. If the Canvas is too large, you can fill the rest of the space with white as shown in Line 23.

3 Language Reference Manual

3.1 Lexical Elements

3.1.1 Identifiers

Identifiers are strings used to name variables and functions. The first character of an identifier must be a letter of the alphabet and can be followed by alphanumeric characters and underscores. They are case sensitive. Identifiers can also only be declared once at the top of the function's scope. The regular expression ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* is used to parse what can be considered an identifier.

3.1.2 Keywords

The following keywords are reserved for use in the language and cannot be used as identifiers. These keywords are case sensitive.

if, else, for, while, return, null, int, bool, true, false, void, string, array, size, canvas, rows, columns

3.1.3 Comments

The characters **:(** *introduce a comment, which terminates with the character* :**)** This notation for comments can be used for both inline and block.

3.1.4 Literals

Literals are constants, either string, numeric, or boolean values. Type casting is not supported, so changing the type of a variable from its initial type will throw an error.

3.1.4.1 String Literals

String literals are enclosed in double quotes, and are sequences of zero or more non-double-quote characters. The regular expression '"' ([^ '"'] | "\\\"")* '"' defines what can be considered a string literal in our scanner.

Examples: “Crayon is cool”

3.1.4.2 Integer Literals

Integer literals are 32-bit representations of whole numbers in base-10 represented by a sequence of one or more digits Integers preceded by the dash character are negative numbers, separate from the subtraction operator mentioned below. The regular expression ['0'-'9']+ defines what can be considered an integer literal.

Examples: 42 *or* -6

3.1.4.3 Boolean Literals

A boolean literal is represented by a bit and have the values *true* or *false.*

3.1.4.4 Array Literals

An array literal is represented by a comma-separated list of integers enclosed by brackets, and are instantiated internally using pointers to a contiguous block of integers.

Example: [1, 2, 3, 4, 5]

3.1.5 Operators
Operators are characters that perform actions on different elements, such as addition, subtraction, and other mathematical processes. Boolean operators such as “&&” and “||” can also be used on boolean literals.
*Examples: + or * or - or* || *or* &&

3.1.6 Delimiters
Delimiters are characters that separate other elements and tell the compiler how to interpret associated delimiters.

3.1.6.1 Parentheses and Braces
Parentheses force the evaluation of parts of an algorithm in a specific order and also enclose arguments of a function, including for our *size*, *rows*, and *columns* in-built functions used for getting size information of array and canvas types .

3.1.6.2 Commas
Commas separate arguments of a function.

3.1.6.3 Brackets
Brackets are used for the initialization of array and canvas types and the access and assignment of array and canvas values, including individual colors in a canvas type.

3..6.4 Semicolon
A semicolon indicates the end of a sequence of code.

3.1.6.5 Curly Braces
Curly braces enclose function definitions and blocks of code. Blocks enclosed within curly braces do not need to be terminated by semicolons.

3.1.7 Whitespace
Whitespace separates different elements and can be used within string literals. Whitespace characters include spaces, tabs, newlines, and vertical tabs.

3.2 Data Types
Crayon is statically typed. The types of all variables are known at compile time and cannot be changed after being declared at the top of a function.

3.2.1 Primitive Data Types

3.2.1.1 int
These are 32-bit signed integers that can range from −2,147,483,648 to 2,147,483,647.
Example: int a = 2;

3.2.1.2 string
All text values will be of this type, and string literals can be assigned to this type.
Example: string s = “hello”;

3.2.1.3 boolean
A truth value that can be either true or false. *Example*: boolean b = true;

3.2.1.4 void
Used only as a return type in a function definition; specifying void as the return type of a function means that the function does not return anything. *Example:* void myFunc() {}

3.2.2 Complex Data Types

3.2.2.1 Arrays
Arrays are ordered, fixed-size lists that can be used to hold integers. All elements of an array must be of integer type. An array must be initialized with its size.

3.2.2.1.1 Declaring Arrays
You can declare an array by indicating the type of the elements that the array will contain (in our case, always int, followed by brackets enclosing the number of elements an array will hold, followed by an identifier for the array. An array literal value can then be assigned to an identifier to initialize this value.

Example:
array int[5] myArray; declares an array named myArray that can hold 5 integers.
myArray = [1, 2, 3, 4, 5];

3.2.2.1.2 Accessing and setting array elements
Array elements can be accessed by providing the desired integer index of the element in the array you wish to access enclosed within brackets next to the identifier of the array. This same notation can also be used on the left hand side of an assignment, if the right hand side matches the corresponding integer type.

Examples:
myArray[1]; returns the element in myArray at index 1. Array elements can be set by accessing the element via the desired index in which to place the item and then assigning the desired value to the entry. For example:

myArray[1]=4; sets the element in myArray at index 1 to 4.

3.2.2.2 Canvas
Canvases are finite-sized three-dimensional arrays that hold integer values to represent RGB values of pixels in an image. The first two dimensions, x and y, define the number of columns and rows in this matrix-like data structure, which could correspond to the dimensions of an image. The third dimension, z, always equals three, to represent the red, green, and blue color values. These canvas values can be modified to create the image. A canvas can be rendered by the corresponding Crayon graphics library to produce the raster image in PPM format.

3.2.2.2.1 Declaring Canvas
You can declare a canvas by indicating the dimension sizes in brackets, x for the number of rows and y for the number of columns.

Example:
canvas[20,20] myCanvas;

3.2.2.2.1 Accessing canvas elements and setting color
You can access the color at a certain pixel of the canvas by including its row and column coordinate location in brackets, similar to accessing an array element but in two dimensions.

Example:
myCanvas[5, 10]; returns the 3-integer color array corresponding to a pixel at row 5 and column 10 of the Canvas

myCanvas[5, 10] = [255, 255, 255]; sets the pixel at row 5 and column 10 to the full RGB value for white via this 3-integer array on the right hand side

myCanvas[5, 10, 0]; returns the R value at pixel (5,10). The third element of the access array corresponds to the Red, Green, or Blue value (0, 1, 2 respectively)

myCanvas[5, 10, 2] = 255; sets the blue value for the pixel at row 5 and column 10 to 255

3.2.2.3 Pointer
Pointers store the memory address for integers in our language. Pointers work for Canvas and Array types. *Example:*

canvas $ ptr; declares a canvas pointer
array $ ptr2; declares an array pointer

3.2.2.3.1 Setting Pointer Reference
ptr = &myCanvas; sets the pointer equal to the memory address of myCanvas. &myCanvas will return a pointer to the Red value of pixel at 0th column and 0th row (myCanvas[0,0,0])

Ptr2 = &myArray; sets the pointer equal to the memory address of myArray. &myArray will return a pointer to the first element of the array (myArray[0])

3.2.2.3.2 Dereferencing
~ptr; returns the R, G, or B value at the memory address

~ptr2; returns the element of the array at the memory address

3.2.2.3.3 Pointer Arithmetic
@24@ ptr; moves the address value of **ptr** up by 24. For a canvas, this means moving forward 8 pixels (3 elements in each pixel). For an array, this means moving forward 24 elements.

@-24@ ptr; moves the address value of **ptr** down by 24.

3.3 Expressions and Operators

3.3.1 Expressions
Expressions are made up of one or more operands and zero or more operators. Innermost expressions are evaluated first, as determined by grouping into parentheses, and operator precedence helps determine order of evaluation. Expressions are otherwise evaluated left to right.

3.3.2 Operators
The table below presents the language operators (including assignment operators, mathematical operators, logical operators, and comparison operators), descriptions, and associativity rules.

Operator	**Meaning**
+	addition
-	subtraction
*	multiplication
/	division
=	assignment
%	modulus
++	increment
--	decrement
==	equality
!=	not equal
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
&&	and
||	or

!	not

3.3.3 The if Statement
The if statement is used to execute a statement if a specified condition is met. If the specified condition is not met, the statement is skipped over. The general form of an if statement is as follows:

```
if( condition ) {
        action1;
}
else {
        Default action;
}
```

“if” must be followed with “else,” although the else may have no statement associated with it:

```
if( condition ) {
        action1;
        } else{ }
```

“else if” can be used to continue imposing multiple conditions before the final “else” statement.

3.3.4 Functions

3.4.1 Function Definitions
Function definitions consist of an initial keyword “function,” a return type, a function identifier, a set of parameters and their types, and then a block of code to execute when that function is called with the specified parameters. When using a canvas in functions outside of your scope, you would pass a canvas pointer rather than the canvas itself in order to retain access and change its values. An example of an addition function definition is as follows:

```
int sum(int a, int b) {
        return a + b;
}
```

3.3.4.2 Calling Functions
A function can be called its identifier followed by its params in parentheses.
Example: sum(1,2);

3.3.4.3 Built-in Functions

3.3.4.3.1 The print functions
The **print_string** function can be used to print out strings to the command line. The general structure for calling the print_string function is as follows:

```
        print_string("welcometothejungle");
```

For integers, use **print:**

    print(3005);

For booleans, use **printb:**

    printb(3<5);

Anything within the parentheses will be printed; it must be of type string, integer, or boolean. It will automatically print with a newline.

3.3.4.3.2 The size functions

The **size(a)** function takes in an array a and returns the integer for length of the array.
Example:
array int[5] a;
size(a); *:(returns 5 :)*

The **rows(c)** function takes in a canvas c and returns the integer for its number of rows.
Example:
canvas [10][5] a;
rows(a); *:(returns 10 :)*

The **columns(c)** function takes in a canvas c and returns the integer for its number of columns.
Example:
canvas [10][5] a;
columns(a); *:(returns 5 :)*

3.3.4.3.3 The PPM functions

The **readfile(filename)** function, show in in the tutorial, takes in a string value for the filename of the PPM to open and read, and returns a canvas pointer to the PPM values, starting at the number of columns, then rows, and then all color values for the pixels. This pointer can be used to assign corresponding values in an actual canvas type.
Example:
canvas $ ptr;
ptr = readfile("myPicture.ppm");

The **writefile(ptr, nrows, ncols, filename)** function, shown in the tutorial, takes in a canvas pointer, integer for number of rows in canvas, integer for number of columns in canvas, and string value for filename for the PPM file to be written. It is void, but saves a ppm file corresponding to the values in the canvas.

Example:
canvas[10,10] c;
canvas $ ptr;

```
ptr = &c;
writefile(ptr, 10, 10, “myPicture.ppm”);
```

3.3.4 Standard Library

Function name	*Argument*	*Returns*	*Functionality*
printArray	array a, int length	void	Prints out elements of array
canvasAvg	Canvas $ p1, canvas $ p2, int row1, int row2, int col1, int col2	void	Takes in 2 canvas pointers and their sizes and averages color values of both and sets those values in-place to p1
canvasFillRGB	Canvas $ p1, int row, int cols, int R, int G, int B	void	Takes in canvas and size, RGB color value, fills all pixels of p1 with input color in-place
canvasFillColor	Canvas $p1, int row, int cols, string colorname	void	Takes in canvas and size, color value by string (for common colors), fills pixels of p1 with it
canvasGreyscale	Canvas $p1, int row, int cols	void	Takes in canvas and converts to greyscale by averaging pixel’s RGB values
drawRectangle	Canvas $p1, int row, int cols, int x_coord, int y_coord, int length, int height	void	Takes in canvas, size, starting coordinates of shape, length and height of rectangle, and draws rectangle according to parameters in canvas
canvasInvert	Canvas $p1, int row, int cols	void	Takes in canvas, size, and inverts its color values
setPointer	Canvas $p1, int row, int cols, int i, int j	Canvas$	Takes in canvas pointer, size, and returns new pointer at index [i,j] for same canvas
drawCircle	Canvas $p1, int row, int cols, int x_coord, int y_coord, int radius	void	Takes in canvas, size, center coordinates of circle, radius, and draws approximate circle in canvas

3.3.5 Context Free Grammar

program:

```
  decls EOF

decls:
   /* nothing */ =
 | decls vdecl
 | decls fdecl

fdecl:
   typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE

formals_opt:
    /* nothing */
  | formal_list

formal_list:
    typ ID
  | formal_list COMMA typ ID

typ:
    INT
  | BOOL
  | VOID
  | STRING
  | ARRAY typ LBRACK NUM_LIT RBRACK
  | CANVAS DOLLAR
  | ARRAY DOLLAR
  | CANVAS LBRACK NUM_LIT COMMA NUM_LIT RBRACK

vdecl_list:
    /* nothing */
  | vdecl_list vdecl

vdecl:
   typ ID SEMI

stmt_list:
    /* nothing */
  | stmt_list stmt

stmt:
    expr SEMI
  | RETURN SEMI
  | RETURN expr SEMI
  | LBRACE stmt_list RBRACE
  | IF LPAREN expr RPAREN stmt %prec NOELSE
  | IF LPAREN expr RPAREN stmt ELSE stmt
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
  | WHILE LPAREN expr RPAREN stmt
```

```
expr_opt:
   /* nothing */
  | expr

expr:
   NUM_LIT
  | STRING_LIT
  | TRUE
  | FALSE
  | ID
  | NULL
  | expr PLUS   expr
  | expr MINUS  expr
  | expr TIMES  expr
  | expr DIVIDE expr
  | expr EQ     expr
  | expr NEQ    expr
  | expr LT     expr
  | expr LEQ    expr
  | expr GT     expr
  | expr GEQ    expr
  | expr AND    expr
  | expr OR     expr
  | MINUS expr %prec NEG
  | NOT expr
  | expr ASSIGN expr
  | ID LPAREN actuals_opt RPAREN
  | LPAREN expr RPAREN
  | ID LBRACK expr RBRACK
  | LBRACK arr_lit RBRACK
  | ID LBRACK expr COMMA expr RBRACK
  | ID LBRACK expr COMMA expr COMMA expr RBRACK
  | SIZE LPAREN ID RPAREN
  | ROWS LPAREN ID RPAREN
  | COLUMNS LPAREN ID RPAREN
  | AMPER ID
  | TILDE ID
  | AT expr AT ID

actuals_opt:
   /* nothing */
  | actuals_list

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

