COMPSCI 119
Extra Credit Lab #1
Bar Codes
Professor William T. Verts

Introduction and Background

In this assignment you are to create bar codes from user input. There are two common bar codes in use today, UPC-A and EAN-13, as well as many alternative codes that we need not consider here. The 12-digit Universal Product Code (UPC) has been in use on most commercial products in America since the 1970s. The 13-digit European Article Number (EAN) is a *superset* of UPC, which means that EAN-13 has the same configuration and layout as UPC-A and generates the same code as UPC in the case where the leading digit is 0. Since they are so closely allied, we will consider only the EAN-13 code here (that is, if you can generate EAN-13, you can also generate UPC-A for free). Here is a valid EAN-13 bar code shown below:

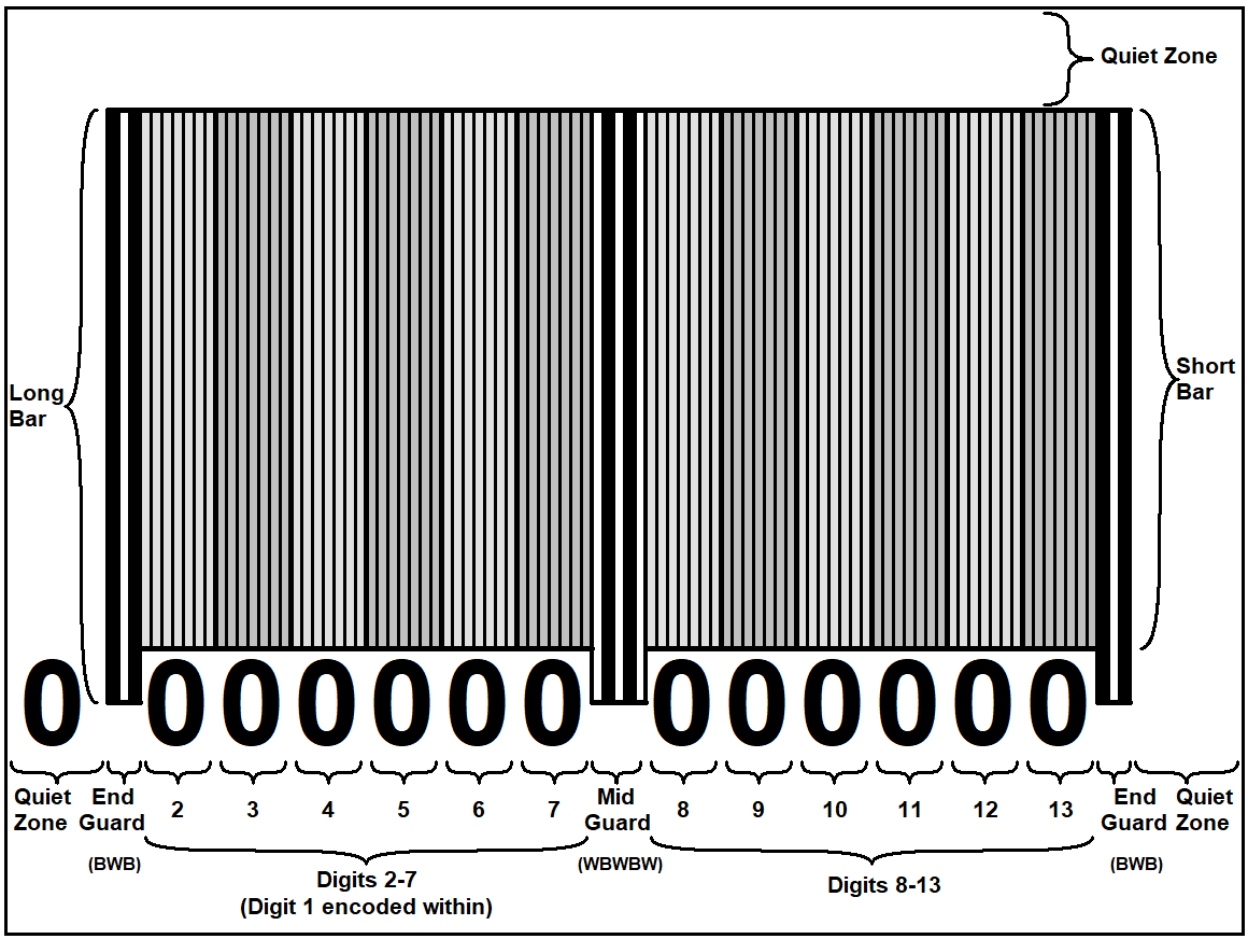
The purpose of your Python program is to let a user enter any number containing up to 13 digits, such as 9781524943998, and then draw out the bar code as shown here (without the black border and without the digit characters). If the number contains fewer than 13 digits, enough 0s will be appended to the left to make a total of 13.

