フォントで FizzBuzz する方法（1）:
OpenType 機能

にせねこ（@nixeneko）
サークル“ヒュアリニオス”

2017年12月31日

目次

第1章

はじめに

1.1 FizzBuzz 問題とは何か

プログラマが基本的なプログラミング能力を持っているか判定するためのものとして、FizzBuzz 問題がある。もしご存知であれば本節を読み飛ばしてかまわない。

Fizz Buzz は、英語圏で長距離ドライブ中や飲み会の時に行われる言葉遊びである。何人かの参加者が順番に数字をカウントアップして言っていくが、3の倍数の場合は“Fizz”、5の倍数の場合は“Buzz”、3と5の倍数（つまり15の倍数）のときは“Fizz Buzz”と言わないといけないルールになっている。この時、ゲームの発言は次のように進行する。

```
1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14,
FizzBuzz, 16, 17, Fizz, 19, Buzz, Fizz, 22, 23, Fizz, Buzz, 26,
Fizz, 28, 29, FizzBuzz, 31, 32, Fizz, 34, Buzz, Fizz, 37, 38,
Fizz, Buzz, 41, Fizz, 43, 44, FizzBuzz, 46, 47, Fizz, 49, Buzz, …
```

Jeff Atwood は、プログラマ志望者に対してこのゲームを出力するプログラムを作成させることによって、コードが書けない志願者を判別する手法を提唱した。これを FizzBuzz 問題という。プログラムの出力は、ちょうど上掲したような文字列となる。

例えば、本誌は LaTeX（さらにいえばXeLaTeX）で書かれているが、LaTeX で利用

している TEX 言語[*1]では FizzBuzz 問題は次のように書ける（コードは読まなくてよい）。

```
\documentclass[a4paper]{article}

\def\checkfizzbuzz#1{%
  \newcount\n \newcount\m \newcount\modthree \newcount\modfive%
  \n=#1 \m=#1 \modthree=#1 \modfive=#1%
  \divide \n by 3%
  \multiply \n by 3%
  \advance \modthree by -\n%
  \divide \m by 5%
  \multiply \m by 5%
  \advance \modfive by -\m%
  \ifnum\modthree=0 {Fizz}\fi%
  \ifnum\modfive=0 {Buzz}\fi%
  \multiply\t by \f%
  \ifnum\t>0 #1\fi%
}

\def\fizzbuzz#1{%
  \newcount\cnt \newcount\last%
  \last=#1%
  \advance \last by -1%
  \cnt=0%
  \loop\ifnum\cnt<\last%
    \advance\cnt by 1%
    \checkfizzbuzz{\the\cnt}, %
  \repeat%
  \advance\cnt by 1%
  \checkfizzbuzz{\the\cnt}%
}

\begin{document}
\fizzbuzz{100}
\end{document}
```

本誌では、このような FizzBuzz 問題を、フォントの機能を使って実装することを試

[*1] “TEX”は「テック」、“LATEX”は「ラテック」と読む。それぞれ「テフ」「ラテフ」と読む流儀もある。

みる。なぜそんなことをやるのか？　できそうだからである。

FizzBuzz を実装すると言っても、フォントプログラムの出力は限られていて、文字列を入力としてそれに対応する文字図形を表示することしかできない。そのため、今回は、

```
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, …
```

というような数字をカンマで区切った文字列が与えられた場合に、3や5の倍数の部分を適切に Fizz, Buzz, FizzBuzz で置き換え、次のように表示するようなフォントをつくるということにする。

```
1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14,
FizzBuzz, …
```

1.2 FontForge

本誌では、フォントを編集するために、FontForge[*2]というオープンソースのフォントエディタを利用する。Windows, Mac, Linux で動作する。

インストール方法については特に解説しないが、Ubuntu[*3]ではリポジトリに入っているので、`fontforge` パッケージを `apt` コマンドなどでインストールすればよい。他のディストリでもパッケージマネージャのリポジトリに入っている場合が多いと思うが、最新版が使いたければ自分でコンパイルする必要がある。

また、Windows であればインストーラが配布されているのでそれを利用すると簡単に導入できる。Mac は環境がないのでわからない。

本誌では、次章以降、Ubuntu 16.04 LTS にインストールしたものを基準にして解説していく。

[*2] `http://fontforge.github.io/`

[*3] Linux の一種。`https://www.ubuntu.com/`

第2章

OpenType 機能: GSUB

2.1　OpenType とは

本誌で扱うフォントは OpenType 規格のものを指す。OpenType とはフォントの形式で、Adobe と Microsoft が共同で策定したものである。この規格は、Apple と Microsoft が共同で策定した TrueType 規格と、Adobe による PostScript フォントの流れを引く CFF (Compact Font Format) 規格を融合させたもの[*4]である。OpenType 規格のフォントは図形のアウトラインデータ表現として TrueType 形式か CFF 形式かのどちらかを利用している。

今日において TrueType フォント／ OpenType フォントという場合、ふつうそれぞれ図形の表現として TrueType 形式／ CFF 形式を利用した OpenType フォントのことを指す。要するに、現在一般的なパソコンやスマホで使われているフォントは OpenType 規格であるといえる。

