Build-A-Char Project Specification

Jess Hu, Will Olsen

Video Explanation: https://drive.google.com/file/d/1vASNTjQVFMkyLatQbp2yEF75SCaf3CXc/view?ts=657ea89a

0.1 Introduction

What problem does your language solve? Why does this problem need its own programming language?

The purpose of our character generator programming language is to allow anyone, regardless of their artistic ability, to generate a custom character that they can use as inspiration for an original character in whatever they may need a character for. Someone who may not be as artistically inclined may have trouble generating a character of their own completely from scratch. Or, maybe they want to visualize how certain pieces of clothing and accessories may look when paired. This language addresses both of those problems.

We believe that this program should have its own programming language because it accomplishes a specific task and makes that task more feasible for a wider audience of people, regardless of their computer savviness. The user of this programming language would not need to know how to code to generate a custom character of their own. This makes our programming language accessible to people that extends beyond those with an understanding of CS, specifically of F sharp.

0.2 Design Principles

Languages can solve problems in many ways. What are the guiding aesthetic or technical principles that underpin its design?

Aesthetically, our programming language is guided by simplicity, due to its straightforward input that follows a rigid BNF grammar. The formatting of the input is designed to be easily understood by the user, due to the simple list format of the input, and no two inputs can generate the same character. As for technical ideas, our programming language will hopefully generate a final PNG composed of stacked PNGS that the user can then download onto their own device and use however they please. Our programming language will print out usage info before asking for the user's input so that the user knows how to format the description of the character the want to generate. If the user inputs an incorrectly formatted prompt into the command line, our programming language will remind the user of the correct formatting and continue to ask the user for a description until they input a valid one.

0.3 Examples

Provide three example programs in your language that will eventually work. Unlike the previous text you wrote here, these examples should conform to your formal grammar. Explain exactly how each example will be executed (e.g., dotnet run "example-1.lang") and provide the expected output (e.g., 2).

1)

```
dotnet run

happy orange cat blue tshirt and green pants black slippers pink flower
```

EXECUTION: use ImageSharp library to composite the following PNG files in order: Orange_Cat.PNG, Green_Pants.PNG, Blue_Shirt.PNG, Black_Slippers.PNG, Pink_Flower.PNG, Happy.PNG

OUTCOME: PNG file containing a drawing of a happy orange cat wearing blue shirt and green pants, black slippers, and pink flower.

2)

```
dotnet run

sad white dog pink dress white blank purple cowboy boots blue scarf
```

EXECUTION: use ImageSharp library to composite the following PNG files in order: Brown_Bear.PNG, Pink_Dress.PNG, Purple_Cowboy_Boots.PNG, Blue_Scarf.PNG, Sad.PNG

OUTCOME: PNG file containing a drawing of a sad brown bear wearing pink dress, purple boots, and blue scarf

3)

```
dotnet run

neutral white bunny black suit black blank black sneakers yellow star
```

EXECUTION: use ImageSharp library to composite the following PNG files in order: White_Bunny.PNG, Black_Suit.PNG, Black_Sneakers.PNG, Yellow_Star.PNG, Neutral.PNG

OUTCOME: PNG file containing a drawing of a neutral white bunny wearing black suit, black sneakers, and yellow star

0.4 Language Concepts

What are the core concepts a user needs to understand in order to write programs? Think in terms of both "primitives" and "combining forms." What are the key ideas and how are they combined?

In order to use our programming language, the user must be familiar with the English language, or at least all of the words included in our BNF grammar, such as all of the emotions, animals, clothing, and colors. This is because in order to build any character, the user must be able to decide what features they want their character to have.

All of the primitives in our language are words taken from the English language and maintain the same meanings they have in English. These primatives include all of the words that follow ¡color¿ in our BNF grammar, such as bunny, suit, dress, heels, flower, and gloves, to name a few. Colors are also primatives in our language. As for combining forms, the ones in our language include evaluations for animal, top, bottom, shoes, and accessory, as they each combine the primitive color with another primitive to make a PNG file name. Each combining form makes a phrase, while each primitive is a single item. Then these filenames are loaded as Images in ImageSharp, composited using DrawImage, and saved.