```
arr_lit:
   expr { [$1] }
  | arr_lit COMMA expr
```

4 Project Plan

4.1 Planning Process

Throughout the planning, development, and testing of our language, we used a generally Agile (iterative) approach. More specifically, we focused our efforts intensely on each portion of the process, reviewing revising the plan at each stage, and testing thoroughly before moving on. We set long term guidelines that were loose enough to allow continuous revision, yet strong enough to keep our focus on creating a flexible raster-graphics language. Some of the deadlines/ guidelines for our planning process were suggested by our TA mentor.

The stages we planned were as follows (steps 2, 3-6, 8 were iterative) :

1. Language brainstorm and design
2. Language Reference Manual draft
3. Development environment and version control setup
4. Compiler front-end development
5. Semantic and type-checking development
6. Code generation development
7. Standard Library creation
8. Complete testing and debugging
9. Final report and presentation completion

4.2 Specification Process

The main source for our specification was the Language Reference Manual (LRM). Although we frequently made revisions - the delimiters for comments and the removal of the Pixel type, for instance - due to necessity or preference, we always stuck to the core goals laid out in the beginning. For the design of our compiler, we took a lot of cues from lectures and MicroC, while incorporating our own changes to fit the specification of our language - e.g. dealing with two-dimensional array types. Most importantly, our Agile development process allowed us to change the specification repeatedly, without derailing the entire project.

4.3 Development Process

As outlined above, we developed our compiler in the order of front-end (parser, AST) -> semantics checker -> code generator. However, we did this iteratively, so that new changes could be incorporated smoothly. This means that even though "Hello World" was completed early in the process, the fully functional and specification-complete compiler wasn't finished until much later. We used a GitHub private repository to collaborate on our code. Additionally, we used a Virtual Machine running the Linux *Ubuntu* OS to run our compiler and execute code.

4.4 Testing Process

In each step of our development process, we ran numerous tests to ensure that each new "feature" properly functioned. This entailed writing unit tests for standard library functions and language features such as for loops and variable assignments. See the testing plan section for more.

4.5 Programming Style Guide

Tabs and spaces:

- 2 spaces

Commenting:

- Crayon comments to describe test cases and standard library functions
- OCaml comments to describe parts of compiler

Writing test cases:

- test_name_a: describes the first test for name()
- fail_name_b: describes the second exception test for name()

4.6 Project Timeline

Approximate Date	**Goal Met**
February 8	Language Proposal Complete
February 22	Language Reference Manual Complete
March 30	Preliminary Compiler Built (hello_world.cry runs)
April 29	Secondary Compiler Version Built (arrays, Canvas type)

May 8	Final Compiler Version Built (pointers, writefile)
May 9	Standard Library Complete
May 9	System Testing and Debugging Complete
May 10	Final Report Complete

4.7 Roles and Responsibilities

Generally, the responsibilities of the project were shared across the team. The chart below represents which team member(s) carried the most weight in ensuring the completion of each responsibility.

Name	**Role/ Responsibilities**
Naman Agrawal	Manager / compiler front end; semantics
Vaidehi Dalmia	Tester / test design; code generation
Ganesh Ravichandran	Language Guru / semantics; code generation
David Smart	System Architect / test design; compiler front end
All	Standard Library Functions

4.8 Software Development Environment

For our version control system we used a private Github private repository. This allowed us to collaborate remotely, merge changes to the codebase, and revert to previous versions if necessary.

Additionally, we decided to run our compiler on Ubuntu 16.04 in order to run LLVM 3.8. Since we all have either Macs or PCs, we set up virtual machines using VirtualBox (and ssh-ing into the machine when necessary). An added benefit to this is that the runtime environment was similar for all of us.

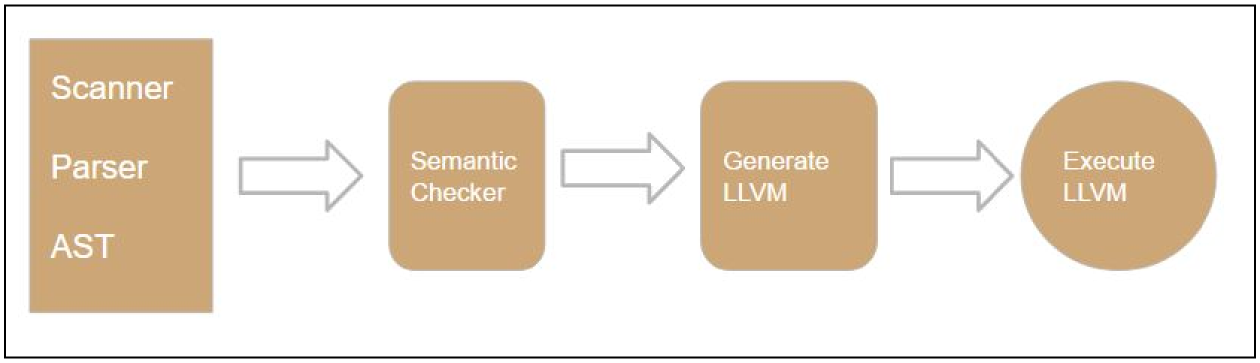
The languages used in this project were:
- C for the Makefile and writefile (for creating PPM files)
- OCaml for all compiler files (AST, parser, semant, etc)
- Crayon for standard library function

4.9 Project Log

Date	**Task Completed**
February 8	Language brainstorm and proposal
February 15	Syntax and Formal Grammar defined
February 22	Language Reference Manual
March 22	Preliminary Compiler Front End
March 30	Preliminary Semantics and Code Generation
March 30	Hello_world.cry complete
April 27	Array type implemented
April 29	Crayon type implemented
May 4	Pointers implemented
May 8	File editing implemented
May 9	Standard library development complete
May 9	System wide testing and debugging complete
May 10	Final report complete

5 Architectural Design

5.1 Block Diagram



5.2 Design Description

The description of the diagram in 5.1 is as follows:

In the compiler front-end, our scanner reads plain text characters, disregards whitespace and comments, then passes valid tokens to the parser - which constructs the Abstract Syntax Tree based on our grammar and the structure of the AST.

Once a valid AST is formed, the semantic checker ensures that types are correct and validates all aspects of it, raising errors if there are inconsistencies.

Next, the code generator constructs LLVM (Low Level Virtual Machine) code from Crayon code. Although not pictured in our diagram, but our code was also linked with compiled code from our C file, which was used for reading and writing files. The linking aspect can be viewed in our Makefile and testall.sh script (See Appendix for more information).

Finally, a file can be passed to our complete compiler, which outputs LLVM code. Running **lli** executes the LLVM code.

Naman and David focused more on fine-tuning the front-end (scanner, parser, AST) and the semantic type checker, while Vaidehi and Ganesh focused more on the code generation stage and dealing with the linking for C files.

6 Test Plan

6.1 Example Programs

Below are some representative source language programs with their corresponding target language program generated in LLVM.

Example 1: The below code tests both readfile and writefile. It takes in a ppm file, gets the canvas pointer for the ppm file, converts the pointer back to a canvas and creates a new ppm file with the new canvas.

Source Program:

```
:(tests writefile and readfile ppm production :)

int main()
{
        int row;
        int col;
        int i;
        int j;
        int k;
        int m;
        canvas $ p;
        canvas $ q;
        canvas[600, 600] c;
        p = readfile("image.ppm");
        row = ~p;
        p = @1@ p;
        col = ~p;
        print(row);
        print(col);
        for(i = 0; i < 600; i = i + 1) {
                for(j = 0; j < 600; j = j + 1) {
                        for(k = 0; k < 3; k = k + 1) {
                                if(i<row && j<col){
                                        p = @1@ p;
                                        c[i, j, k] = ~p;
                                }
                                else{
                                        c[i, j, k] = 255;
                                }
                        }
                }
        }
        q = &c;
        writefile(q, 600, 600,"newimage.ppm");
        return 0;
}
```

Target Result:

```
; ModuleID = 'Crayon'

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@tmp = private unnamed_addr constant [10 x i8] c"image.ppm\00"
@tmp.2 = private unnamed_addr constant [13 x i8] c"newimage.ppm\00"

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

declare i32 @strcompare(i8*, i8*)

declare i32 @writefile(i32*, i32, i32, i8*)

declare i32* @readfile(i8*)

define i32 @main() {
entry:
  %row = alloca i32
  %col = alloca i32
  %i = alloca i32
  %j = alloca i32
  %k = alloca i32
  %m = alloca i32
  %p = alloca i32*
  %q = alloca i32*
  %c = alloca [600 x [600 x [3 x i32]]]
  %readfile = call i32* @readfile(i8* getelementptr inbounds ([10 x i8], [10 x i8]* @tmp, i32 0, i32 0))
  store i32* %readfile, i32** %p
  %p1 = load i32*, i32** %p
  %p2 = load i32, i32* %p1
  store i32 %p2, i32* %row
  %p3 = getelementptr inbounds i32*, i32** %p, i32 0
  %p4 = load i32*, i32** %p3
  %p5 = getelementptr inbounds i32, i32* %p4, i32 1
  store i32* %p5, i32** %p
  %p6 = load i32*, i32** %p
  %p7 = load i32, i32* %p6
  store i32 %p7, i32* %col
  %row8 = load i32, i32* %row
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 %row8)
  %col9 = load i32, i32* %col
  %printf10 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 %col9)
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %merge43, %entry
  %i46 = load i32, i32* %i
  %tmp47 = icmp slt i32 %i46, 600
  br i1 %tmp47, label %while_body, label %merge48

while_body:                                       ; preds = %while
```

```
  store i32 0, i32* %j
  br label %while11

while11:                                          ; preds = %merge38, %while_body
  %j41 = load i32, i32* %j
  %tmp42 = icmp slt i32 %j41, 600
  br i1 %tmp42, label %while_body12, label %merge43

while_body12:                                     ; preds = %while11
  store i32 0, i32* %k
  br label %while13

while13:                                          ; preds = %merge, %while_body12
  %k36 = load i32, i32* %k
  %tmp37 = icmp slt i32 %k36, 3
  br i1 %tmp37, label %while_body14, label %merge38

while_body14:                                     ; preds = %while13
  %i15 = load i32, i32* %i
  %row16 = load i32, i32* %row
  %tmp = icmp slt i32 %i15, %row16
  %j17 = load i32, i32* %j
  %col18 = load i32, i32* %col
  %tmp19 = icmp slt i32 %j17, %col18
  %tmp20 = and i1 %tmp, %tmp19
  br i1 %tmp20, label %then, label %else

merge:                                            ; preds = %else, %then
  %k34 = load i32, i32* %k
  %tmp35 = add i32 %k34, 1
  store i32 %tmp35, i32* %k
  br label %while13

then:                                             ; preds = %while_body14
  %p21 = getelementptr inbounds i32*, i32** %p, i32 0
  %p22 = load i32*, i32** %p21
  %p23 = getelementptr inbounds i32, i32* %p22, i32 1
  store i32* %p23, i32** %p
  %i24 = load i32, i32* %i
  %j25 = load i32, i32* %j
  %k26 = load i32, i32* %k
  %c27 = getelementptr [600 x [600 x [3 x i32]]], [600 x [600 x [3 x i32]]]* %c, i32 0, i32 %i24, i32 %j25, i32 %k26
  %p28 = load i32*, i32** %p
  %p29 = load i32, i32* %p28
  store i32 %p29, i32* %c27
  br label %merge

else:                                             ; preds = %while_body14
  %i30 = load i32, i32* %i
  %j31 = load i32, i32* %j
  %k32 = load i32, i32* %k
  %c33 = getelementptr [600 x [600 x [3 x i32]]], [600 x [600 x [3 x i32]]]* %c, i32 0, i32 %i30, i32 %j31, i32 %k32
  store i32 255, i32* %c33
  br label %merge
```

```
merge38:                                  ; preds = %while13
  %j39 = load i32, i32* %j
  %tmp40 = add i32 %j39, 1
  store i32 %tmp40, i32* %j
  br label %while11

merge43:                                  ; preds = %while11
  %i44 = load i32, i32* %i
  %tmp45 = add i32 %i44, 1
  store i32 %tmp45, i32* %i
  br label %while

merge48:                                  ; preds = %while
  %c49 = getelementptr inbounds [600 x [600 x [3 x i32]]], [600 x [600 x [3 x i32]]]* %c, i32 0, i32 0, i32 0, i32 0
  store i32* %c49, i32** %q
  %q50 = load i32*, i32** %q
  %writefile = call i32 @writefile(i32* %q50, i32 600, i32 600, i8* getelementptr inbounds ([13 x i8], [13 x i8]* @tmp.2, i32
0, i32 0))
  ret i32 0
}
```

Example 2: The code below initialises canvases and arrays and tests whether assignments for them work. In addition it also checks whether pointers for both work.

Source Program:

```
:( Tests re-assignment of colors to canvas :)
int main() {

  canvas [5,5] g;
  array int[3] a;
  array int[3] b;
  canvas $ p;

  g[0,0] = [251, 252, 253];
  g[0,1] = [1, 2, 3];

  a = g[0,0];
  b = g[0,1];

  print(a[0]);
  print(b[0]);

  p = &g;
  ~p = 255;

  print(~p);

  return 0;
}
```

Target Result:

```
; ModuleID = 'Crayon'

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"

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

declare i32 @strcompare(i8*, i8*)

declare i32 @writefile(i32*, i32, i32, i8*)

declare i32* @readfile(i8*)

define i32 @main() {
entry:
  %g = alloca [5 x [5 x [3 x i32]]]
  %a = alloca [3 x i32]
  %b = alloca [3 x i32]
  %p = alloca i32*
  %g1 = getelementptr [5 x [5 x [3 x i32]]], [5 x [5 x [3 x i32]]]* %g, i32 0, i32 0, i32 0
  store [3 x i32] [i32 251, i32 252, i32 253], [3 x i32]* %g1
  %g2 = getelementptr [5 x [5 x [3 x i32]]], [5 x [5 x [3 x i32]]]* %g, i32 0, i32 0, i32 1
  store [3 x i32] [i32 1, i32 2, i32 3], [3 x i32]* %g2
  %g3 = getelementptr [5 x [5 x [3 x i32]]], [5 x [5 x [3 x i32]]]* %g, i32 0, i32 0, i32 0
  %g4 = load [3 x i32], [3 x i32]* %g3
  store [3 x i32] %g4, [3 x i32]* %a
  %g5 = getelementptr [5 x [5 x [3 x i32]]], [5 x [5 x [3 x i32]]]* %g, i32 0, i32 0, i32 1
  %g6 = load [3 x i32], [3 x i32]* %g5
  store [3 x i32] %g6, [3 x i32]* %b
  %a7 = getelementptr [3 x i32], [3 x i32]* %a, i32 0, i32 0
  %a8 = load i32, i32* %a7
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 %a8)
  %b9 = getelementptr [3 x i32], [3 x i32]* %b, i32 0, i32 0
  %b10 = load i32, i32* %b9
  %printf11 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 %b10)
  %g12 = getelementptr inbounds [5 x [5 x [3 x i32]]], [5 x [5 x [3 x i32]]]* %g, i32 0, i32 0, i32 0, i32 0
  store i32* %g12, i32** %p
  %p13 = load i32*, i32** %p
  store i32 255, i32* %p13
  %p14 = load i32*, i32** %p
  %p15 = load i32, i32* %p14
  %printf16 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 %p15)
  ret i32 0
}
```

6.2 Test Suites

Because we had taken a lot of the basic functionality of our program from MicroC. We wrote a lot of the small tests like variable declarations, loops, simple operators and function calls by looking at the way MicroC implemented their tests. We then wrote additional tests every time we implemented a new feature such as arrays, canvas', pointers. We added "black box" tests (tests for large programs that could fail in multiple areas) for features such as writefile and readfile.

6.3 Test Automation

The test directory contains all our tests(".cry" files) and correct output for each test script (".out" files). Our testing automation program is similar to that of Micro C and is available in our main directory. It can be invoked by calling "./testall.sh". It will run each of the test files that start with "test_" and end with ".cry" and compares its output to the file with the same name but with the extension ".out". If the test prints the expected output, "testall.sh" will print the filename + "...OK".

The testing file along with the test files are included in the Appendix.

David and Vaidehi wrote the tests similar to MicroC and the test cases for writefile and readfile. Naman and Ganesh wrote the test cases for arrays, canvas', pointers.

7 Lessons Learned

David:

The most important lesson I learned is the importance of flexibility. Being flexible in our approach to design changes was very beneficial when things got rough. For instance when we couldn't figure out how to manipulate our canvases with our standard library functions - because size must be known beforehand - we had to pivot our design to incorporate pointers. However, since we set up our canvas type to be very flexible, pointers were easily implemented for arrays and canvases in one go. Additionally, I would say that harder deadlines are necessary. I would advise future teams to set hard deadlines and focus on them, because working with friends can be almost too much fun.

Naman:

I think the most important lesson I learned was that as a manager, I need to be on top of scheduling and planning out work for everyone in the group. I considered myself and my colleagues to all be responsible for the whole project, but I think that as a manager I should have taken more ownership of the project at the start. Then it would be my job to ensure that we all fairly split the work. Nevertheless, I am very happy with the way my team split up the work overall.

Vaidehi:

The most important lessons I learned involved the difficulties in building a multi-level application like a compiler. This is the first time that I have had to deal with so many different files all operating in sync with each other, and oftentimes it was very frustrating to follow an error through semant.ml and ast.ml and codegen.ml. I learned a lot about how to organize my workspace on the computer to efficiently debug errors and write tests for different parts of the compiler.

Ganesh:

By far the most important lesson I learned in this project the magic of how a compiler works. I have been coding almost my entire life, and it has always seemed like magic to me that I can type Python into my computer and it does amazing things. But to finally understand how code is scanned, parsed, semantically checked, and turned into assembly code was really fantastic. This has been the most challenging and rewarding engineering project I have ever worked on, and I learned a lot about the challenges of writing a language.

8 Appendix

8.1 Scanner.mll