[*4] バイナリの仕様をみると、TrueType の枠組みに CFF をほとんど丸ごと突っ込めるようにした形になっていて、ある人は「キメラフォーマット」と形容していた。

2.2 高度な組版機能

フォントはコンピュータでテキストを表示する際に使われるもので、文字コードで表現された文字列を対応する文字の図形に置き換えるという機能をもつ。文字の図形を**グリフ**とよぶ。この符号とグリフとの対応は一対一であるとは限らない。アルファベットや日本語ではあまり出番がないが、例えば、アルファベットの“fi”, “ffi”などは(“fi”や“ffi”でなく)その文字の組合せ専用の1つのグリフ(**合字**あるいは**リガチャ (ligature)** とよぶ)を用いて表現されることがある。

他にも、アラビア文字は右から左に書き、文字が続け書きされ、同じ文字でも前後に繋がるかどうかによって形が変わるようになっており、例えば ʿayn という文字は、左右どちらにも繋がらない形が ع 、左にのみ繋がる形が ـع 、左右両方に繋がる形が ـعـ 、右にのみ繋がる形が عـ となっている。このように、単に文字を左から右に並べただけでは適切な表示にはならない文字表記を**複雑な用字系**(complex script)という。

以上のように、美的により良い形を選んだり、複雑な用字系に対応するための機能を「高度な組版機能」などといったりする。これを実現したのが OpenType 規格における **OpenType 機能**(OpenType features)である。

2.3 OpenType 機能とは

OpenType 機能とは複雑な用字系に対応したり高度な組版を行うための仕組みであることは前述した。

これは、細かく分けると、GSUB(グリフの置換)、GPOS(グリフ位置の調整)、GDEF(グリフのプロパティの定義)の3種類からなる。

ここで、プログラミングっぽいことをするにあたって重要なのは GSUB である。グリフの置換を行うことで、状態が表現できるようになる。また、文脈[*5]に依存して、前

[*5] ここでは、「グリフの並び」といった意味合い。

後に特定のグリフが並んだ時だけ置換するようにできるため、要するに条件分岐が表現できる。

2.4 GSUB ことはじめ

GSUB の設定は、OpenType 機能の設定をテキストで記述することができる OpenType feature file[*6]を使うとやりやすい。これは、FontForge からも読み込むことができるほか、Glyphs[*7]などの商用のソフトでも同様の書式で GSUB の設定を行うことができる。

ここから、アイコンフォントの例を見ながら、OpenType feature file の書き方について見ていこう。

2.4.1 ケーススタディ: アイコン Web フォント

Web サイトで、アイコンを Web フォントで表現することがある。アイコン表示用に普通の文章では使わないコードポイントにアイコンを割り当てるということもできるが、その場合だと Web フォントが読み込まれなかった場合、アイコン部分が化けてしまって何もわからないことになる。また、音声読み上げ機能等を使っている場合にもそこに何が書かれているのか正しく認識することができない。

ここで役立つのが、複数の文字に対して一つの図形を割り当てるリガチャである。例えば、home という文字列を家の記号 🏠 に置き換えるような Web フォントを作成すると、フォントが読み込まれていれば 🏠 と表示され、読み込まれなかったとしても home と表示される。そのため、フォントがうまく表示されなかった場合でも何が書かれているかを把握できるため、独自で割り当てたコードポイントを使う時の問題を回避することができる。

では、このようなアイコンフォントを実際に作ってみよう。まず、ベースとするフ

[*6] http://www.adobe.com/devnet/opentype/afdko/topic_feature_file_syntax.html

[*7] https://glyphsapp.com/

ォントを用意する。自分で作ってもいいが、面倒なのでここでは M+ フォント[*8]の `mplus-1p-regular.ttf` を利用することにする。

2.4.2 下準備

最初に `mplus-1p-regular.ttf` を FontForge 開く。

ここで、見通しをよくするため、非 ASCII のグリフを削除し、エンコーディングを Latin1にする。さらに M+ フォントで定義されている GSUB も消しておく。これを済ませたフォントファイル `mplusj-1p-regular_subset.ttf` をサポートページ[*9]からダウンロードできるようにしているので参考にしてほしい。削除の具体的な手順は次のようになる。

1. 0〜126番目（先頭からチルダ“~”まで）のグリフをドラッグして選択する。
2. メニューの 編集 (E) 》 選択 (S) 》 選択範囲の反転 (I) を選ぶ。
3. エンコーディング (N) 》 グリフの切り離し・削除 (V)... を選択し、確認ダイアログの 削除 (R) ボタンを押す。
4. メニューの エンコーディング (N) 》 エンコーディング変換 (R) 》 ISO 8859-1 (Latin1) を選択する。
5. メニューの エレメント (L) 》 フォント情報 (F)... を選択しフォント情報ダイアログを開き、Lookups タブを選択する。
6. GSUB タブの全ての lookup を選択し（一番上の lookup 名をクリック、Shift キーを押しながら一番下の lookup 名をクリック）、Delete ボタンを押して全ての lookup を削除する。
7. GPOS タブを開いて同様に削除する。
8. OK をクリックしてダイアログを閉じる。