0.5 Syntax

Provide a formal syntax your language, written in Backus-Naur form.

For input, the user's input would be of the form:

```
<emotion> <animal> <top> <bottom> <shoes> <accessory>
```

where each element can have a specific color, as defined by the BNF grammar.

```
<expr> ::= <emotion> <animal> <top> <bottom> <shoes> <accessory>

<emotion> ::= happy | neutral | sad | mad | tired

<animal> ::= <color> bunny
           | <color> cat
           | <color> dog
           | <color> bear

<top> ::= <color> shirt
        | <color> sweater
        | <color> hoodie
        | <color> suit
        | <color> dress
        | <color> blank

<bottom> ::= <color> pants
           | <color> shorts
           | <color> skirt
           | <color> blank

<shoes> ::= <color> sneakers
          | <color> cowboy boots
          | <color> combat boots
          | <color> heels
          | <color> slippers
          | <color> blank

<accessory> ::= <color> flower
              | <color> star
              | <color> glasses
              | <color> sunglasses
              | <color> scarf
              | <color> gloves
              | <color> blank

<color> ::= red | orange | yellow | green | blue | purple | pink |
            black | white | brown
```

0.6 Semantics

i. What are the primitive kinds of values in your system? For example, a primitive might be a number, a string, a shape, a sound, and so on. Every primitive should be an idea that a user can explicitly state in a program written in your language.

The primitives in our language are strings that are all words in the English language. These words specify emotions, animals, clothing, and colors. Each animal and article of clothing can be built with a specific color followed by an animal or piece of clothing in the English language that is included in our BNF grammar.

ii. What are the "actions" or compositional elements of your language? In other words, how are values combined? For example, your system might combine primitive "numbers" using an operation like "plus." Or perhaps a user can arrange primitive "notes" in a "sequence."

The primitive strings in our language are combined by arranging them in the sequence specified by the definition of "expr" in our BNF grammar. The emotion and animal is specified first, followed then by top, bottom, shoe, and accessory. Then these strings are concatenated to make strings representing PNG file names. Then they are loaded as images in ImageSharp and composited using the DrawImage method from ImageSharp. A new image is then saved, called "Character.PNG"

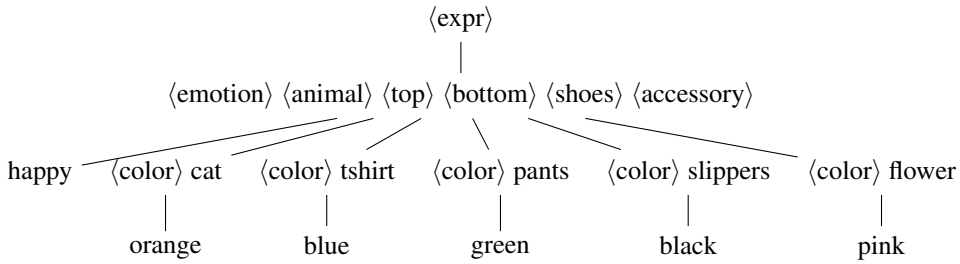
iii. How is your program represented? In other words, what components (types) will be used in your AST? If it helps you to think about this using ML algebraic data types, please use them. Otherwise, a rough sketch like a class hierarchy drawings or even Java class code is OK.

In our AST, the types include ⟨expr⟩, ⟨emotion⟩, ⟨animal⟩, ⟨top⟩, ⟨bottom⟩, ⟨shoes⟩, ⟨accessory⟩, and ⟨color⟩. As indicated in our BNF grammar, an expr is of the form ⟨emotion⟩ ⟨animal⟩ ⟨top⟩ ⟨bottom⟩ ⟨shoes⟩ ⟨accessory⟩. An ⟨emotion⟩ is either happy, neutral, sad, mad, or tired. An ⟨animal⟩ consists of a ⟨color⟩ followed by either bunny, cat, frog, or bear. A ⟨top⟩ consists of a ⟨color⟩ followed by suit, dress, hoodie, sweater, tshirt, or blank. Then, ⟨bottom⟩ is a ⟨color⟩ followed by pants, shorts, skirt, or blank. Then, ⟨shoes⟩ is a ⟨color⟩ followed by sneakers, cowboy boots, combat boots, heels, slippers, or blank. An ⟨accessory⟩ is a ⟨color⟩ followed by either flower, star, glasses, sunglasses, scarf, gloves, or blank. Finally a ⟨color⟩ is red, orange, yellow, green, blue, purple, pink, black, white, or brown.

iv. How do AST elements "fit together" to represent programs as abstract syntax? For the three example programs you gave earlier, provide sample abstract syntax trees.