You will be using the graphics library **`Graphics.py`** that you developed in an earlier assignment.

See pages 446-447 in the Computer Science Companion (4TH Edition) for another reference to UPC and EAN-13 codes.

Layout of EAN-13 Bar Codes

UPC-A and EAN-13 codes have exactly 95 vertical stripes: some long and some shorter, some black and some white. At both the left and right ends of the code are three-stripe guard bars that are always in the order black-white-black. In the middle is a five-stripe guard bar that is always in the order white-black-white-black-white. Between the left and middle guard bars are the stripes for six digits, each digit consisting of seven stripes each. Similarly, there are six more digits of seven stripes each between the middle and right guard bars. Thus, the width of the bar code as painted is 3 + (6×7) + 5 + (6×7) + 3 = 95 stripes. Around the outside of the complete bar code is an additional "quiet zone" that is at least 9 stripes wide; this is to ensure that a scanner isn't confused trying to find the start or end of the code.



There isn't a specific digit position for the leftmost of the 13 digits; that digit is encoded in a distributed fashion within the region for the left six digits (this is due to the UPC-A code being reverse-engineered to hold 13 digits instead of 12, and yet maintain backwards compatibility with UPC-A).

Setting Sizes

The bar code is thus 9{quiet zone} + 95{bar code} + 9{quiet zone} = **113 stripes wide**. There is a lot of flexibility on the height of a bar code, so we can start with the same height as the width. The bar code height will be **113 stripes tall**. At minimum, each stripe will be exactly one pixel wide. For this assignment you will enter a **scale factor** to determine the stripe width, in pixels. For example, if the scale factor is 3 then each stripe will be 3 pixels wide, and the canvas will be therefore 3×113=339 pixels wide by 339 pixels tall.

The length of a **long bar** is 95 stripes, and the length of a **short bar** is 90 stripes. These values must also be multiplied by the scale factor to get their corresponding lengths in pixels.

Selecting Digit Patterns

The process for figuring out the white and black stripes for each digit is a little complicated. In the Python lists **L_Code**, **G_Code**, and **R_Code** below, each list contains ten strings of seven **W** or **B** characters each, indicating a white or black stripe, respectively. These lists are used to paint the actual stripes onto the canvas. The **L_Group** and **R_Group** lists each contain ten strings of six **L**, **G**, or **R** characters, and determine whether the stripe codes of some digit value are to be pulled from **L_Code**, **G_Code**, or **R_Code**, respectively, based on the left-most pf the 13 digits. Type these five lists into your program at the top of the code. Notice the **** at the end of the first line of each list; this is to allow the lists to extend onto the next line.

```
L_Group    = ["LLLLLL",  "LLGLGG",  "LLGGLG",  "LLGGGL",  "LGLLGG",  \
              "LGGLLG",  "LGGGLL",  "LGLGLG",  "LGLGGL",  "LGGLGL"]

R_Group    = ["RRRRRR",  "RRRRRR",  "RRRRRR",  "RRRRRR",  "RRRRRR",  \
              "RRRRRR",  "RRRRRR",  "RRRRRR",  "RRRRRR",  "RRRRRR"]

L_Code     = ["WWWBBWB", "WWBBWWB", "WWBWWBB", "WBBBBWB", "WBWWWBB", \
              "WBBWWWB", "WBWBBBB", "WBBBWBB", "WBBWBBB", "WWWBWBB"]

G_Code     = ["WBWWBBB", "WBBWWBB", "WWBBWBB", "WBWWWWB", "WWBBBWB", \
              "WBBBWWB", "WWWWBWB", "WWBWWWB", "WWWBWWB", "WWBWBBB"]

R_Code     = ["BBBWWBW", "BBWWBBW", "BBWBBWW", "BWWWWBW", "BWBBBWW", \
              "BWWBBBW", "BWBWWWW", "BWWWBWW", "BWWBWWW", "BBBWBWW"]
```

If a particular bar code digit is 7, for example, and if we happen to know the code is to be pulled from **R_Code**, then **R_Code[7]** is **BWWWBWW**, indicating that the stripes to be painted for that digit are black-white-white-white-black-white-white.

