Name: Fahad Shaikh

Final Year M.Tech. at IIT Kanpur.

Contents

Language Features and Description:

- I have decided to call the language FlipBook Language (FBL). Couldn't come up with something creative. In this sections I will describe the language and give an overview of its features. More detailed explanation will be in next sections.
- FBL is basically an commands based language, where the developer specifies what commands are to be executed on the variables. So let's start by listing the datatypes supported in the language:
    - Primitive Types: The language supports only 2 primitive data types. A number and a string (I am considering string to be primitive). A number can be used to represent a decimal or an integer. Some operations require integers in which case automatic truncation is done.
    - Complex Types: There are 3 complex types supported by the language. First is an Image which stores details related to an image like path, position in frame, size etc. Second is a Frame which represents sort of a canvas on which user can draw/position images. This frame will be translated to a page or frame in the resulting pdf or gif respectively. Third is a Pair which is just a pair of numbers used to represent position, size of image etc. internally. As of now user cannot create an variable of type pair but it can easily be provided.
- Now that we have defined the data types lets define what are commands. Commands are basically operations which have to be performed on the variables. A typical command always begins with a Keyword followed by a colon (:), and then a comma separated list of parameters used by a specific command. This is more like assembly style structure. I chose this kind of structure for the language since it seems more intuitive as the user will be carrying out operations on images for most part of the code. We will go into details of commands later.
- Keywords are case insensitive in the language however identifiers are case sensitive.
- Syntax of an identifier is similar to other languages. An identifier begins with a alphabet and the can have any alphanumeric characters. Note that underscores are not allowed.
- Control Flow Instructions: The language as of now supports just one control flow instruction i.e. a Loop. However it can be easily extended to include conditionals etc.
- Operators : For now the language supports just one operator (+). However it can easily be extended to support other operators.
- This is rather a loosely typed language as far as primitive types are concerned. A variable must be declared before it is used. One exception to this rule is a loop variable which will be declared if not already done. Once declared a variable can be assigned any of the primitive types data.
- In a typical program, a user will begin by loading a few images, applying image manipulation operations on them and then adding them to frames at different positions. Finally this frames will be added to the flipbook. In the end the flipbook will be saved in desired format.
- Each program can generate only one flipbook for now but this can be extended as well.
- In the next section I will take an example code and explain its working.

A Typical Program:

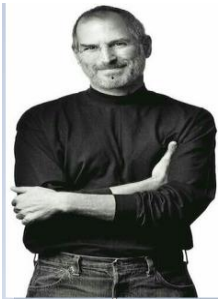
- In this section I will be explaining how the flow of a typical program goes by taking the example of falling_apple.flip. This code is included in the repository and produces falling_apple.gif/falling_apple.pdf.
- Lets break it down line by line.
- A program starts by defining a top level block. Beginning of a block is marked by Begin keyword.
- Then the user would declare their primitive type variables using the VAR command and assign them values using Assign command. Note that these commands can be used only for primitive types.

```
Begin:
    Var: start
    Var : inc
    Assign : start, 100
    Assign : inc, 40
```

- Then the user would load a bunch of images using CREATEIMAGE command and by specifying an identifier and a path where the image is located. After this user can perform image manipulations by using image manipulation commands like RESIZEIMAGE. For now the language has just one image manipulation command.

```
CreateImage: tree, "Tree_edited.jpg"
CreateImage: apple, "apple_new.jpg"
CreateImage: steve, "steve_jobs.jpg"
ResizeImage: apple, ABS, 50, 50
ResizeImage: tree, ABS, 640, 480
ResizeImage: steve, ABS, 75 , 100
```

- Images loaded by this are as follows:


- User can create a Frame using CREATEFRAME command.
- Once user has done the image processing he/she can add the image to a frame using ADDIMAGETOFRAME command which also takes the position of the image. Positions are specified using x and y coordinates w.r.t top left corner.

```
CreateFrame: frame1
AddImageToFrame: tree, frame1, 0,0
AddImageToFrame: steve, frame1, 300,380
AddFrame: frame1
```

- frame1 has the following content:

- User can create an initial frame which just has the background which will stay constant in more than one frames. Then he can us COPYFRAME command to copy this background in different frames. This can save a lot of time.

```
CreateFrame : frame2
CopyFrame: frame1, frame2
```

- Once the frame is ready it can be added to the flipbook using ADDFRAME command.
- Finally user can use LOOP command to stimulate the moving objects by adding these objects to the frames again using ADDIMAGETOFRAME command.

```
LOOP: st, 1, 8:
BEGIN:
    CreateFrame : frame2
    CopyFrame: frame1, frame2
    AddImageToFrame: apple, frame2, 300,start
    AddFrame: frame2
    Assign: start , start + inc
END:
```

- Once all frames are **added** user can call GENERATEFLIP command which will do all the processing and save the flipbook in desired format.
- Finally the programs ends with END command

```
    GENERATEFLIP:
END:
```

Compiler Options:

- I have provided the following compiler options:
    - Output file : -o, followed by a path where the output file must be saved.
    - Output Type: -t, followed by a string pdf/gif the type of output to be produced.
    - Duration: -d, time lapse duration between frames in a gif
    - Loop : -l , whether looping is enabled in a gif
    - Resolution : -r, resolution to be used for producing pdf.

Commands:

Each command is followed by a COLON (:) and then by a bunch of arguments depending on the command.

- **Begin**: - Defines the beginning of the block. Right now has just 2 use cases.
    - Specifies start of program
    - Specifies start of loop body
- **End**: - Denotes end of block
- **Var**: <IDENTIFIER> - Declares a identifier. Identifier can be used to store string or number. If a identifier is declared with same name it gives an error. Write now there is no initialization with declaration. However this can easily be added.
- **Assign**: <IDENTIFIER>, <EXPRESSION> - basically an assignment operator. Evaluates the expression and assigns the result to IDENTIFER. Note that identifier must be a VAR and not an Image or Frame
- **CreateImage**: <IDENTIFER> , <STRING> - Creates a Image object and assigns the given string as path for the image. Note that this command will not load the image but will check if the image exists.
- **ResizeImage**: <IDENTIFIER>, <MULT/ABS> , <EXPRESSION>, <EXPRESSION> - Basic image manipulation. Resizes the image. Third and Fourth parameters specify width and height respectively. The second parameter can be MULT or ABS which are both keywords in the language. If it is MULT the current width and height of image will be multiplied by 3rd and 4th arguments. If it is ABS width and height of image will be set to 3rd and 4th arguments.
- **CreateFrame**: <IDENTIFIER> - Creates a Frame object and denotes it by given identifier. Frame will be initially blank. Default background color is white. However this can easily be made customizable.
- **CopyFrame**: <IDENTIFER>, <IDENTIFIER> - First argument is the source and second is the destination. Basically creates a deep copy of arg1 into arg2. Both the frames must have been created before calling this. This can be used to create a background image which will be common among multiple frames/pages. Basically the same background can be copied and only moving objects need to be redrawn.
- **AddFrame**: <IDENTIFIER> - Identifier must represent a frame. Adds the frame to flipbook. Currently, once added the frame cannot be removed.
- **AddImageToFrame**: <IDENTIFIER>, <IDENTIFIER>, <EXPRESSION>, <EXPRESSION> - First arg specifies image name and second specifies frame name. This command adds the given image to frame. 3rd and 4th arguments specify Position (x,y) w.r.t. top left corner of the frame. Note that this command just saves the instruction to add the image to frame. The image is actually copied into the frame later.
- **Loop**: <IDENTIFIER>, <EXPRESSION>, <EXPRESSION> - Basic loop. First argument is the loop variable if it's not declared till now it will be created otherwise it will overwrite the same

name identifier. 2nd and 3rd argument specify start and end values for the loop variable. The loop initializes the loop variable to **2nd arg** and increments it by **one** till it reaches **(3rd arg) - 1.** Loop Statement must be followed by **Begin:**.

- **GenerateFlip**: This command is called after all frames are added to the flipbook. This will iterate through all the frames perform the necessary actions and the save the flipbook in the given format. Note that all the operations on image like loading the image, resizing it, adding it to frame at given position etc. are executed when this command is encountered.