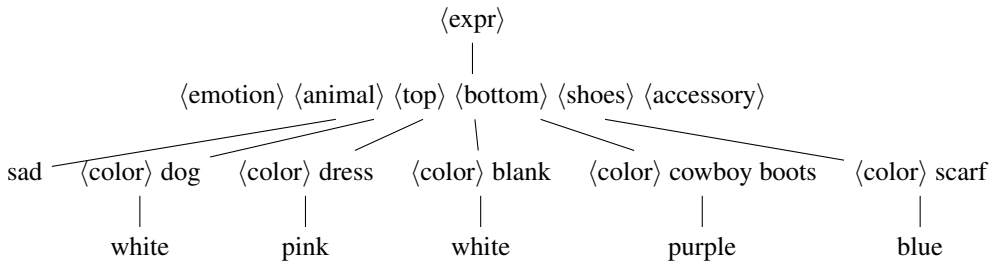
Note: There should be commas included in the expression, but the tree does not allow us to include the commas for some reason. Also, the rest of the tree should be branching out of the non-terminals in the expression, but we could not figure out how to build the tree that way even with TA help.
1)

```
happy orange cat blue tshirt green pants black slippers pink flower
```



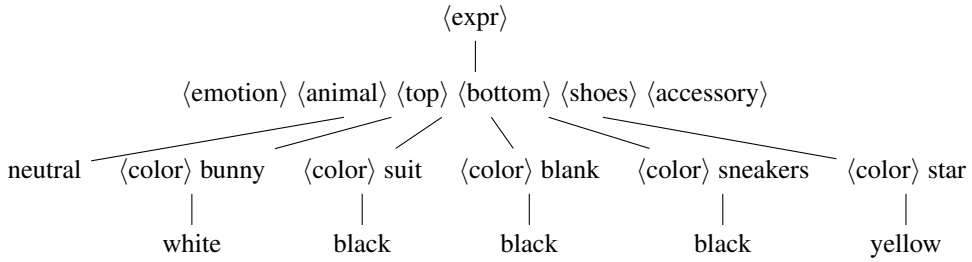
2)

```
sad white dog pink dress white blank purple cowboy boots blue scarf
```



3)

```
neutral white bunny black suit black blank black sneakers yellow star
```



Our language reads any input that is of the form ⟨emotion⟩ ⟨animal⟩ ⟨top⟩ ⟨bottom⟩ ⟨shoes⟩ ⟨accessory⟩. If the given input does not follow that format, the program print usage info and will continue asking for input until the user either enters a correctly formatted input or quits. The output of evaluating a program will be a PNG representing the generated character. This PNG will be built by stacking individual PNGs for each element needed to build the character. For example, the final PNG for the first AST will be composed of a happy face, an orange cat, a blue shirt, green pants, black slippers, and a pink flower. When these elements are stacked in this exact order, the final image will be a completed character. Since the tree is traversed in post-order, the expr will be evaluated only after each of its non-terminals has been evaluated. For types that are composed of a non-terminals, such as ⟨animal⟩, which is composed of a ⟨color⟩ and a word such as bear. In example 1, the colors orange, blue, green, black, and pink must be evaluated so that orange cat, blue shirt, green pants, black slippers, and pink flower can be evaluated. Only after all of these evaluations can the expr be evaluated because there are no non-terminals left to evaluate.

0.7 Remaining Work

In the future, we would love to implement ways for users to input their own PNGs to the library. We would also like to have our programming language be able to generate characters in different poses or with different backgrounds behind them.