```
{ open Parser }

rule token = parse

[' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| ":(" { comment lexbuf } (* Comments *)

| '(' { LPAREN }
| ')' { RPAREN }
| '[' { LBRACK }
| ']' { RBRACK }
| '{' { LBRACE }
| '}' { RBRACE }
| '=' { ASSIGN }

| '+' { PLUS }
| '-' { MINUS }
| '/' { DIVIDE }
| '*' { TIMES }

| ';' { SEMI }
| ',' { COMMA }
| '&' { AMPER }
| '~' { TILDE }
| '$' { DOLLAR }
| '@' { AT }

| '!' { NOT }
| '>' { GT }
| '<' { LT }
| ">=" { GEQ }
| "<=" { LEQ }
| "==" { EQ }
| "!=" { NEQ }
| "||" { OR }
| "&&" { AND }

| "if" { IF }
| "else" { ELSE }
| "for" { FOR }
| "while" { WHILE }
| "return" { RETURN }
| "null" { NULL }
| "int" { INT }
| "boolean" { BOOL }
| "void" { VOID }
| "true" { TRUE }
| "false" { FALSE }
| "string" { STRING }
| "array" { ARRAY }
| "canvas" { CANVAS }
| "size" { SIZE }
| "rows" { ROWS }
| "columns" { COLUMNS }

| ['0'-'9']+ as lxm { NUM_LIT(int_of_string lxm) }
| '"' (([^ '"'] | "\\\"")* as strlit) '"' { STRING_LIT(strlit) }

| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

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

8.2 Ast.ml

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

type uop = Neg | Not

type expr = NumLit of int
        | BoolLit of bool
        | StringLit of string
        | ArrayLit of expr list
        | Id of string
        | Noexpr
        | Null
        | Binop of expr * op * expr
        | Unop of uop * expr
        | Assign of expr * expr
        | Call of string * expr list
        | ArrayAccess of string * expr
        | CanvasAccess of string * expr * expr
        | ColorAccess of string * expr * expr * expr
        | Size of string
        | Rows of string
        | Columns of string
```

```
          | Reference of string
          | Dereference of string
          | MovePointer of expr * string

type typ = Int
         | Bool
         | Void
         | String
         | Array of typ * int
         | Canvas of int * int
         | Pointer

type bind = typ * string

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

type func_decl = {
  typ : typ;
  fname : string;
  formals : bind list;
  locals : bind list;
  body : stmt list;
}

type program = bind list * func_decl list

(* Pretty-printing functions *)

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

let string_of_uop = function
```

```
    Neg -> "-"
  | Not -> "!"

let rec string_of_expr = function
    NumLit(l) -> string_of_int l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | StringLit(s) -> s
  | ArrayLit(l) -> "array lit"
  | Id(s) -> s
  | Null -> "null"
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> string_of_expr v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""
  | ArrayAccess(s, e2) -> (s) ^ "[" ^ (string_of_expr e2) ^ "]"
  | CanvasAccess(s, e1, e2) -> (s) ^ "[" ^ (string_of_expr e1) ^ "," ^ (string_of_expr e2) ^ "]"
  | ColorAccess(s, e1, e2, e3) -> (s) ^ "[" ^ (string_of_expr e1) ^ "," ^ (string_of_expr e2) ^ "," ^
(string_of_expr e3) ^ "]"
  | Size(s) -> "size(" ^ s ^ ")"
  | Rows(s) -> "rows(" ^ s ^ ")"
  | Columns(s) -> "columns(" ^ s ^ ")"
  | Reference(s) -> "&" ^ (s)
  | Dereference(s) -> "~" ^ (s)
  | MovePointer(e, s) -> "@" ^ "@" ^ (s)

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

let string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Void -> "void"
  | String -> "string"
  | Array(t,i) -> "array(" ^ (string_of_int i) ^ ")"
  | Canvas(i1, i2) -> "canvas" ^ "[" ^ (string_of_int i1) ^ "]" ^ "[" ^ (string_of_int i2) ^ "]"
  | Pointer -> "pointer"

let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

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

8.4 Parser.mly

```
/* Ocamlyacc parser for Crayon */

%{ open Ast %}

%token SEMI COMMA LPAREN RPAREN RBRACK LBRACK LBRACE RBRACE
%token PLUS MINUS TIMES DIVIDE ASSIGN
%token EQ NEQ LT LEQ GT GEQ AND OR NOT
%token RETURN IF ELSE FOR WHILE FUNC NEG
%token INT BOOL VOID STRING ARRAY CANVAS TRUE FALSE
%token TILDE AMPER DOLLAR AT

%token <int> NUM_LIT
%token <string> STRING_LIT
%token <string> ID
%token EOF
%token NULL

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

```
%token SIZE
%token ROWS
%token COLUMNS

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

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

fdecl:
   typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
     { { typ = $1;
           fname = $2;
           formals = $4;
           locals = List.rev $7;
           body = List.rev $8 } }

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

formal_list:
    typ ID                 { [($1,$2)] }
  | formal_list COMMA typ ID { ($3,$4) :: $1 }

typ:
    INT { Int }
  | BOOL { Bool }
  | VOID { Void }
  | STRING { String }
  | ARRAY typ LBRACK NUM_LIT RBRACK { Array($2, $4) }
  | CANVAS DOLLAR { Pointer }
  | ARRAY DOLLAR { Pointer }
  | CANVAS LBRACK NUM_LIT COMMA NUM_LIT RBRACK { Canvas($3, $5) }

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

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

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

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

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

expr:
    NUM_LIT         { NumLit($1) }
  | STRING_LIT      { StringLit($1) }
  | TRUE            { BoolLit(true) }
  | FALSE           { BoolLit(false) }
  | ID              { Id($1) }
  | NULL            { Null }
  | expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | expr AND    expr { Binop($1, And,   $3) }
  | expr OR     expr { Binop($1, Or,    $3) }
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr        { Unop(Not, $2) }
  | expr ASSIGN expr   { Assign($1, $3) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN { $2 }
  | ID LBRACK expr RBRACK { ArrayAccess($1, $3) }
  | LBRACK arr_lit RBRACK { ArrayLit(List.rev $2) }
  | ID LBRACK expr COMMA expr RBRACK { CanvasAccess($1, $3, $5) }
  | ID LBRACK expr COMMA expr COMMA expr RBRACK { ColorAccess($1, $3, $5, $7) }
  | SIZE LPAREN ID RPAREN { Size($3) }
  | ROWS LPAREN ID RPAREN { Rows($3) }
  | COLUMNS LPAREN ID RPAREN { Columns($3) }
  | AMPER ID { Reference($2) }
  | TILDE ID { Dereference($2) }
  | AT expr AT ID { MovePointer($2, $4) }

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

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

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

8.5 Semant.ml

```
(* Semantic checking for the Crayon compiler *)

open Ast
module StringMap = Map.Make(String)

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

   Check each global variable, then check each function *)

let check (globals, functions) =

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

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

```
in

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

(**** Checking Global Variables ****)

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

report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd globals);

(**** Checking Functions ****)

if List.mem "print" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function print may not be defined")) else ();

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

(* Function declaration for a named function *)
let built_in_decls =
     StringMap.add "print"
   { typ = Void; fname = "print"; formals = [(Int, "x")];
     locals = []; body = [] }

     (StringMap.add "printb"
   { typ = Void; fname = "printb"; formals = [(Bool, "x")];
     locals = []; body = [] }

     (StringMap.add "print_string"
   { typ = Void; fname = "print_string"; formals = [(String, "x")];
     locals = []; body = [] }

     (StringMap.add "strcompare"
   { typ = Int; fname = "strcompare"; formals = [(String, "s");(String, "s")];
     locals = []; body = [] }



     (StringMap.add "readfile"
   { typ = Pointer; fname = "readfile"; formals = [(String, "s")];
     locals = []; body = [] }

   (StringMap.singleton "writefile"
   { typ = Void; fname = "writefile"; formals = [(Pointer, "c"); (Int, "nrows"); (Int, "ncols"); (String,
"filename")];
```

```
    locals = []; body = [] }))))
  in

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

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

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

let check_function func =

  List.iter (check_not_void (fun n -> "illegal void formal " ^ n ^
    " in " ^ func.fname)) func.formals;

  report_duplicate (fun n -> "duplicate formal " ^ n ^ " in " ^ func.fname)
    (List.map snd func.formals);

  List.iter (check_not_void (fun n -> "illegal void local " ^ n ^
    " in " ^ func.fname)) func.locals;

  report_duplicate (fun n -> "duplicate local " ^ n ^ " in " ^ func.fname)
    (List.map snd func.locals);

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

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

  let array_access_type = function
    Array(t, _) -> t
    | _ -> raise (Failure ("Can only access a[x] from an array a"))
  in

  let canvas_access_type = function
    Canvas(_, _) -> Array(Int, 3)
    | _ -> raise (Failure ("illegal canvas access") )
  in

  let color_access_type = function
```

```
    Canvas(_, _) -> Int
    | _ -> raise (Failure ("Can only access c[x,y] from a canvas c"))
  in


  let rec check_array_literal m ty idx =
    let l = List.length m in
      match (ty, List.nth m idx) with
    (Array(Int, _), NumLit _) -> if idx == l - 1 then Array(Int, l) else check_array_literal m (Array(Int, l))
(succ idx)
    | _ -> raise (Failure ("Illegal array literal"))
  in

  let array_type s = match (List.hd s) with
    | NumLit _ -> Array(Int, List.length s)
    | _ -> raise ( Failure ("Array type must be integer"))
  in

  (* Return the type of an expression or throw an exception *)
  let rec expr = function
            NumLit _ -> Int
    | BoolLit _ -> Bool
    | StringLit _ -> String
    | ArrayLit s -> check_array_literal s (array_type s) 0
    | Size(s) -> (match (type_of_identifier s) with
             Array(_, _) -> Int)
    | Rows(s) -> (match (type_of_identifier s) with
             Canvas(_, _) -> Int)
    | Columns(s) -> (match (type_of_identifier s) with
             Canvas(_, _) -> Int)
    | ArrayAccess(s, e1) -> let _ = (match (expr e1) with
                                Int -> Int
                              | _ -> raise (Failure ("attempting to access with a non-integer type"))) in
                        array_access_type (type_of_identifier s)
    | CanvasAccess(s, e1, e2) -> let _ = (match (expr e1) with
                                Int -> Int
                              | _ -> raise (Failure ("attempting to access with a non-integer type")))
                            and _ = (match (expr e2) with
                                Int -> Int
                              | _ -> raise (Failure ("attempting to access with a non-integer type"))) in
                            canvas_access_type (type_of_identifier s)
    | ColorAccess(s, e1, e2, e3) -> let _ = (match (expr e1) with
                                Int -> Int
                              | _ -> raise (Failure ("attempting to access with a non-integer type")))
                            and _ = (match (expr e2) with
                                Int -> Int
                              | _ -> raise (Failure ("attempting to access with a non-integer type")))
                            and _ = (match (expr e3) with
                    Int -> Int
                   | _ -> raise (Failure ("attempting to access with a non-integer type"))) in
                  color_access_type (type_of_identifier s)
| Id s -> type_of_identifier s
| Reference(s) -> (match (type_of_identifier s) with
        Canvas(_, _) -> Pointer
       | Array(_, _) -> Pointer
       | _ -> raise ( Failure ("Cannot reference a type that is not a canvas or array"))
       )
| Dereference(s) -> (match (type_of_identifier s) with
        Pointer(_) -> Int
       | _ -> raise ( Failure ("Cannot dereference a type that is not a canvas or array pointer"))
       )
| MovePointer(e, s) -> (match (type_of_identifier s) with
         Pointer -> Pointer
        | _ -> raise ( Failure ("Cannot move a type that is not a array or canvas pointer"))
       )
| Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2 in
     (match op with
     Add | Sub | Mult | Div when t1 = Int && t2 = Int -> Int
      | Equal | Neq when t1 = t2 -> Bool
      | Less | Leq | Greater | Geq when t1 = Int && t2 = Int -> Bool
      | And | Or when t1 = Bool && t2 = Bool -> Bool
    | _ -> raise (Failure ("illegal binary operator " ^
        string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
        string_of_typ t2 ^ " in " ^ string_of_expr e))
    )
  | Unop(op, e) as ex -> let t = expr e in
      (match op with
        Neg when t = Int -> Int
       | Not when t = Bool -> Bool
     | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
                        string_of_typ t ^ " in " ^ string_of_expr ex)))
| Noexpr -> Void
| Assign(e1, e2) as ex -> let lt = ( match e1 with
  | ArrayAccess(s, _) -> (match (type_of_identifier s) with
      Array(_, _) -> Int
      | _ -> raise (Failure ("Can only access integer value via a[i] from a canvas a"))
     )
  | CanvasAccess(s, _, _) -> (match (type_of_identifier s) with
       Canvas(_, _) -> Array(Int, 3)
       | _ -> raise (Failure ("Can only access array value via c[i,j] from a canvas c"))
      )
  | ColorAccess(s, _, _, _) -> (match (type_of_identifier s) with
        Canvas(_, _) -> Int
       | _ -> raise (Failure ("Can only access color value via c[i,j,k] from a canvas c"))
      )
  | _ -> expr e1)
```

```
    and rt = expr e2 in
    check_assign lt rt (Failure ("Illegal assignment " ^ string_of_typ lt ^
        " = " ^ string_of_typ rt ^ " in " ^
        string_of_expr ex))
  | Call(fname, actuals) as call -> let fd = function_decl fname in
     if List.length actuals != List.length fd.formals then
       raise (Failure ("expecting " ^ string_of_int
         (List.length fd.formals) ^ " arguments in " ^ string_of_expr call))
     else
       List.iter2 (fun (ft, _) e -> let et = expr e in
          ignore (check_assign ft et
            (Failure ("illegal actual argument found " ^ string_of_typ et ^
            " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e))))
         fd.formals actuals;
       fd.typ
 in

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

 (* Verify a statement or throw an exception *)
 let rec stmt = function
               Block sl -> let rec check_block = function
       [Return _ as s] -> stmt s
     | Return _ :: _ -> raise (Failure "nothing may follow a return")
     | Block sl :: ss -> check_block (sl @ ss)
     | s :: ss -> stmt s ; check_block ss
     | [] -> ()
    in check_block sl
  | Expr e -> ignore (expr e)
  | Return e -> let t = expr e in if t = func.typ then () else
     raise (Failure ("return gives " ^ string_of_typ t ^ " expected " ^
                string_of_typ func.typ ^ " in " ^ string_of_expr e))

  | If(p, b1, b2) -> check_bool_expr p; stmt b1; stmt b2
  | For(e1, e2, e3, st) -> ignore (expr e1); check_bool_expr e2;
                      ignore (expr e3); stmt st
  | While(p, s) -> check_bool_expr p; stmt s
 in

 stmt (Block func.body)

in
List.iter check_function functions
```

8.6 Codegen.ml

```
(* Code generation: translate takes a semantically checked AST and
produces LLVM IR

LLVM tutorial: Make sure to read the OCaml version of the tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)

open Exceptions

module L = Llvm
module A = Ast

module StringMap = Map.Make(String)
module String = String

let translate (globals, functions) =
  let context = L.global_context () in
  let the_module = L.create_module context "Crayon"
  and i32_t  = L.i32_type  context
  and i8_t   = L.i8_type   context
  and i1_t   = L.i1_type   context
  and void_t = L.void_type context
  and array_t = L.array_type
  and pointer_t = L.pointer_type in

  let ltype_of_typ = function
      A.Int -> i32_t
    | A.Bool -> i1_t
    | A.Void -> void_t
    | A.String -> pointer_t i8_t
    | A.Array(typ, size) -> (match typ with
        A.Int -> array_t i32_t size
      | _ -> raise ( UnsupportedArrayType ))
    | A.Canvas(size1, size2) -> array_t (array_t (array_t i32_t 3) size2) size1
    | A.Pointer -> pointer_t i32_t

  in
```

```
(* Declare each global variable; remember its value in a map *)
let global_vars =
  let global_var m (t, n) =
    let init = L.const_int (ltype_of_typ t) 0
    in StringMap.add n (L.define_global n init the_module) m in
  List.fold_left global_var StringMap.empty globals in