Other Utility Commands:

- **Print**: <EXPRESSION> : Can be used for debugging. Prints the value of given expression on the console.
- **SaveImage**: <IDENTIFIER>, <STRING> - Saves the image represented by the identifier at given path. Again mainly used to see the modified image.
- **GetImageSize**: <IDENTIFIER>, <IDENTIFIER>, <IDENTIFIER> - 1st argument must represent and image. Saves the width and height of image in variables represented by the 2nd and 3rd arguments. Can be used for image manipulation and positioning.

Operators:

- **+ (plus):** As expected will add the values of the 2 operands. If any of the operand is a string this will perform string concatenation i.e. even if other operand is a number it will be appended as if it was a string. If both operands are numbers addition will be performed.
- Frankly I did not get time to implement any other operators. Adding other arithmetic operators is trivial. I didn't see any use of adding Logical Operators size the language doesn't have conditional statement. They can be added if needed.

Grammar:

- LITERAL : EOF | NEWLINE | NUMBER | STRING | COLON | COMA
- EOF : '\0'
- NEWLINE : '\n'
- NUMBER : DIGITS DOT DIGITS
- DIGITS : DIGIT+
- IDENTIFIER : CHAR (CHAR|DIGIT)+
- STRING : " (CHAR)* "
- COLON : ':'
- COMA : ','
- DOT : '.'
- TERM : NUMBER | STRING | IDENTIFIER
- EXPRESSION : TERM OPERATOR TERM | TERM
- OPERATOR : '+'
- COMMANDS : KEYWORD : ((IDENTIFIER)* (KEYWORD)* (EXPRESSION)* (,)*)*

Keywords

- BEGIN
- VAR
- ASSIGN
- CREATEIMAGE
- CREATEFRAME
- COPYFRAME
- ADDFRAME
- RESIZEIMAGE
- ADDIMAGETOFRAME
- SAVEIMAGE
- PRINT
- GETIMAGESIZE
- GENERATEFLIP
- LOOP
- END

Classes and their Description:

- In this section I will talk about the classes created and give a brief description.
- First there is the model package defining complex data types : Image, Frame and Pair.
- Then there is the compilerTools package. It has following class:
    - Lexer: Lexer for our compiler. Most important method is nextToken() which would fetch the next token skipping whitespaces. This class also maintains the source code, cursor position current line number etc.
    - Parser: Parser for our compiler. Has methods to match rules in grammar. Most important method is commands() which parses the commands. This class has an object of Executor and Lexer class and maintains information like current token.
    - Token: Defines a token. Each token has a text and a kind.
    - TokenType: Enum to define types of tokens.
- Then we have the Execution package with following classes:
    - Executor: This takes care of execution of commands. It has an object of State and ImageProcessor class.
    - State: Maintains the runtime state of the compiler. Has information like symbol table, current nesting level etc.
- Then we have the ImageTools package:
    - ImageProcessor: This class is responsible to do all dealings with the actual images.

Features I would have liked to introduce:

- I would like to talk about features that I really wanted to add but couldn't because of less time.
    1. First is definitely a bit better implementation of loops. My implementation was a bit hasty given the short amount of time. Also the options for loops are limited. I would have definitely liked to allow the user to specify the increment and termination condition. Maybe add a while loop for more flexibility.
    2. More Flexibility with frames: For now the only operations user can do on frames is COPYFRAME and Add image to it. Also once the frame is added to flipbook user can't edit it. I would like add a few more options like remove a specific image from a frame, reposition an image in a frame etc. Given the structure of language these operations shouldn't be too hard to add.
    3. More image manipulation operations: I would like to add more image manipulation operations. Some easy ones to add would be rotation, applying filters etc. Also some complex operations like cropping, clipping etc. can be added.
    4. More operators: This was something I didn't just to save time. Adding more operators is trivial and I would have done that if I had a few extra hours.
    5. More customization: I would like to allow the user to have more customizations like specifying background color which creating a frame (currently it's white), specify size of frame (currently 640 x 480) etc. Again this is easy to add and I have already made provisions in the State class.

References:

- *http://web.eecs.utk.edu/~azh/blog/teenytinycompiler1.html*
- *https://pillow.readthedocs.io/en/stable/*