CS 4115 Project Proposal

**Dmitriy Gromov (dg2720), Feifei Zhong (fz2185),
Yilei Wang (yw2493), Xin Ye (xy2190), Joe Lee (jyl2157)**

Introduction

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

We plan to solve this problem by implementing a programming language (EZ-ASCII) for the purpose of creating and manipulating ASCII Art, a form of drawing pictures using only the characters defined by the ASCII character set. The goal of the language will be to provide simple mechanisms for dealing with ASCII images. It will allow users to create images by adding and manipulating each character individually or by working with an existing image which is loaded and converted to an ASCII format. The language will provide a way to make functions or modules, which will enable developers to build re-use libraries to create more interesting constructs and images using this language.

EZ-ASCII and ASCII art

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

For image manipulation, the user will have the complete freedom to change any character in that image or change all characters of a given intensity to a different one. This way, an image's color can be "inverted" or simply darkened to conform to the user's tastes. Furthermore, the user will have the ability to apply simple transformations to a range of characters (e.g. shift several cells up and to the left).

One of the main uses for this programing language will naturally be to convert images from other formats into an ASCII format. However, using the various abilities available to the programmer, he/she should quickly be able to set up functionality to draw various shapes and constructs and manually create images.

Language Details

Basic Types Supported:

- Boolean – A true or false value
- Int – Some 32 bit integer
- Char – A character in the ASCII table

Complex Types:

- Array 1D, 2D – 1 or 2 dimensional collection of any type.
- Struct – A grouping of instances of other types to be treated as one type.
- Function references – A reference to a function that can be used as a type.

Branching Statements:

If, ElseIf, Else

Each statement will have a block of code associated with it. "If" will test some condition (in the form of a Boolean) and execute its block of code if the condition is true. If the condition is false, it will continue to the next ElseIf condition, evaluate it, and execute its block if the condition is true, or continue on to the next ElseIf condition. Finally, if the execution reaches the Else statement, the code there will be executed automatically. If there are no ElseIf statements and there is an Else, the Else block will be executed (if the If condition is not met). ElseIf and Else cannot exist without an If.

Loops:

For

This is a loop with a code block, an associated condition, and the condition variable. In this statement, the user will be able to specify the initial state of the condition variable, and how it should be updated at the end of each execution of the loop. The loop will continue to execute the code block until the specified condition is met.

Functions:

The user will be able to create functions that will act as blocks of code that can be called when desired. They will accept a list of input parameters and return some value at the end of their execution. It will be possible to specify no input parameters or return nothing if this is desired. Recursive functions will be allowed. Functions can also be passed as parameters to other functions.

Variables:

A variable is a named instance of some type in this language. Its scope of existence is limited to either the global level where all code in a program can see it or the function level where it is limited to only the function in which it is defined. Names of variables in the same scope must be unique. A variable in a function can have the same name as a global variable but referring to it from inside of the function will access the variable as defined in the function and not as defined globally.

Operators:

- Basic Arithmetic Operators: +, -, *, /, %
- Equality ==, ~=
- Assignment =

Program Linking and Execution:

- **Import keyword**: Will allow you to specify another source file, which will be appended to the current one at linking time.
- If defined, programs will start at a function called **main**. There can be at most one main defined in the program. If main is not defined the program will execute beginning to end as the code is written.

Language functionality:

Reading in an image File:

OCaml image libraries will read in an image, convert it to black and white, and compress the image if deemed too big. We will then apply Floyd–Steinberg dithering to the created image to effectively normalize the points we have and see which ones we can discard. Then, for each character-sized block of the image, we will determine the intensity of the pixels and create a character to represent that. These characters will be stored in a 2D array and returned to the user for manipulation.

Sample program:

Simplest way of generating ASCII art from an image file

```
# open image, convert to grayscale, compress and normalize, and return
# an array of chars (charmap optional)
char[][] img = load("/users/bob/hello_world.png", charmap)

# do fun manipulation with the img ...

# print array
print(img)
```

Mapping a function to a subset of elements

Let the asterisks in the ASCII image above mark the borders of the image. Say the ASCII image is stored in an array, p_img, and that we want to move the entire rectangle to the right by 4 characters (without worrying about anything inside the rectangle).

```
# return true if c is an underscore or pipe character
bool is_line(char c) {
    if c == '_' || c == '|' {
        return True
    }
    else {
        return False
    }
}

# return a new 2D array with character at point p "moved"
# to the right by 4 spaces
char[][] incr_x_4(char[][] img, point p) {
    img[p.row + 4][p.col + 4] = img[p.row][p.col]
    img[p.row][p.col] = EMPTY # white space
    return img
}

int main() {
    char[][] p_img = load("/users/bob/hello_world.png")

    # move the rectangle to the right by 4 characters
    # map takes an image, a function to map, and a conditional function
    p_img = map(p_img, incr_x_4, is_line)

    print(p_img)
}
```