An Example

So, how do we know if a particular digit-pattern is to be pulled from **L_Code**, **G_Code**, or **R_Code**? The answer depends on whether we are painting a digit in the left group of six or in the right group of six, and it also depends on the first (leftmost) digit. This information is encoded in the lists **L_Group** and **R_Group**.

Let's use as an example the EAN-13 code presented earlier: 9781524943998. We first divide it into three fields: 9-781524-943998.

The left most digit is 9, so we look up **L_Group[9]** and **R_Group[9]**. **L_Group[9]** is **"LGGLGL"** and **R_Group[9]** is **"RRRRRR"** (yes, they all are – that isn't a mistake). The left group of six digits, 781524, will use **LGGLGL**, and the right group of six digits, 943998, will use **RRRRRR** to figure out which black-and-white stripe codes to use. This is shown below:

For 781524: **L_Code[7]**, **G_Code[8]**, **G_Code[1]**, **L_Code[5]**, **G_Code[2]**, **L_Code[4]**

For 943998: **R_Code[9]**, **R_Code[4]**, **R_Code[3]**, **R_Code[9]**, **R_Code[9]**, **R_Code[8]**

Once this information is known, the bar code can be created from the following stripes:

Left guard bar	**BWB**	(long)
L_Code[7]	**WBBBWBB**	
G_Code[8]	**WWWBWWB**	
G_Code[1]	**WBBWWBB**	
L_Code[5]	**WBBWWWB**	
G_Code[2]	**WWBBWBB**	
L_Code[4]	**WBWWWBB**	
Mid guard bar	**WBWBW**	(long)
R_Code[9]	**BBBWBWW**	
R_Code[4]	**BWBBBWW**	
R_Code[3]	**BWWWWBW**	
R_Code[9]	**BBBWBWW**	
R_Code[9]	**BBBWBWW**	
R_Code[8]	**BWWBWWW**	
Right guard bar	**BWB**	(long)

Each black stripe can be painted using some number of calls to the **addLine** function or the **VerticalLine** function, based on the scale factor. That is, if the scale factor is 3, then each black stripe will be three calls to **addLine** or **VerticalLine** with successive X coordinates.

More Details on the Process

0. Put your name in a comment at the top of the program.

1. Ask the user for a 13-character string, all digits. If the string is shorter than 13 characters, pad it on the left with 0 characters to make it exactly 13 digits. Convert the string into a list of 13 integers, where each integer is in the range [0…9].

2. Ask the user for a scale (1-10). This represents the width of a stripe, in pixels (that is, at **`Scale`**=1 a stripe is one pixel wide; at **`Scale`**=5 a stripe is five pixels wide).

3. Determine the width and height of the canvas, based on the number of needed stripes and the **`Scale`** value. Similarly, determine the length of the long and short lines. Create the canvas.

4. Based on the first of the 13 digits, determine from **`L_Group`** and **`R_Group`** the code patterns to use.

5. Paint the left guard pattern with long lines.

6. Paint digits 2-7 as a series of black and white stripes with short lines, with the patterns picked from **`L_Code`** and **`G_Code`**.

7. Paint the middle guard pattern with long lines.

8. Paint digits 8-13 as a series of black and white stripes with short lines, with the patterns picked from **`R_Code`**.

9. Paint the right guard pattern with long lines.

10. Save the image as **`BarCode.bmp`** on your disk.

When your program is complete, save it as **`BarCode.py`** on your disk. Try it with the code for the Companion, as well as other EAN-13 and UPC-A codes that you find. If you want to try a 12-digit UPC-A code, remember to prefix it with a 0 character (that is, a UPC-A code such as 123456789012 should be entered as EAN-13 code 0123456789012). Test your program with different scale factors as well.

Submit this program using the on-line form as **Lab #E1**. We will test your program by running it with different bar codes and different scale factors, and will grade as follows:

Scoring

A: -10 No student name in a comment at the top of the **`BarCode.py`** file.
B: -10 Program crashes before running to completion, either in **`BarCode.py`** or in **`Graphics.py`**.
C: -8 **`BarCode.py`** runs to completion, but **`BarCode.bmp`** is not created.
D: -6 **`BarCode.py`** runs to completion, and **`BarCode.bmp`** is created, but contains little if anything that is recognizable.
E: -4 **`BarCode.py`** runs to completion and **`BarCode.bmp`** is created, but the bars are not in the right places, have the wrong lengths or values, or do not correctly scale to arbitrary sizes.