(* Declare printf(), which the print built-in function will call *)
let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
let printf_func = L.declare_function "printf" printf_t the_module in

(* Declare the built-in strcompare() function *)
let strcompare_t = L.function_type i32_t [| pointer_t i8_t ; pointer_t i8_t |] in
let strcompare_func = L.declare_function "strcompare" strcompare_t the_module in

(* Declare the built-in writeFile() function *)
let writefile_t = L.function_type i32_t [| pointer_t i32_t; i32_t; i32_t; pointer_t i8_t |] in
let writefile_func = L.declare_function "writefile" writefile_t the_module in

  (* Declare the built-in readFile() function *)
let readfile_t = L.function_type (pointer_t i32_t) [| pointer_t i8_t |] in
let readfile_func = L.declare_function "readfile" readfile_t the_module in

(* Define each function (arguments and return type) so we can call it *)
let function_decls =
  let function_decl m fdecl =
    let name = fdecl.A.fname
    and formal_types =
        Array.of_list (List.map (function (t,s) -> ltype_of_typ t) fdecl.A.formals)
    in let ftype = L.function_type (ltype_of_typ fdecl.A.typ) formal_types in
    StringMap.add name (L.define_function name ftype the_module, fdecl) m in
  List.fold_left function_decl StringMap.empty functions in

(* Fill in the body of the given function *)
let build_function_body fdecl =
  let (the_function, _) = StringMap.find fdecl.A.fname function_decls in
  let builder = L.builder_at_end context (L.entry_block the_function) in

  let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder
  and string_format_str = L.build_global_stringptr "%s\n" "fmt" builder
  in

  (* Construct the function's "locals": formal arguments and locally
     declared variables.  Allocate each on the stack, initialize their
     value, if appropriate, and remember their values in the "locals" map *)
  let local_vars =
    let add_formal m (t, n) p = L.set_value_name n p;
        let local = L.build_alloca (ltype_of_typ t) n builder in
```

```
            ignore (L.build_store p local builder);
            StringMap.add n local m in

      let add_local m (t, n) =
            let local_var = L.build_alloca (ltype_of_typ t) n builder
            in StringMap.add n local_var m in

      let formals = List.fold_left2 add_formal StringMap.empty fdecl.A.formals
          (Array.to_list (L.params the_function)) in
      List.fold_left add_local formals fdecl.A.locals in

    (* Return the value for a variable or formal argument *)
    let lookup n = try StringMap.find n local_vars
                with Not_found -> StringMap.find n global_vars
    in

    let build_array_access s i1 i2 builder isAssign =
        if isAssign
            then L.build_gep (lookup s) [| i1; i2 |] s builder
            else L.build_load (L.build_gep (lookup s) [| i1; i2 |] s builder) s builder

    in

    let build_canvas_access s i1 i2 i3 builder isAssign =
        if isAssign
            then L.build_gep (lookup s) [| i1; i2; i3 |] s builder
        else L.build_load (L.build_gep (lookup s) [| i1; i2; i3 |] s builder) s builder
    in

    let build_color_access s i1 i2 i3 i4 builder isAssign =
        if isAssign
            then L.build_gep (lookup s) [| i1; i2; i3; i4 |] s builder
        else L.build_load (L.build_gep (lookup s) [| i1; i2; i3; i4 |] s builder) s builder
    in

    let build_reference s builder isCanvas =
        if isCanvas
          then L.build_in_bounds_gep (lookup s) [| L.const_int i32_t 0; L.const_int i32_t 0; L.const_int
i32_t 0; L.const_int i32_t 0 |] s builder
        else L.build_in_bounds_gep (lookup s) [| L.const_int i32_t 0; L.const_int i32_t 0 |] s builder
    in

    let build_dereference s builder isAssign =
      if isAssign
        then L.build_load (lookup s) s builder
      else
        L.build_load (L.build_load (lookup s) s builder) s builder
    in
```

```
  let build_move_pointer s n builder =
    L.build_in_bounds_gep (L.build_load (L.build_in_bounds_gep (lookup s) [| L.const_int i32_t 0 |] s
builder) s builder) [| n |] s builder
  in

  let check_if_func =
    List.fold_left (fun m (t, n) -> StringMap.add n t m)
    StringMap.empty (globals @ fdecl.A.formals @ fdecl.A.locals)
  in

  let type_of_identifier s =
   let symbols = check_if_func in
   StringMap.find s symbols
  in

  (* Construct code for an expression; return its value *)
  let rec expr builder = function
          A.NumLit i -> L.const_int i32_t i
   | A.StringLit s -> L.build_global_stringptr s "tmp" builder
   | A.BoolLit b -> L.const_int i1_t (if b then 1 else 0)
   | A.Noexpr -> L.const_int i32_t 0
   | A.Id s -> L.build_load (lookup s) s builder
   | A.ArrayLit s -> L.const_array i32_t (Array.of_list (List.map (expr builder) s))
   | A.Size s -> (match (type_of_identifier s) with
             A.Array(_, l) -> (L.const_int i32_t l)
            | _ -> raise (IllegalSizeArgument) )
   | A.Rows s -> (match (type_of_identifier s) with
             A.Canvas(r, c) -> (L.const_int i32_t r)
            | _ -> raise (IllegalRowArgument) )
   | A.Columns s -> (match (type_of_identifier s) with
             A.Canvas(r, c) -> (L.const_int i32_t c)
            | _ -> raise (IllegalColumnArgument) )
   | A.Assign (e1, e2) ->
      let e1' =
        (match e1 with
           A.Id(s) -> (lookup s)
          | A.Dereference(s) -> build_dereference s builder true
          | A.ArrayAccess(s, ind1) ->
            let i = expr builder ind1 in
              build_array_access s (L.const_int i32_t 0) i builder true
          | A.CanvasAccess(s, ind1, ind2) ->
            let i = expr builder ind1 and j = expr builder ind2 in
              build_canvas_access s (L.const_int i32_t 0) i j builder true
          | A.ColorAccess(s, ind1, ind2, ind3) ->
           let i = expr builder ind1 and j = expr builder ind2 and k = expr builder ind3 in
            build_color_access s (L.const_int i32_t 0) i j k builder true
          | _ -> raise (IllegalAssignment))
    and e2' = expr builder e2 in
    ignore (L.build_store e2' e1' builder); e2'
  | A.Binop (e1, op, e2) ->
        let e1' = expr builder e1
        and e2' = expr builder e2 in
        (match op with
          A.Add     -> L.build_add
        | A.Sub     -> L.build_sub
        | A.Mult    -> L.build_mul
    | A.Div     -> L.build_sdiv
        | A.And     -> L.build_and
        | A.Or      -> L.build_or
        | A.Equal   -> L.build_icmp L.Icmp.Eq
        | A.Neq     -> L.build_icmp L.Icmp.Ne
        | A.Less    -> L.build_icmp L.Icmp.Slt
        | A.Leq     -> L.build_icmp L.Icmp.Sle
        | A.Greater -> L.build_icmp L.Icmp.Sgt
        | A.Geq     -> L.build_icmp L.Icmp.Sge
        ) e1' e2' "tmp" builder
  | A.Unop(op, e) ->
            let e' = expr builder e in
        (match op with
          A.Neg     -> L.build_neg
    | A.Not     -> L.build_not) e' "tmp" builder
  | A.ArrayAccess (s, ind1) ->
    let i = expr builder ind1 in
      build_array_access s (L.const_int i32_t 0) i builder false
  | A.CanvasAccess(s, ind1, ind2) ->
    let i = expr builder ind1 and j = expr builder ind2 in
      build_canvas_access s (L.const_int i32_t 0) i j builder false
  | A.ColorAccess(s, ind1, ind2, ind3) ->
    let i = expr builder ind1 and j = expr builder ind2 and k = expr builder ind3 in
    build_color_access s (L.const_int i32_t 0) i j k builder false
  | A.Reference(s) -> (match (type_of_identifier s) with
                A.Array(_, l) -> build_reference s builder false
              | A.Canvas(_, _) -> build_reference s builder true
        )
  | A.Dereference(s) -> build_dereference s builder false
  | A.MovePointer(e, s) -> let e' = expr builder e in
      build_move_pointer s e' builder
  | A.Call ("print", [e]) | A.Call ("printb", [e]) ->
            L.build_call printf_func [| int_format_str ; (expr builder e) |]
            "printf" builder
  | A.Call ("writefile", e) -> let actuals= List.rev (List.map (expr builder) (List.rev e)) in
    L.build_call writefile_func (Array.of_list actuals) "writefile" builder
  | A.Call ("readfile", [e]) ->
    L.build_call readfile_func [| (expr builder e) |] "readfile" builder
   | A.Call ("strcompare", e) -> let actuals= List.rev (List.map (expr builder) (List.rev e)) in
```

```
      L.build_call strcompare_func (Array.of_list actuals) "strcompare" builder
  | A.Call ("print_string", [e]) ->
      L.build_call printf_func [| string_format_str ; (expr builder e) |] "printf" builder
  | A.Call (f, act) ->
     let (fdef, fdecl) = StringMap.find f function_decls in
         let actuals = List.rev (List.map (expr builder) (List.rev act)) in
         let result = (match fdecl.A.typ with A.Void -> ""
                                  | _ -> f ^ "_result") in
     L.build_call fdef (Array.of_list actuals) result builder
in

(* Invoke "f builder" if the current block doesn't already
   have a terminal (e.g., a branch). *)
let add_terminal builder f =
  match L.block_terminator (L.insertion_block builder) with
        Some _ -> ()
  | None -> ignore (f builder) in

(* Build the code for the given statement; return the builder for
   the statement's successor *)
let rec stmt builder = function
        A.Block sl -> List.fold_left stmt builder sl
  | A.Expr e -> ignore (expr builder e); builder
  | A.Return e -> ignore (match fdecl.A.typ with
          A.Void -> L.build_ret_void builder
        | _ -> L.build_ret (expr builder e) builder); builder
  | A.If (predicate, then_stmt, else_stmt) ->
     let bool_val = expr builder predicate in
         let merge_bb = L.append_block context "merge" the_function in

         let then_bb = L.append_block context "then" the_function in
         add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
           (L.build_br merge_bb);

         let else_bb = L.append_block context "else" the_function in
         add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
           (L.build_br merge_bb);

         ignore (L.build_cond_br bool_val then_bb else_bb builder);
         L.builder_at_end context merge_bb

  | A.While (predicate, body) ->
          let pred_bb = L.append_block context "while" the_function in
          ignore (L.build_br pred_bb builder);

          let body_bb = L.append_block context "while_body" the_function in
          add_terminal (stmt (L.builder_at_end context body_bb) body)
            (L.build_br pred_bb);
            let pred_builder = L.builder_at_end context pred_bb in
            let bool_val = expr pred_builder predicate in

            let merge_bb = L.append_block context "merge" the_function in
            ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
            L.builder_at_end context merge_bb

      | A.For (e1, e2, e3, body) -> stmt builder
              ( A.Block [A.Expr e1 ; A.While (e2, A.Block [body ; A.Expr e3]) ] )
    in

    (* Build the code for each statement in the function *)
    let builder = stmt builder (A.Block fdecl.A.body) in

    (* Add a return if the last block falls off the end *)
    add_terminal builder (match fdecl.A.typ with
        A.Void -> L.build_ret_void
      | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in

  List.iter build_function_body functions;
  the_module
```

8.7 Crayon.ml

```
(* Top-level of the Crayon compiler: scan & parse the input,
   check the resulting AST, generate LLVM IR, and dump the module *)

type action = Ast | LLVM_IR | Compile

let _ =
  let action = if Array.length Sys.argv > 1 then
    List.assoc Sys.argv.(1) [ ("-a", Ast);       (* Print the AST only *)
                              ("-l", LLVM_IR);  (* Generate LLVM, don't check *)
                              ("-c", Compile) ] (* Generate, check LLVM IR *)
  else Compile in
  let lexbuf = Lexing.from_channel stdin in
  let ast = Parser.program Scanner.token lexbuf in
  Semant.check ast;
  match action with
    Ast -> print_string (Ast.string_of_program ast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate ast))
  | Compile -> let m = Codegen.translate ast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)
```

8.8 Exceptions.ml

```
(* Codegen Exceptions *)
exception UnsupportedArrayType
exception IllegalAssignment
exception IllegalColumnArgument
exception IllegalRowArgument
exception IllegalSizeArgument
```

8.9 Loadfile.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int strcompare(char *name1,char *name2){
	return(strcmp(name1, name2));
}

void writefile(int *c,int nrows,int ncols, char* filename){
	printf("%d",nrows);
	printf("%d",ncols);

	FILE *f = fopen(filename,"wb");
	if (f==NULL){
		printf("Error opening file!\n");
		exit(1);
	}
	fprintf(f,"P3\n");
	fprintf(f,"%d %d\n",ncols,nrows);
	fprintf(f,"255\n");
	int i,j,k;
	for(i=0;i<nrows;i++){
		for(j=0;j<ncols;j++){
			for(k=0;k<3;k++){
				fprintf(f,"%d",*(c+((nrows*i+j)*3+k)));
				fprintf(f," ");
			}
		}
		fprintf(f,"\n");
	}
	fclose(f);
}

int* readfile(char *filename){
	int ncols;
	int nrows;
	int max_colour;
```

```
        int x;
        int i = 0;

        FILE *f = fopen(filename,"rb");
        if (f==NULL){
                printf("Error opening file!\n");
                exit(1);
        }

        fscanf (f, "P3 %d %d %d", &ncols, &nrows, &max_colour);
        printf("%d",ncols);

        int *temp= malloc((ncols*nrows*3+2) * sizeof(int));
        if(temp == NULL)
  {
     printf("malloc returned null");
     exit(1);
  }

        temp[0] = ncols;
        temp[1] = nrows;
        for(i=2;i<ncols*nrows*3+2;i=i+1){
                fscanf(f,"%d",&x);
                temp[i] = x;
        }

        fclose(f);
        return (temp);
}
```

Makefile

```
# Make sure ocamlbuild can find opam-managed packages: first run
#
# eval `opam config env`

# Easiest way to build: using ocamlbuild, which in turn uses ocamlfind

all : crayon.native loadfile.o

crayon.native :
	ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis -cflags -w,+a-4 \
		crayon.native

# "make clean" removes all generated files

.PHONY : clean
```

```
clean :
	ocamlbuild -clean
	rm -rf testall.log *.diff crayon scanner.ml parser.ml parser.mli
	rm -rf loadfile
	rm -rf *.cmx *.cmi *.cmo *.cmx *.o *.s *.ll *.out *.exe *.ppm

# More detailed: build using ocamlc/ocamlopt + ocamlfind to locate LLVM

OBJS = ast.cmx codegen.cmx parser.cmx scanner.cmx semant.cmx crayon.cmx

crayon : $(OBJS)
	ocamlfind ocamlopt -linkpkg -package llvm -package llvm.analysis $(OBJS) -o crayon

scanner.ml : scanner.mll
	ocamllex scanner.mll

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

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

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

%.cmx : %.ml
	ocamlfind ocamlopt -c -package llvm $<

# Testing the "loadfile" example

loadfile : loadfile.c
	cc -o -rdynamic loadfile -DBUILD_TEST loadfile.c

### Generated by "ocamldep *.ml *.mli" after building scanner.ml and parser.ml
ast.cmo :
ast.cmx :
codegen.cmo : ast.cmo
codegen.cmx : ast.cmx
crayon.cmo : semant.cmo scanner.cmo parser.cmi codegen.cmo ast.cmo
crayon.cmx : semant.cmx scanner.cmx parser.cmx codegen.cmx ast.cmx
parser.cmo : ast.cmo parser.cmi
parser.cmx : ast.cmx parser.cmi
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx
semant.cmo : ast.cmo
semant.cmx : ast.cmx
parser.cmi : ast.cmo
```

```
# Building the tarball

TESTS = add1 arith1 arith2 arith3 fib for1 for2 func1 func2 func3  \
   func4 func5 func6 func7 func8 gcd2 gcd global1 global2 global3              \
   hello if1 if2 if3 if4 if5 local1 local2 ops1 ops2 var1 var2                 \
   while1 while2 loadfile

FAILS = assign1 assign2 assign3 dead1 dead2 expr1 expr2 for1 for2              \
   for3 for4 for5 func1 func2 func3 func4 func5 func6 func7 func8 \
   func9 global1 global2 if1 if2 if3 nomain return1 return2 while1  \
   while2

TESTFILES = $(TESTS:%=test-%.mc) $(TESTS:%=test-%.out) \
          $(FAILS:%=fail-%.mc) $(FAILS:%=fail-%.err)

TARFILES = ast.ml codegen.ml Makefile crayon.ml parser.mly README scanner.mll \
        semant.ml testall.sh $(TESTFILES:%=tests/%) loadfile.c arcade-font.pbm \
        font2c

crayon-llvm.tar.gz : $(TARFILES)
        cd .. && tar czf crayon-llvm/crayon-llvm.tar.gz \
                $(TARFILES:%=crayon-llvm/%)
```

Testall.sh

```
#!/bin/sh

# Regression testing script for Crayo
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# Path to the Crayon compiler. "./crayon.native"
# Try "_build/crayon.native" if ocamlbuild was unable to create a symbolic link.
CRAYON="./crayon.native"
#CRAYON="_build/crayon.native"
```

```
# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
error=0
globalerror=0

keep=0

Usage() {
   echo "Usage: testall.sh [options] [.cry files]"
   echo "-k    Keep intermediate files"
   echo "-h    Print this help"
   exit 1
}

SignalError() {
   if [ $error -eq 0 ] ; then
            echo "FAILED"
            error=1
   fi
   echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
   generatedfiles="$generatedfiles $3"
   echo diff -b $1 $2 ">" $3 1>&2
   diff -b "$1" "$2" > "$3" 2>&1 || {
            SignalError "$1 differs"
            echo "FAILED $1 differs from $2" 1>&2
   }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
   echo $* 1>&2
   eval $* || {
            SignalError "$1 failed on $*"
            return 1
   }
}

# RunFail <args>
# Report the command, run it, and expect an error
```

```
RunFail() {
   echo $* 1>&2
   eval $* && {
           SignalError "failed: $* did not report an error"
           return 1
   }
   return 0
}

Check() {
   error=0
   basename=`echo $1 | sed 's/.*\\///
                      s/.cry//'`
   reffile=`echo $1 | sed 's/.cry$//'`
   basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

   echo -n "$basename..."

   echo 1>&2
   echo "###### Testing $basename" 1>&2

   generatedfiles=""

   generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe ${basename}.out" &&
   Run "$CRAYON" "<" $1 ">" "${basename}.ll" &&
   Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
   Run "$CC" "-o" "${basename}.exe" "${basename}.s" "loadfile.o" &&
   Run "./${basename}.exe" > "${basename}.out" &&
   Compare ${basename}.out ${reffile}.out ${basename}.diff

   # Report the status and clean up the generated files

   if [ $error -eq 0 ] ; then
           if [ $keep -eq 0 ] ; then
              rm -f $generatedfiles
           fi
           echo "OK"
           echo "###### SUCCESS" 1>&2
   else
           echo "###### FAILED" 1>&2
           globalerror=$error
   fi
}

CheckFail() {
   error=0
   basename=`echo $1 | sed 's/.*\\///
                      s/.cry//'`
```

```
    reffile=`echo $1 | sed 's/.cry$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
    RunFail "$CRAYON" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
    Compare ${basename}.err ${reffile}.err ${basename}.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
            if [ $keep -eq 0 ] ; then
               rm -f $generatedfiles
            fi
            echo "OK"
            echo "###### SUCCESS" 1>&2
    else
            echo "###### FAILED" 1>&2
            globalerror=$error
    fi
}

while getopts kdpsh c; do
    case $c in
            k) # Keep intermediate files
               keep=1
               ;;
            h) # Help
               Usage
               ;;
    esac
done

shift `expr $OPTIND - 1`

LLIFail() {
  echo "Could not find the LLVM interpreter \"$LLI\"."
  echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
  exit 1
}

which "$LLI" >> $globallog || LLIFail
```

```
if [ ! -f loadfile.o ]
then
   echo "Could not find loadfile.o"
   echo "Try \"make loadfile.o\""
   exit 1
fi

if [ $# -ge 1 ]
then
   files=$@
else
   files="tests/test_*.cry"
fi

for file in $files
do
   case $file in
            *test_*)
               Check $file 2>> $globallog
               ;;
            *fail_*)
               CheckFail $file 2>> $globallog
               ;;
            *)
               echo "unknown file type $file"
               globalerror=1
               ;;
   esac
done

exit $globalerror
```

Stdlib.cry

```
:( String compare
 Takes in two strings
 Returns 1 if equal, 0 if not :)
boolean strcomp(string s1, string s2) {
 int i;

 i = strcompare(s1, s2);

 if (i == 0) {
  return true;
 }
 return false;
}

:( Square root
  Takes in one integer
  Returns the closest integer square root :)
int sqrt(int n) {
    int i;
    int j;
    if (n < 0) {
      print_string("n must be greater than 0");
      return -1;
    }
    else if (n == 1) {
      return 1;
    }
    j = 1 + (n / 2);

    for (i = 0; i < j; i = i + 1) {
        if (i * i == n) {
            return i;
        }
        else if (i * i > n) {
            return i - 1;
        }
    }
    return i;
}

:( Square
  Takes in one integer
  Returns square of argument :)
int sq(int n) {
  return n * n;
}

:( Print array
  Takes in one array pointer
  Prints array :)
void printArray(array $ x, int l) {
  int i;
  print_string("[ ");
  for (i = 0; i < l; i = i + 1) {
    print(~x);
    x = @1@ x;
  }
  print_string("]");
}
```

```
:( Backwards Array Printing
  Takes in one array pointer
  Prints array backwards :)
void printArrayBackwards(array $ x, int l) {
  int i;

  x = @4@ x;

  print_string("[ ");
  for (i = 0; i < l; i = i + 1) {
    print(~x);
    x = @-1@ x;
  }
  print_string("]");
}

:( Array averaging
  Takes in two array pointers and the lengths of the arrays
  Chanes array of first argument by replacing its values with averages of its inputs
  Error Handling: Checks if arrays are the same length :)
void arrayAvg(array $ a1, array $ a2, int l1, int l2) {
  int i;
  int avg;
  int x2;

  array $ p1;
  array $ p2;

  if (l1 != l2) {
    print_string("Error: arrays must have the same length");
  }
  else {

    for (i = 0; i < l1; i = i + 1) {

      p1 = @i@ a1;
      p2 = @i@ a2;
      avg = sqrt( sq(~p1) + sq(~p2) );

      ~p1 = avg;

    }
  }
}

:( Canvas averaging
  Takes in two canvas pointers and the row number and column number of each array
```

```
 Changes canvas of first argument by replacing its values with averages of its inputs
 Error Handling: Checks if arrays are the same length :)
void canvasAvg(canvas $ x1, canvas $ x2, int r1, int c1, int r2, int c2) {
 int i;
 int avg;

 canvas $ p1;
 canvas $ p2;

 if (r1 != r2 || c1 != c2) {
  print_string("Error: canvases must have the same size");
 }
 else {

  for (i = 0; i < r1 * c1 * 3; i = i + 1) {

   p1 = @i@ x1;
   p2 = @i@ x2;
   avg = (~p1 + ~p2) / 2;

   print_string("***");
   print(~p1);
   print(~p2);
   print(avg);
   print_string("***");

   ~p1 = avg;

  }
 }
}

:( Canvas Fill by RGB
 Takes in a canvas pointer and the row number and column number
 Changes each color value to RGB input of canvas in place :)
void canvasFillRGB(canvas $ x1, int r1, int c1, int R, int G, int B) {
 int i;
 canvas $ p1;
 p1 = x1;

 for (i = 0; i < (r1 * c1); i = i + 1) {
  ~p1 = R;
  p1 = @1@ p1;
  ~p1 = G;
  p1 = @1@ p1;
  ~p1 = B;
  p1 = @1@ p1;
 }
```

```
}

:( Canvas Fill by Color
  Takes in a canvas pointer, the row number and column number, and a color as string
  Instantiates each element to RGB value of color and in place :)
void canvasFillColor(canvas $ x1, int r1, int c1, string color) {
  if (strcomp(color, "red")) {
    canvasFillRGB(x1, r1, c1, 255, 0, 0);
  }
  else if (strcomp(color, "orange")) {
    canvasFillRGB(x1, r1, c1, 255, 128, 0);
  }
  else if (strcomp(color, "yellow")) {
    canvasFillRGB(x1, r1, c1, 255, 255, 0);
  }
  else if (strcomp(color, "green")) {
    canvasFillRGB(x1, r1, c1, 0, 255, 0);
  }
  else if (strcomp(color, "blue")) {
    canvasFillRGB(x1, r1, c1, 0, 0, 255);
  }
  else if (strcomp(color, "indigo")) {
    canvasFillRGB(x1, r1, c1, 75, 0, 130);
  }
  else if (strcomp(color, "violet")) {
    canvasFillRGB(x1, r1, c1, 128, 0, 128);
  }
  else if (strcomp(color, "columbiablue")) {
    canvasFillRGB(x1, r1, c1, 196, 216, 226);
  }
  else if (strcomp(color, "white")) {
    canvasFillRGB(x1, r1, c1, 255, 255, 255);
  }
  else if (strcomp(color, "black")) {
    canvasFillRGB(x1, r1, c1, 0, 0, 0);
  }
  else {
    print_string("this color not found");
  }
}

:( Canvas Blanking
  Takes in a canvas pointer and the row number and column number
  Instantiates each color value to white in place :)
void canvasBlank(canvas $ x1, int r1, int c1) {
    canvasFillColor(x1, r1, c1, "white");
}
```

```
:( Greyscale
  Takes in a canvas pointer and the row number and column number
  Changes canvas color values to average value of each RGB pixel :)
void canvasGreyscale(canvas $ x1, int r1, int c1) {
  int R;
  int G;
  int B;
  int avg;
  int i;
  canvas $ p1;
  canvas $ p2;
  p1 = x1;
  p2 = x1;

  for (i = 0; i < (r1 * c1); i = i + 1) {
    R = ~p1;
    p1 = @1@ p1;
    G = ~p1;
    p1 = @1@ p1;
    B = ~p1;

    avg = (R + G + B) / 3;

    ~p2 = avg;
    p2 = @1@ p2;
    ~p2 = avg;
    p2 = @1@ p2;
    ~p2 = avg;

    p1 = @1@ p1;
    p2 = @1@ p2;
  }
}

:( Draw Rectangle
  Takes in canvas pointer, row number, column number,
  starting row coordinate i, starting col coordinate j with 0,0 starting at top left,
  length of rectangle, height of rectangle
  Changes canvas to draw black rectangle over corresponding pixel :)
void drawRectangle(canvas $ x1, int r1, int c1, int i, int j, int l, int h) {
  int a;
  int b;
  int c;

  canvas $ p1;
  canvas $ q1;
  canvas $ s1;
  p1 = x1;
```

```
if(l > c1) {
  print_string("Length of shape larger than canvas column length. Canvas not changed.");
}
else if(h > r1) {
  print_string("Height of shape larger than canvas row length. Canvas not changed.");
}
else if(i > r1) {
  print_string("Starting row coordinate of shape out of canvas bounds. Canvas not changed.");
}
else if(j > c1) {
  print_string("Starting col coordinate of shape out of canvas bounds. Canvas not changed.");
}
else if(i + h > r1) {
  print_string("Rectangle height will go out of bounds. Canvas not changed.");
}
else if(j + l > c1) {
  print_string("Rectangle width will go out of bounds. Canvas not changed.");
}
else {
  s1 = setPointer(p1, r1, c1, i, j);
  p1 = s1;

  :( Moves left to right, drawing black on pixels :)
  for(a = 0; a < l; a = a + 1) {
    for(b = 0; b < 3; b = b + 1) {
      ~p1 = 0;
      p1 = @1@ p1;
    }
  }

  :( Moves top to bottom, drawing black on pixels :)
  for(a = 0; a < h; a = a + 1) {
    q1 = p1;
    ~p1 = 0;
    q1 = @1@ q1;
    ~q1 = 0;
    q1 = @1@ q1;
    ~q1 = 0;
    for(b = 0; b < c1; b = b + 1) {
      for(c = 0; c < 3; c = c + 1) {
        p1 = @1@ p1;
      }
    }
  }

  :( Moves right to left, drawing black on pixels :)
  for(a = 0; a < l; a = a + 1) {
      for(b = 0; b < 3; b = b + 1) {
        ~p1 = 0;
        p1 = @-1@ p1;
      }
    }

    :( Moves bottom to top, drawing black on pixels :)
    for(a = 0; a < h; a = a + 1) {
      q1 = p1;
      ~p1 = 0;
      q1 = @1@ q1;
      ~q1 = 0;
      q1 = @1@ q1;
      ~q1 = 0;
      for(b = 0; b < c1; b = b + 1) {
        for(c = 0; c < 3; c = c + 1) {
          p1 = @-1@ p1;
        }
      }
    }
  }
}

:( Invert
  Takes in a canvas pointer and the row number and column number
  Changes canvas color values to inverted value of each RGB pixel :)
void canvasInvert(canvas $ x1, int r1, int c1) {
  int i;
  canvas $ p1;
  p1 = x1;

  for (i = 0; i < r1 * c1 * 3; i = i + 1) {
    ~p1 = 255 - ~p1;
    p1 = @1@ p1;
  }
}

:( Set Pointer
  Takes in a canvas pointer and the row number and column number of canvas,
  index i and j for coordinate to set pointer at
  Moves pointer to desired destination and returns :)
canvas $ setPointer(canvas $ x1, int r1, int c1, int i, int j) {
  int a;
  int b;
  int c;
  canvas $ p1;

  p1 = x1;
  :( Moves pointer to correct column :)
  for(a = 0; a < j; a = a + 1) {
    for(b = 0; b < 3; b = b + 1) {
      p1 = @1@ p1;
    }
  }

  :( Moves pointer to correct row :)
  for(a = 0; a < i; a = a + 1) {
    for(b = 0; b < c1; b = b + 1) {
      for(c = 0; c < 3; c = c + 1) {
        p1 = @1@ p1;
      }
    }
  }

  return p1;
}

:( Draw 3 Filled Rectangles
  Takes in canvas pointer, row number, column number,
  starting row coordinate i, starting col coordinate j with 0,0 starting at top left,
  length of rectangle, height of rectangle
  Changes canvas to draw three black rectangles over corresponding pixels :)
void fillThreeRectangles(canvas $ x1, int r1, int c1, int i, int j, int l, int h) {
  int a;
  int b;
  int c;
  int d;

  canvas $ p1;
  canvas $ q1;
  canvas $ z1;
  p1 = x1;

  if(l > c1) {
    print_string("Length of shape larger than canvas column length. Canvas not changed.");
  }
  else if(h > r1) {
    print_string("Height of shape larger than canvas row length. Canvas not changed.");
  }
  else if(i > r1) {
    print_string("Starting x coordinate of shape out of canvas bounds. Canvas not changed.");
  }
  else if(j > c1) {
    print_string("Starting y coordinate of shape out of canvas bounds. Canvas not changed.");
  }
  else if(i + h > r1) {
    print_string("Rectangle height will go out of bounds. Canvas not changed.");
  }
  else if(j + l > c1) {
    print_string("Rectangle width will go out of bounds. Canvas not changed.");
  }
  else {
    for(a = 0; a < j; a = a + 1) {
      for(b = 0; b < 3; b = b + 1) {
        p1 = @1@ p1;
      }
    }

    for(a = 0; a < i; a = a + 1) {
      for(b = 0; b < c1; b = b + 1) {
        for(c = 0; c < 3; c = c + 1) {
          p1 = @1@ p1;
        }
      }
    }

    z1 = p1;
    for (c = 0; c < h; c = c + 1) {
      for(a = 0; a < l; a = a + 1) {
        for(b = 0; b < 3; b = b + 1) {
          ~p1 = 0;
          p1 = @1@ p1;
        }
      }
      d = c1*c;
      print(d);
      p1 = @d@ z1;
    }
  }
}

:( Draw Circle
  Takes in canvas pointer, row number, column number,
  centered at row coordinate x0, starting col coordinate y0 with 0,0 starting at top left,
  radius of circle
  Changes canvas to draw black circle over corresponding pixel,
  Based on Midpoint circle algorithm from Wikipedia :)
void drawCircle(canvas $ x1, int r1, int c1, int x0, int y0, int radius) {
  int x;
  int y;
  int err;
  canvas $ p1;
  canvas $ q1;
```

```
p1 = x1;
q1 = p1;
x = radius;
y = 0;
err = 0;

while (x >= y) {
 p1 = setPointer(q1, r1, c1, x0 + x, y0 + y);
 ~p1 = 0;
 p1 = @1@ p1;
 ~p1 = 0;
 p1 = @1@ p1;
 ~p1 = 0;

 p1 = setPointer(q1, r1, c1, x0 + y, y0 + x);
 ~p1 = 0;
 p1 = @1@ p1;
 ~p1 = 0;
 p1 = @1@ p1;
 ~p1 = 0;

 p1 = setPointer(q1, r1, c1, x0 - y, y0 + x);
 ~p1 = 0;
 p1 = @1@ p1;
 ~p1 = 0;
 p1 = @1@ p1;
 ~p1 = 0;

 p1 = setPointer(q1, r1, c1, x0 - x, y0 + y);
 ~p1 = 0;
 p1 = @1@ p1;
 ~p1 = 0;
 p1 = @1@ p1;
 ~p1 = 0;

 p1 = setPointer(q1, r1, c1, x0 - x, y0 - y);
 ~p1 = 0;
 p1 = @1@ p1;
 ~p1 = 0;
 p1 = @1@ p1;
 ~p1 = 0;

 p1 = setPointer(q1, r1, c1, x0 - y, y0 - x);
 ~p1 = 0;
 p1 = @1@ p1;
 ~p1 = 0;
 p1 = @1@ p1;
```

```
  ~p1 = 0;

  p1 = setPointer(q1, r1, c1, x0 + y, y0 - x);
  ~p1 = 0;
  p1 = @1@ p1;
  ~p1 = 0;
  p1 = @1@ p1;
  ~p1 = 0;

  p1 = setPointer(q1, r1, c1, x0 + x, y0 - y);
  ~p1 = 0;
  p1 = @1@ p1;
  ~p1 = 0;
  p1 = @1@ p1;
  ~p1 = 0;

  y = y + 1;
  if (err <= 0) {
    err = err + 2*y + 1;
  }
  if (err > 0) {
    x = x - 1;
    err = err - 2*x + 1;
  }
 }
}
```

README.md

# crayon
Crayon is a C-style programming language for creating raster graphics, compiled with OCaml into LLVM

## Introduction
Crayon is a raster-graphics creation language that simplifies the digital painting of images through code. Raster graphics are images created from two-dimensional arrays of hex codes that represent the rectangular grid of pixels we see for many computer graphics. Inspired by this paradigm, the Crayon programming language provides a way for the user to create any sort of pixelated image (in color) that they can imagine - like the beautiful creations often made on Microsoft Paint, but worse.

The crux of Crayon's design is the assigning of pixel color values to an array-like Canvas. In the same way that a painter assigns different colors to sections of a physical canvas, so can a programmer to Crayon's digital canvas. The main objective for the programmer should be the artistic or systematic implementation of color.

## Todo
#### Standard Library
- Lines

- ~~Shapes (circle, square, rectangle, triangle)~~
- Scaling
- Rotation
- Cutting and pasting images
- ~~Printing arrays, canvas~~
- ~~Adding matrices, arrays~~
- ~~Instantiate canvas with all white (0, 0, 0)~~
- ~~Fill color~~
- ~~Color effects (invert, black and white, sepia)~~

#### Compiler Stuff
- Exceptions (access out of bound)
- Proofread code
- All tests
- ~~Height and length~~
- ~~Pointers for Canvas~~
- ~~Pointer ref/deref for expressions~~
- ~~Load PPM file from Canvas~~
- Access rows, columns

## Installation (from The MicroC compiler)

Coded in OCaml, this takes a highly stripped-down subset of C (ints,
bools, and void types, arithmetic, if-else, for, and while statements,
and user-defined functions) and compiles it into LLVM IR.

It needs the OCaml llvm library, which is most easily installed through opam.

Install LLVM and its development libraries, the m4 macro preprocessor,
and opam, then use opam to install llvm.

The version of the OCaml llvm library should match the version of the LLVM
system installed on your system.

In addition to print, which calls the C library function printf(),
this gratuitiously includes a primitive function "printbig," which
prints large ASCII-encoded characters.

The stock C compiler compiles printbig.o. testall.sh runs the microc
executable on each testcase (.mc file) to produce a .ll file, invokes
"llc" (the LLVM compiler) to produce a .s (assembly) file, then
invokes "cc" (the stock C compiler) to assemble the .s file, link in
printbig.o, and generate an executable. See testall.sh for details.

Installation under Ubuntu 15.10

LLVM 3.6 is the default under 15.10, so we ask for a matching version of the

OCaml library.

```
sudo apt-get install -y ocaml m4 llvm opam
opam init
opam install llvm.3.6 ocamlfind
eval `opam config env`

make
./testall.sh
```

Installation under Ubuntu 14.04

The default LLVM package is 3.4, so we install the matching OCaml
library using opam. The default version of opam under 14.04 is too
old; we need to use a newer package.

```
sudo apt-get install m4 llvm software-properties-common

sudo add-apt-repository --yes ppa:avsm/ppa
sudo apt-get update -qq
sudo apt-get install -y opam
opam init

eval `opam config env`

opam install llvm.3.4 ocamlfind
```

Installation under OS X

1. Install Homebrew:

```
  ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

2. Verify Homebrew is installed correctly:

```
  brew doctor
```

3. Install opam:

```
  brew install opam
```

4. Set up opam:

```
  opam init
```

5. Install llvm:

```
brew install llvm
```

Take note of where brew places the llvm executables. It will show you the path to them under the CAVEATS section of the post-install terminal output. For me, they were in /usr/local/opt/llvm/bin. Also take note of the llvm version installed. For me, it was 3.6.2.

6. Have opam set up your enviroment:

```
eval `opam config env`
```

7. Install the OCaml llvm library:

```
opam install llvm.3.6
```

Ensure that the version of llvm you install here matches the version you installed via brew. Brew installed llvm version 3.6.2, so I install llvm.3.6 with opam.

IF YOU HAVE PROBLEMS ON THIS STEP, it's probably because you are missing some external dependencies. Ensure that libffi is installed on your machine. It can be installed with

```
brew install libffi
```

If, after this, opam install llvm.3.6 is still not working, try running

```
opam list --external --required-by=llvm.3.6
```

This will list all of the external dependencies required by llvm.3.6. Install all the dependencies listed by this command.

IF THE PREVIOUS STEPS DO NOT SOLVE THE ISSUE, it may be a problem with using your system's default version of llvm. Install a different version of llvm and opam install llvm with that version by running:

```
brew install homebrew/versions/llvm37
opam install llvm.3.7
```

Where the number at the end of both commands is a version different from the one your system currently has.

8. Create a symbolic link to the lli command:

```
sudo ln -s /usr/local/opt/llvm/bin/lli /usr/bin/lli
```

Create the symlink from wherever brew installs the llvm executables
and place it in your bin. From step 5, I know that brew installed
the lli executable in the folder, /usr/local/opt/llvm/bin/, so this
is where I symlink to. Brew might install the lli executables in a
different location for you, so make sure you symlink to the right
directory.

IF YOU GET OPERATION NOT PERMITTED ERROR, then this is probably a
result of OSX's System Integrity Protection.

One way to get around this is to reboot your machine into recovery
mode (by holding cmd-r when restarting). Open a terminal from
recovery mode by going to Utilities -> Terminal, and enter the
following commands:

```
csrutil disable
reboot
```

After your machine has restarted, try the `ln....` command again,
and it should succeed.

IMPORTANT: the prevous step disables System Integrity Protection,
which can leave your machine vulnerable. It's highly advisable to
reenable System Integrity Protection when you are done by
rebooting your machine into recovery mode and entering the following
command in the terminal:

```
csrutil enable
reboot
```

Another solution is to update your path, e.g.,

```
export PATH=$PATH:/usr/local/opt/llvm/bin
```

A third solution is to modify the definition of LLI in testall.sh to
point to the absolute path, e.g., LLI="/usr/local/opt/llvm/bin/lli"

9. To run and test, navigate to the MicroC folder. Once there, run

```
make ; ./testall.sh
```

MicroC should build without any complaints and all tests should
pass.

IF RUNNING ./testall.sh FAILS ON SOME TESTS, check to make sure you
have symlinked the correct executable from your llvm installation.
For example, if the executable is named lli-[version], then the

previous step should have looked something like:

```
sudo ln -s /usr/local/opt/llvm/bin/lli-3.7 /usr/bin/lli
```

As before, you may also modify the path to lli in testall.sh

To run and test:

```
$ make
ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis -cflags -w,+a-4 microc.native
Finished, 22 targets (0 cached) in 00:00:01.
cc    -c -o printbig.o printbig.c
$ ./testall.sh
test-arith1...OK
test-arith2...OK
test-arith3...OK
test-fib...OK
...
fail-while1...OK
fail-while2...OK
```

Test_arith_a.cry

```
:( to test arithmetic :)

int main()
{
  print(10+5);
  return 0;
}
```

Test_arith_b.cry

```
:(to test order of operations:)

int main()
{
  print(5 + 10 * 2);
  return 0;
}
```

Test_arith_c.cry

```
:(tests arithmatic with variables :)

int main()
{
	int a;
	a = 10;
```

```
            a = a + 5;
            print(a);
            return(0);
}
```

Test_array_a.cry

```
:( tests assigning integers to an array  :)

int main() {

            array int[5] a;
            int i;

            for(i = 0; i<5; i = i+1){
                        a[i] = i + 1;
                        print(a[i]);
            }

            return 0;
}
```

Test_array_b.cry

```
:( tests re-assigning integers to an array  :)

int main() {

            array int[5] a;
            int i;

            for(i = 0; i<5; i = i+1){
                        a[i] = i + 1;
            }

            for(i = 0; i<5; i = i+1){
                        a[i] = a[i] + 1;
                        print(a[i]);
            }

            return(0);
}
```

Test_bool.cry

```
:( tests reassignment of boolean variables :)

int main()
{
            boolean b;
            boolean a;
```

```
        b = true;
        a = false;
        a = b;
        if(a){ print(1);}
        return 0;
}
```

Test_canvas_a.cry

```
:( Tests assignment of colors to canvas :)

int main() {

  canvas [5,5] g;
  array int[3] a;
  array int[3] b;

  g[0,0] = [251, 252, 253];
  g[0,1] = [1, 2, 3];

  a = g[0,0];
  b = g[0,1];

  print(a[0]);
  print(b[0]);

  return 0;
}
```

Test_canvas_b.cry

```
:( Tests re-assignment of colors to canvas :)

int main() {

  canvas [5,5] g;
  array int[3] a;
  array int[3] b;
  canvas $ p;

  g[0,0] = [251, 252, 253];
  g[0,1] = [1, 2, 3];

  a = g[0,0];
  b = g[0,1];

  print(a[0]);
  print(b[0]);

  p = &g;
  ~p = 255;

  print(~p);

  return 0;
}
```

Test_canvas_c.cry

```
:( Tests read/assignment of colors to canvas using indeces :)

int main() {

  canvas [5,5] g;
  array int[3] a;
  array int[3] b;

  g[0,0] = [251, 252, 253];
  g[0,1] = [1, 2, 3];

  print(g[0,0,0]);
  print(g[0,1,1]);

  g[0,0,0] = 0;
  g[0,1,1] = 0;

  print(g[0,0,0]);
  print(g[0,1,1]);

  return 0;
}
```

Test_demo.cry

```
:( Crayon Demo :)

int main() {
  canvas [100,100] canv1;
  canvas [100,100] canv2;
  canvas [100,100] canv3;

  canvas $ p1;
  canvas $ p2;
  canvas $ p3;

  int r1;
  int c1;

  int r2;
  int c2;
  int r3;
  int c3;

  string s;
  boolean b;

  r1 = rows(canv1);
  r2 = rows(canv2);
  r3 = rows(canv3);

  c1 = columns(canv1);
  c2 = columns(canv2);
  c3 = columns(canv3);

  p1 = &canv1;
  p2 = &canv2;
  p3 = &canv3;

  canvasFillColor(p1, r1, c1, "red");
  writefile(p1, r1, c1, "red.ppm");

  canvasFillColor(p2, r2, c2, "blue");
  fill3Rectangles(p2, r2, c2, 20, 20, 30, 40);
  writefile(p2, r2, c2, "blueandrectangles.ppm");

  canvasAvg(p1, p2, r1, c1, r2, c2);
  writefile(p1, r1, c1, "purpleandrectangles.ppm");

  return 0;
}

:( String compare
  Takes in two strings
  Returns 1 if equal, 0 if not :)
boolean strcomp(string s1, string s2) {
  int i;

  i = strcompare(s1, s2);

  if (i == 0) {
    return true;
  }
  return false;

}

:( Square root
  Takes in one integer
  Returns the closest integer square root :)
int sqrt(int n) {
    int i;
    int j;
    if (n < 0) {
      print_string("n must be greater than 0");
      return -1;
    }
    else if (n == 1) {
      return 1;
    }
    j = 1 + (n / 2);

    for (i = 0; i < j; i = i + 1) {
        if (i * i == n) {
            return i;
        }
        else if (i * i > n) {
            return i - 1;
        }
    }
    return i;
}

:( Square
  Takes in one integer
  Returns square of argument :)
int sq(int n) {
  return n * n;
}

:( Print array
  Takes in one array pointer
  Prints array :)
void printArray(array $ x, int l) {
  int i;
  print_string("[ ");
  for (i = 0; i < l; i = i + 1) {
    print(~x);
    x = @1@ x;
  }
  print_string("]");
}

:( Backwards Array Printing
  Takes in one array pointer
  Prints array backwards :)
```

```
void printArrayBackwards(array $ x, int l) {
  int i;

  x = @4@ x;

  print_string("[ ");
  for (i = 0; i < l; i = i + 1) {
    print(~x);
    x = @-1@ x;
  }
  print_string("]");
}

:( Array averaging
  Takes in two array pointers and the lengths of the arrays
  Chanes array of first argument by replacing its values with averages of its inputs
  Error Handling: Checks if arrays are the same length :)
void arrayAvg(array $ a1, array $ a2, int l1, int l2) {
  int i;
  int avg;
  int x2;

  array $ p1;
  array $ p2;

  if (l1 != l2) {
    print_string("Error: arrays must have the same length");
  }
  else {

    for (i = 0; i < l1; i = i + 1) {

      p1 = @i@ a1;
      p2 = @i@ a2;
      avg = sqrt( sq(~p1) + sq(~p2) );

      ~p1 = avg;

    }
  }
}

:( Canvas averaging
  Takes in two canvas pointers and the row number and column number of each array
  Changes canvas of first argument by replacing its values with averages of its inputs
  Error Handling: Checks if arrays are the same length :)
void canvasAvg(canvas $ x1, canvas $ x2, int r1, int c1, int r2, int c2) {
  int i;
  int avg;

  canvas $ p1;
  canvas $ p2;

  if (r1 != r2 || c1 != c2) {
    print_string("Error: canvases must have the same size");
  }
  else {

    for (i = 0; i < r1 * c1 * 3; i = i + 1) {

      p1 = @i@ x1;
      p2 = @i@ x2;
      avg = (~p1 + ~p2) / 2;

      print_string("***");
      print(~p1);
      print(~p2);
      print(avg);
      print_string("***");

      ~p1 = avg;

    }
  }
}

:( Canvas Fill by RGB
  Takes in a canvas pointer and the row number and column number
  Changes each color value to RGB input of canvas in place :)
void canvasFillRGB(canvas $ x1, int r1, int c1, int R, int G, int B) {
  int i;
  canvas $ p1;
  p1 = x1;

  for (i = 0; i < (r1 * c1); i = i + 1) {
    ~p1 = R;
    p1 = @1@ p1;
    ~p1 = G;
    p1 = @1@ p1;
    ~p1 = B;
    p1 = @1@ p1;
  }
}

:( Canvas Fill by Color
  Takes in a canvas pointer, the row number and column number, and a color as string
  Instantiates each element to RGB value of color and in place :)
void canvasFillColor(canvas $ x1, int r1, int c1, string color) {
  if (strcomp(color, "red")) {
    canvasFillRGB(x1, r1, c1, 255, 0, 0);
  }
  else if (strcomp(color, "orange")) {
    canvasFillRGB(x1, r1, c1, 255, 128, 0);
  }
  else if (strcomp(color, "yellow")) {
    canvasFillRGB(x1, r1, c1, 255, 255, 0);
  }
  else if (strcomp(color, "green")) {
    canvasFillRGB(x1, r1, c1, 0, 255, 0);
  }
  else if (strcomp(color, "blue")) {
    canvasFillRGB(x1, r1, c1, 0, 0, 255);
  }
  else if (strcomp(color, "indigo")) {
    canvasFillRGB(x1, r1, c1, 75, 0, 130);
  }
  else if (strcomp(color, "violet")) {
    canvasFillRGB(x1, r1, c1, 128, 0, 128);
  }
  else if (strcomp(color, "columbiablue")) {
    canvasFillRGB(x1, r1, c1, 196, 216, 226);
  }
  else if (strcomp(color, "white")) {
    canvasFillRGB(x1, r1, c1, 255, 255, 255);
  }
  else if (strcomp(color, "black")) {
    canvasFillRGB(x1, r1, c1, 0, 0, 0);
  }
  else {
    print_string("this color not found");
  }
}

:( Canvas Blanking
  Takes in a canvas pointer and the row number and column number
  Instantiates each color value to white in place :)
void canvasBlank(canvas $ x1, int r1, int c1) {
    canvasFillColor(x1, r1, c1, "white");
}

:( Greyscale
  Takes in a canvas pointer and the row number and column number
  Changes canvas color values to average value of each RGB pixel :)
void canvasGreyscale(canvas $ x1, int r1, int c1) {
  int R;
  int G;
  int B;
  int avg;
  int i;
  canvas $ p1;
  canvas $ p2;
  p1 = x1;
  p2 = x1;

  for (i = 0; i < (r1 * c1); i = i + 1) {
    R = ~p1;
    p1 = @1@ p1;
    G = ~p1;
    p1 = @1@ p1;
    B = ~p1;

    avg = (R + G + B) / 3;

    ~p2 = avg;
    p2 = @1@ p2;
    ~p2 = avg;
    p2 = @1@ p2;
    ~p2 = avg;

    p1 = @1@ p1;
    p2 = @1@ p2;
  }
}

:( Draw Rectangle
  Takes in canvas pointer, row number, column number,
  starting row coordinate i, starting col coordinate j with 0,0 starting at top left,
  length of rectangle, height of rectangle
  Changes canvas to draw black rectangle over corresponding pixel :)
void drawRectangle(canvas $ x1, int r1, int c1, int i, int j, int l, int h) {
  int a;
  int b;
  int c;

  canvas $ p1;
  canvas $ q1;
  p1 = x1;

  if(l > c1) {
    print_string("Length of shape larger than canvas column length. Canvas not changed.");
  }
  else if(h > r1) {
```

```
  print_string("Height of shape larger than canvas row length. Canvas not changed.");
}
else if(i > r1) {
  print_string("Starting x coordinate of shape out of canvas bounds. Canvas not changed.");
}
else if(j > c1) {
  print_string("Starting y coordinate of shape out of canvas bounds. Canvas not changed.");
}
else if(i + h > r1) {
  print_string("Rectangle height will go out of bounds. Canvas not changed.");
}
else if(j + l > c1) {
  print_string("Rectangle width will go out of bounds. Canvas not changed.");
}
else {
  :( Moves pointer to correct column :)
  for(a = 0; a < j; a = a + 1) {
    for(b = 0; b < 3; b = b + 1) {
      p1 = @1@ p1;
    }
  }

  :( Moves pointer to correct row :)
  for(a = 0; a < i; a = a + 1) {
    for(b = 0; b < c1; b = b + 1) {
      for(c = 0; c < 3; c = c + 1) {
        p1 = @1@ p1;
      }
    }
  }

  :( Moves left to right, drawing black on pixels :)
  for(a = 0; a < l; a = a + 1) {
    for(b = 0; b < 3; b = b + 1) {
      ~p1 = 0;
      p1 = @1@ p1;
    }
  }

  :( Moves top to bottom, drawing black on pixels :)
  for(a = 0; a < h; a = a + 1) {
    q1 = p1;
    ~p1 = 0;
    q1 = @1@ q1;
    ~q1 = 0;
    q1 = @1@ q1;
    ~q1 = 0;
    for(b = 0; b < c1; b = b + 1) {
        for(c = 0; c < 3; c = c + 1) {
          p1 = @1@ p1;
        }
      }
    }

    :( Moves right to left, drawing black on pixels :)
    for(a = 0; a < l; a = a + 1) {
      for(b = 0; b < 3; b = b + 1) {
        ~p1 = 0;
        p1 = @-1@ p1;
      }
    }

    :( Moves bottom to top, drawing black on pixels :)
    for(a = 0; a < h; a = a + 1) {
      q1 = p1;
      ~p1 = 0;
      q1 = @1@ q1;
      ~q1 = 0;
      q1 = @1@ q1;
      ~q1 = 0;
      for(b = 0; b < c1; b = b + 1) {
        for(c = 0; c < 3; c = c + 1) {
          p1 = @-1@ p1;
        }
      }
    }
  }
}

void fill3Rectangles(canvas $ x1, int r1, int c1, int i, int j, int l, int h) {
  int a;
  int b;
  int c;
  int d;

  canvas $ p1;
  canvas $ q1;
  canvas $ z1;
  p1 = x1;

  if(l > c1) {
    print_string("Length of shape larger than canvas column length. Canvas not changed.");
  }
  else if(h > r1) {
    print_string("Height of shape larger than canvas row length. Canvas not changed.");
  }
  else if(i > r1) {
    print_string("Starting x coordinate of shape out of canvas bounds. Canvas not changed.");
  }
  else if(j > c1) {
    print_string("Starting y coordinate of shape out of canvas bounds. Canvas not changed.");
  }
  else if(i + h > r1) {
    print_string("Rectangle height will go out of bounds. Canvas not changed.");
  }
  else if(j + l > c1) {
    print_string("Rectangle width will go out of bounds. Canvas not changed.");
  }
  else {
    for(a = 0; a < j; a = a + 1) {
      for(b = 0; b < 3; b = b + 1) {
        p1 = @1@ p1;
      }
    }

    for(a = 0; a < i; a = a + 1) {
      for(b = 0; b < c1; b = b + 1) {
        for(c = 0; c < 3; c = c + 1) {
          p1 = @1@ p1;
        }
      }
    }

    z1 = p1;
    for (c = 0; c < h; c = c + 1) {
      for(a = 0; a < l; a = a + 1) {
        for(b = 0; b < 3; b = b + 1) {
          ~p1 = 0;
          p1 = @1@ p1;
        }
      }
      d = c1*c;
      print(d);
      p1 = @d@ z1;
    }
  }
}
```

Test_fib.cry

```
:(tests recursion:)

int fib(int x)
{
  if (x < 2) return 1;
  return fib(x-1) + fib(x-2);
}

int main()
{
  print(fib(0));
  print(fib(1));
  print(fib(2));
  print(fib(3));
  print(fib(4));
  print(fib(5));
  return 0;
}
```

Test_for_a.cry

```
:( tests iteration through for loop :)

int main()
{
  int i;
  for (i = 0 ; i < 5 ; i = i + 1) {
    print(i);
  }
  print(42);
  return 0;
}
```

Test_for_b.cry

```
:( tests a for loop without incrementer :)

int main()
{
  int i;
  i = 0;
  for ( ; i < 5; ) {
    print(i);
    i = i + 1;
  }
  print(42);
  return 0;
}
```

Test_fun_a.cry

```
:(to test if add and functions work:)

int add(int x, int y)
{
  return x + y;
}

int main()
{
  int a;
  a = add(17, 25);
  print(a);
  print( add(17, 25) );
  return 0;
}
```

Test_fun_b.cry

```
:( tests scopes :)

int fun(int x, int y)
{
  return 0;
}

int main()
{
  int i;
  i = 1;

  fun(i = 2, i = i+1);

  print(i);
  return 0;
}
```

Test_fun_c.cry

```
:(tests value passing to function:)

void printem(int a, int b, int c, int d)
{
  print(a);
  print(b);
  print(c);
  print(d);
}

int main()
{
  printem(42,17,192,8);
  return 0;
}
```

Test_fun_d.cry

```
:(tests scope:)

int a;

void foo(int c)
{
  a = c + 42;
}

int main()
{
  foo(73);
  print(a);
  return 0;
}
```

Test_global_a.cry

```
int a;
int b;

void printa()
{
  print(a);
}

void printb()
{
  print(b);
}

void incab()
{
  a = a + 1;
  b = b + 1;
}

int main()
{
  a = 42;
  b = 21;
  printa();
  printb();
  incab();
  printa();
  printb();
  return 0;
}
```

Test_global_b.cry

```
:( Should hide the global i :)

boolean i;

int main()
{
  int i;

  i = 42;
  print(i + i);
  return 0;
}
```

Test_global_c.cry

```
:(tests correct boolean evaluation :)

int i;
boolean b;
int j;

int main()
{
  i = 42;
  j = 10;
  print(i + j);
  return 0;
}
```

Test_if_a.cry

```
:(tests correct boolean evaluation :)

int main()
{
  if (true) print(42);
  print(17);
  return 0;
}
```

Test_if_b.cry

```
:(tests correct boolean evaluation :)

int main()
{
  if (true) print(42); else print(8);
  print(17);
  return 0;
}
```

Test_if_c.cry

```
:(tests correct boolean evaluation :)

int main()
{
  if (false) print(42);
  print(17);
  return 0;
}
```

Test_if_d.cry

```
:(tests correct boolean evaluation :)

int main()
{
  if (false) print(42); else print(8);
  print(17);
  return 0;
}
```

Test_if_e.cry

```
:(tests correct boolean evaluation :)

int cond(boolean b)
{
  int x;
  if (b)
    x = 42;
  else
    x = 17;
  return x;
}

int main()
{
 print(cond(true));
 print(cond(false));
 return 0;
}
```

Test_local_a.cry

```
:(tests correct boolean evaluation :)

void foo(boolean i)
{
  int i;
  i = 42;
  print(i + i);
}

int main()
{
  foo(true);
  return 0;
}
```

Test_local_b.cry

```
int foo(int a, boolean b)
{
  int c;
  boolean d;

  c = a;

  return c + 10;
}

int main() {
 print(foo(37, false));
 return 0;
}
```

Test_ops_a.cry

```
int main()
{
  print(1 + 2);
  print(1 - 2);
  print(1 * 2);
  print(100 / 2);
  print(99);
  printb(1 == 2);
  printb(1 == 1);
  print(99);
  printb(1 != 2);
  printb(1 != 1);
  print(99);
  printb(1 < 2);
  printb(2 < 1);
  print(99);
  printb(1 <= 2);
  printb(1 <= 1);
  printb(2 <= 1);
  print(99);
  printb(1 > 2);
  printb(2 > 1);
  print(99);
  printb(1 >= 2);
  printb(1 >= 1);
  printb(2 >= 1);
  return 0;
}
```

Test_ops_b.cry

```
int main()
{
  printb(true);
  printb(false);
  printb(true && true);
  printb(true && false);
  printb(false && true);
  printb(false && false);
  printb(true || true);
  printb(true || false);
  printb(false || true);
  printb(false || false);
  printb(!false);
  printb(!true);
  print(-10);
  print(--42);
}
```

Test_pointer_a.cry

```
:( tests pointer arithmetic :)

int main() {
  array int[3] arr;
  array $ ap;
  int i;

  arr = [1,2,3];
  ap = &arr;

  print(~ap);

  i = 0;

  ap = @i + 1@ ap;

  print(~ap);

  return 0;
}
```

Test_pointer_b.cry

```
:( tests pointer assignment/ dereference :)

int main(){
        array int[3] a;
        array $ ptr;

        a = [0, 1, 2];

        ptr = &a;
        ~ptr = 16;

        print(a[0]);
        print(~ptr);

        return(0);
}
```

Test_readfile.cry

```
int main()
{
  int row;
  int r;
  int col;
  int co;
  int i;
  int j;
  int k;
  int l;
  int m;
  int n;
  canvas $ p;
  canvas $ a;
  canvas $ q;
  canvas $ b;
  canvas[100, 100] c;
  canvas[100, 100] d;
  p = readfile("red.ppm");
  a = readfile("blueandrectangles.ppm");

  row = ~p;
  p = @1@ p;
  col = ~p;
  print(row);
  print(col);


  for(i = 0; i < 100; i = i + 1) {
   for(j = 0; j < 100; j = j + 1) {
     for(k = 0; k < 3; k = k + 1) {
       if(i<row && j<col){
         p = @1@ p;
         c[i, j, k] = ~p;
       }
       else{
         c[i, j, k] = 255;
       }

     }
   }
 }
 q = &c;
 writefile(q, 100, 100,"newimage_a.ppm");

 r = ~a;
 a = @1@ a;
 co = ~a;
 print(r);
 print(co);


 for(l = 0; l < 100; l = l + 1) {
   for(m = 0; m < 100; m = m + 1) {
     for(n = 0; n < 3; n = n + 1) {
       if(l<r && m<co){
         a = @1@ a;
         d[l, m, n] = ~a;
       }
       else{
         d[l, m, n] = 255;
       }

     }
   }
 }
 b = &d;
 writefile(b, 100, 100,"newimage_b.ppm");
 return 0;
}
```

Test_stdlib.cry

```
:( Crayon Standard Library :)

int main() {
  canvas [100,100] canv1;
  canvas [100,100] canv2;
  canvas [100,100] canv3;
  canvas $ p1;
  canvas $ p2;
  canvas $ p3;
  int r1;
  int c1;
  int r2;
  int c2;
  int r3;
  int c3;
  string s;
  boolean b;

  r1 = rows(canv1);
  r2 = rows(canv2);
  r3 = rows(canv3);

  c1 = columns(canv1);
  c2 = columns(canv2);
  c3 = columns(canv3);

  p1 = &canv1;
  p2 = &canv2;
  p3 = &canv3;

  canvasFillColor(p1, r1, c1, "red");
  writefile(p1, r1, c1, "red.ppm");

  canvasFillColor(p2, r2, c2, "columbiablue");
  :( drawRectangle(p2, r2, c2, 20, 20, 30, 40); :)
  drawCircle(p2, r2, c2, 50, 50, 15);
  writefile(p2, r2, c2, "blueandrectangles.ppm");

  canvasAvg(p1, p2, r1, c1, r2, c2);
  canvasInvert(p1, r1, c2);
  writefile(p1, r1, c1, "invertedpurpleandrectangles.ppm");

  return 0;
}

:( String compare
  Takes in two strings
  Returns 1 if equal, 0 if not :)
boolean strcomp(string s1, string s2) {
  int i;

  i = strcompare(s1, s2);
  if (i == 0) {
    return true;
  }
  return false;

}

:( Square root
  Takes in one integer
  Returns the closest integer square root :)
int sqrt(int n) {
    int i;
    int j;
    if (n < 0) {
      print_string("n must be greater than 0");
      return -1;
    }
    else if (n == 1) {
      return 1;
    }
    j = 1 + (n / 2);

    for (i = 0; i < j; i = i + 1) {
        if (i * i == n) {
            return i;
        }
        else if (i * i > n) {
            return i - 1;
        }
    }
    return i;
}

:( Square
  Takes in one integer
  Returns square of argument :)
int sq(int n) {
  return n * n;
}

:( Print array
  Takes in one array pointer
  Prints array :)
void printArray(array $ x, int l) {
  int i;
  print_string("[ ");
  for (i = 0; i < l; i = i + 1) {
    print(~x);
    x = @1@ x;
  }
  print_string("]");
}

:( Backwards Array Printing
  Takes in one array pointer
  Prints array backwards :)
void printArrayBackwards(array $ x, int l) {
  int i;

  x = @4@ x;

  print_string("[ ");
  for (i = 0; i < l; i = i + 1) {
    print(~x);
    x = @-1@ x;
  }
  print_string("]");
}

:( Array averaging
  Takes in two array pointers and the lengths of the arrays
  Chanes array of first argument by replacing its values with averages of its inputs
  Error Handling: Checks if arrays are the same length :)
void arrayAvg(array $ a1, array $ a2, int l1, int l2) {
  int i;
  int avg;
  int x2;

  array $ p1;
  array $ p2;

  if (l1 != l2) {
    print_string("Error: arrays must have the same length");
  }
  else {

    for (i = 0; i < l1; i = i + 1) {

      p1 = @i@ a1;
      p2 = @i@ a2;
      avg = sqrt( sq(~p1) + sq(~p2) );

      ~p1 = avg;

    }
  }
```

```
}

:( Canvas averaging
  Takes in two canvas pointers and the row number and column number of each array
  Changes canvas of first argument by replacing its values with averages of its inputs
  Error Handling: Checks if arrays are the same length :)
void canvasAvg(canvas $ x1, canvas $ x2, int r1, int c1, int r2, int c2) {
  int i;
  int avg;

  canvas $ p1;
  canvas $ p2;

  if (r1 != r2 || c1 != c2) {
    print_string("Error: canvases must have the same size");
  }
  else {

    for (i = 0; i < r1 * c1 * 3; i = i + 1) {

      p1 = @i@ x1;
      p2 = @i@ x2;
      avg = (~p1 + ~p2) / 2;

      print_string("***");
      print(~p1);
      print(~p2);
      print(avg);
      print_string("***");

      ~p1 = avg;

    }
  }
}

:( Canvas Fill by RGB
  Takes in a canvas pointer and the row number and column number
  Changes each color value to RGB input of canvas in place :)
void canvasFillRGB(canvas $ x1, int r1, int c1, int R, int G, int B) {
  int i;
  canvas $ p1;
  p1 = x1;

  for (i = 0; i < (r1 * c1); i = i + 1) {
    ~p1 = R;
    p1 = @1@ p1;
    ~p1 = G;
    p1 = @1@ p1;
    ~p1 = B;
    p1 = @1@ p1;
  }
}

:( Canvas Fill by Color
  Takes in a canvas pointer, the row number and column number, and a color as string
  Instantiates each element to RGB value of color and in place :)
void canvasFillColor(canvas $ x1, int r1, int c1, string color) {
  if (strcomp(color, "red")) {
    canvasFillRGB(x1, r1, c1, 255, 0, 0);
  }
  else if (strcomp(color, "orange")) {
    canvasFillRGB(x1, r1, c1, 255, 128, 0);
  }
  else if (strcomp(color, "yellow")) {
    canvasFillRGB(x1, r1, c1, 255, 255, 0);
  }
  else if (strcomp(color, "green")) {
    canvasFillRGB(x1, r1, c1, 0, 255, 0);
  }
  else if (strcomp(color, "blue")) {
    canvasFillRGB(x1, r1, c1, 0, 0, 255);
  }
  else if (strcomp(color, "indigo")) {
    canvasFillRGB(x1, r1, c1, 75, 0, 130);
  }
  else if (strcomp(color, "violet")) {
    canvasFillRGB(x1, r1, c1, 128, 0, 128);
  }
  else if (strcomp(color, "columbiablue")) {
    canvasFillRGB(x1, r1, c1, 196, 216, 226);
  }
  else if (strcomp(color, "white")) {
    canvasFillRGB(x1, r1, c1, 255, 255, 255);
  }
  else if (strcomp(color, "black")) {
    canvasFillRGB(x1, r1, c1, 0, 0, 0);
  }
  else {
    print_string("this color not found");
  }
}

:( Canvas Blanking
  Takes in a canvas pointer and the row number and column number
  Instantiates each color value to white in place :)
```

```
void canvasBlank(canvas $ x1, int r1, int c1) {
   canvasFillColor(x1, r1, c1, "white");
}

:( Greyscale
  Takes in a canvas pointer and the row number and column number
  Changes canvas color values to average value of each RGB pixel :)
void canvasGreyscale(canvas $ x1, int r1, int c1) {
  int R;
  int G;
  int B;
  int avg;
  int i;
  canvas $ p1;
  canvas $ p2;
  p1 = x1;
  p2 = x1;

  for (i = 0; i < (r1 * c1); i = i + 1) {
    R = ~p1;
    p1 = @1@ p1;
    G = ~p1;
    p1 = @1@ p1;
    B = ~p1;

    avg = (R + G + B) / 3;

    ~p2 = avg;
    p2 = @1@ p2;
    ~p2 = avg;
    p2 = @1@ p2;
    ~p2 = avg;

    p1 = @1@ p1;
    p2 = @1@ p2;
  }
}

:( Draw Rectangle
  Takes in canvas pointer, row number, column number,
  starting row coordinate i, starting col coordinate j with 0,0 starting at top left,
  length of rectangle, height of rectangle
  Changes canvas to draw black rectangle over corresponding pixel :)
void drawRectangle(canvas $ x1, int r1, int c1, int i, int j, int l, int h) {
  int a;
  int b;
  int c;
```

```
canvas $ p1;
canvas $ q1;
canvas $ s1;
p1 = x1;

if(l > c1) {
  print_string("Length of shape larger than canvas column length. Canvas not changed.");
}
else if(h > r1) {
  print_string("Height of shape larger than canvas row length. Canvas not changed.");
}
else if(i > r1) {
  print_string("Starting row coordinate of shape out of canvas bounds. Canvas not changed.");
}
else if(j > c1) {
  print_string("Starting col coordinate of shape out of canvas bounds. Canvas not changed.");
}
else if(i + h > r1) {
  print_string("Rectangle height will go out of bounds. Canvas not changed.");
}
else if(j + l > c1) {
  print_string("Rectangle width will go out of bounds. Canvas not changed.");
}
else {
  s1 = setPointer(p1, r1, c1, i, j);
  p1 = s1;

  :( Moves left to right, drawing black on pixels :)
  for(a = 0; a < l; a = a + 1) {
    for(b = 0; b < 3; b = b + 1) {
      ~p1 = 0;
      p1 = @1@ p1;
    }
  }

  :( Moves top to bottom, drawing black on pixels :)
  for(a = 0; a < h; a = a + 1) {
    q1 = p1;
    ~p1 = 0;
    q1 = @1@ q1;
    ~q1 = 0;
    q1 = @1@ q1;
    ~q1 = 0;
    for(b = 0; b < c1; b = b + 1) {
      for(c = 0; c < 3; c = c + 1) {
        p1 = @1@ p1;
      }
    }
    }

    :( Moves right to left, drawing black on pixels :)
    for(a = 0; a < l; a = a + 1) {
      for(b = 0; b < 3; b = b + 1) {
        ~p1 = 0;
        p1 = @-1@ p1;
      }
    }

    :( Moves bottom to top, drawing black on pixels :)
    for(a = 0; a < h; a = a + 1) {
      q1 = p1;
      ~p1 = 0;
      q1 = @1@ q1;
      ~q1 = 0;
      q1 = @1@ q1;
      ~q1 = 0;
      for(b = 0; b < c1; b = b + 1) {
        for(c = 0; c < 3; c = c + 1) {
          p1 = @-1@ p1;
        }
      }
    }
  }
}

:( Invert
  Takes in a canvas pointer and the row number and column number
  Changes canvas color values to inverted value of each RGB pixel :)
void canvasInvert(canvas $ x1, int r1, int c1) {
  int i;
  canvas $ p1;
  p1 = x1;

  for (i = 0; i < r1 * c1 * 3; i = i + 1) {
    ~p1 = 255 - ~p1;
    p1 = @1@ p1;
  }
}

:( Set Pointer
  Takes in a canvas pointer and the row number and column number of canvas,
  index i and j for coordinate to set pointer at
  Moves pointer to desired destination and returns :)
canvas $ setPointer(canvas $ x1, int r1, int c1, int i, int j) {
  int a;
  int b;
  int c;
  canvas $ p1;

  p1 = x1;

  :( Moves pointer to correct column :)
  for(a = 0; a < j; a = a + 1) {
    for(b = 0; b < 3; b = b + 1) {
      p1 = @1@ p1;
    }
  }

  :( Moves pointer to correct row :)
  for(a = 0; a < i; a = a + 1) {
    for(b = 0; b < c1; b = b + 1) {
      for(c = 0; c < 3; c = c + 1) {
        p1 = @1@ p1;
      }
    }
  }

  return p1;
}

:( Draw 3 Filled Rectangles
  Takes in canvas pointer, row number, column number,
  starting row coordinate i, starting col coordinate j with 0,0 starting at top left,
  length of rectangle, height of rectangle
  Changes canvas to draw three black rectangles over corresponding pixels :)
void fillThreeRectangles(canvas $ x1, int r1, int c1, int i, int j, int l, int h) {
  int a;
  int b;
  int c;
  int d;

  canvas $ p1;
  canvas $ q1;
  canvas $ z1;
  p1 = x1;

  if(l > c1) {
    print_string("Length of shape larger than canvas column length. Canvas not changed.");
  }
  else if(h > r1) {
    print_string("Height of shape larger than canvas row length. Canvas not changed.");
  }
  else if(i > r1) {
    print_string("Starting x coordinate of shape out of canvas bounds. Canvas not changed.");
  }
  else if(j > c1) {
    print_string("Starting y coordinate of shape out of canvas bounds. Canvas not changed.");
  }
  else if(i + h > r1) {
    print_string("Rectangle height will go out of bounds. Canvas not changed.");
  }
  else if(j + l > c1) {
    print_string("Rectangle width will go out of bounds. Canvas not changed.");
  }
  else {
    for(a = 0; a < j; a = a + 1) {
      for(b = 0; b < 3; b = b + 1) {
        p1 = @1@ p1;
      }
    }

    for(a = 0; a < i; a = a + 1) {
      for(b = 0; b < c1; b = b + 1) {
        for(c = 0; c < 3; c = c + 1) {
          p1 = @1@ p1;
        }
      }
    }

    z1 = p1;
    for (c = 0; c < h; c = c + 1) {
      for(a = 0; a < l; a = a + 1) {
        for(b = 0; b < 3; b = b + 1) {
          ~p1 = 0;
          p1 = @1@ p1;
        }
      }
      d = c1*c;
      print(d);
      p1 = @d@ z1;
    }
  }
}

:( Draw Circle
  Takes in canvas pointer, row number, column number,
  centered at row coordinate x0, starting col coordinate y0 with 0,0 starting at top left,
  radius of circle
  Changes canvas to draw black circle over corresponding pixel,
  Based on Midpoint circle algorithm from Wikipedia :)
void drawCircle(canvas $ x1, int r1, int c1, int x0, int y0, int radius) {
  int x;
  int y;
  int err;
  canvas $ p1;
  canvas $ q1;

  p1 = x1;
  q1 = p1;
  x = radius;
  y = 0;
  err = 0;

  while (x >= y) {
    p1 = setPointer(q1, r1, c1, x0 + x, y0 + y);
    ~p1 = 0;
    p1 = @1@ p1;
    ~p1 = 0;
    p1 = @1@ p1;
    ~p1 = 0;

    p1 = setPointer(q1, r1, c1, x0 + y, y0 + x);
    ~p1 = 0;
    p1 = @1@ p1;
    ~p1 = 0;
    p1 = @1@ p1;
    ~p1 = 0;

    p1 = setPointer(q1, r1, c1, x0 - y, y0 + x);
    ~p1 = 0;
    p1 = @1@ p1;
    ~p1 = 0;
    p1 = @1@ p1;
    ~p1 = 0;

    p1 = setPointer(q1, r1, c1, x0 - x, y0 + y);
    ~p1 = 0;
    p1 = @1@ p1;
    ~p1 = 0;
    p1 = @1@ p1;
    ~p1 = 0;

    p1 = setPointer(q1, r1, c1, x0 - x, y0 - y);
    ~p1 = 0;
    p1 = @1@ p1;
    ~p1 = 0;
    p1 = @1@ p1;
    ~p1 = 0;

    p1 = setPointer(q1, r1, c1, x0 - y, y0 - x);
```

```
    ~p1 = 0;
    p1 = @1@ p1;
    ~p1 = 0;
    p1 = @1@ p1;
    ~p1 = 0;

    p1 = setPointer(q1, r1, c1, x0 + y, y0 - x);
    ~p1 = 0;
    p1 = @1@ p1;
    ~p1 = 0;
    p1 = @1@ p1;
    ~p1 = 0;

    p1 = setPointer(q1, r1, c1, x0 + x, y0 - y);
    ~p1 = 0;
    p1 = @1@ p1;
    ~p1 = 0;
    p1 = @1@ p1;
    ~p1 = 0;

    y = y + 1;
    if (err <= 0) {
      err = err + 2*y + 1;
    }
    if (err > 0) {
      x = x - 1;
      err = err - 2*x + 1;
    }
  }
}
```

Test_var_a.cry

```
int main()
{
  int a;
  a = 42;
  print(a);
  return 0;
}
```

Test_var_b.cry

```
int a;

void foo(int c)
{
  a = c + 42;
}
```

```
int main()
{
  foo(73);
  print(a);
  return 0;
}
```

Test_while_a.cry

```
int main()
{
  int i;
  i = 5;
  while (i > 0) {
    print(i);
    i = i - 1;
  }
  print(42);
  return 0;
}
```

Test_while_b.cry

```
int foo(int a)
{
  int j;
  j = 0;
  while (a > 0) {
    j = j + 2;
    a = a - 1;
  }
  return j;
}

int main()
{
  print(foo(7));
  return 0;
}
```

Test_writefile.cry

```
:(tests writefile ppm production :)

int main(){
        int i;
        int j;
        canvas [100,100] g;
        canvas $ p;

        for(i = 0; i <100; i = i + 1) {
```

```
		for(j = 0; j < 100; j = j + 1) {
			g[i,j] = [255, 0, 0];
		}
	}
	p = &g;
	writefile(p,100,100, "coolimage.ppm");
	return 0;
}
```