これ（図2.1）を元に実装していく。

[*8] `https://mplus-fonts.osdn.jp/`

[*9] `http://hyalinios.hatenadiary.com/entry/c93-gsub`

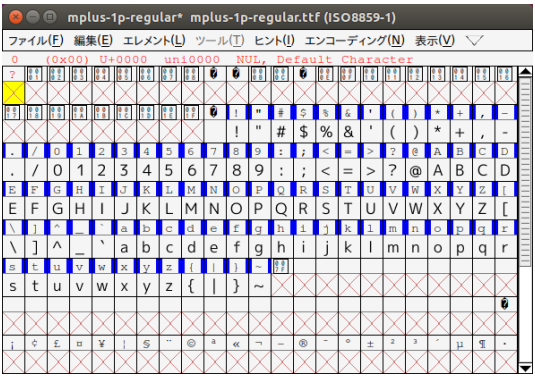


図 2.1. 用意した、たたき台となるフォント。

2.4.3 グリフの用意

さて、今回は home という文字列が家のマーク 🏠 へと置換されるようにしたいが、置換を定義する前にグリフを用意しなければならない。なので、home という名前でグリフを作成する。メニューの エンコーディング (N) 》 エンコーディングスロットを追加 (A)... を選択し、開いたダイアログで追加するスロット数に1を指定し、OK を押す。

そうすると、グリフ一覧の一番最後に NameMe.256 という名前のグリフが追加される。このグリフを右クリックし グリフ情報 (I)... を選択する。開いたダイアログの Glyph Name を home に変更し、OK を押す。

これで home という名前のグリフが作成された。このグリフをダブルクリックするとアウトラインウィンドウが開き、グリフのアウトラインが編集できる。今回は適当に家っぽい図形 🏠 を描いてみた（図2.2）。また、ここで ファイル (F) 》 取り込み (I)... から SVG 等をインポートすることもできる。

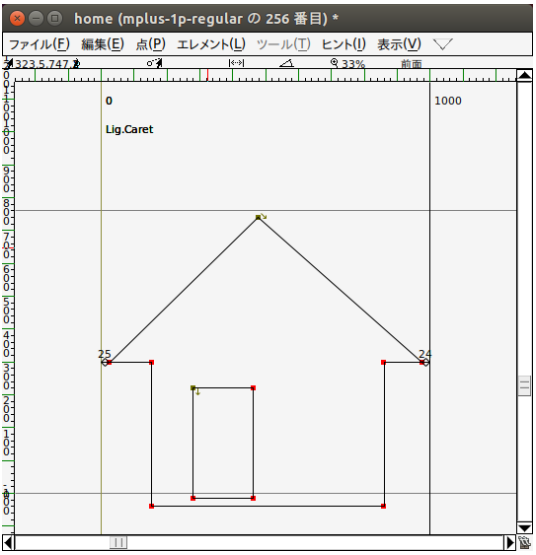


図 2.2. 家っぽい図形 🏠 を用意した。

2.4.4 GSUB の定義

遂に GSUB の定義にとりかかる。次のような内容のテキストファイルを作成する。名前は何でもよいが、ここでは `home.fea` とする。

リスト 2.1. `home.fea` ……グリフ列 `home` の並びを 🏠 に置き換える

```
1  # 用字系(script)と言語の範囲
2  languagesystem DFLT dflt; #デフォルト用字系・デフォルト言語
3  languagesystem latn dflt; #ラテン文字・デフォルト言語
4
5  # 合字の設定
6  feature liga {
7      substitute h o m e by home; #グリフ列homeを家の絵で置換
8  } liga;
```

コードの説明は後述するが、メニューの ファイル (F) 》 Merge Feature Info... からこ

の home.fea を取り込むと、GSUB の設定が適用される。設定が動いているかどうかは、メニューの ウィンドウ (W) 》 メトリックウィンドウを開く (M) から開けるメトリックウィンドウによって確認することができる。上の欄に home という文字列を入力すると、下に家っぽい図形 🏠 が表示されるのが確認できる(図2.3)。

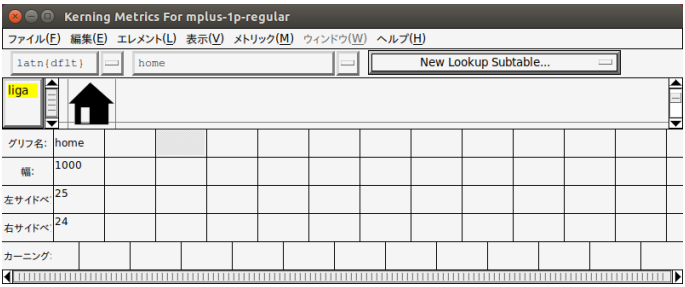


図 2.3. メトリックウィンドウで 🏠 のリガチャの動作が確認できる。

これを、ファイル (F) 》 フォントを出力 (G)... から出力することができる。出力形式に TrueType を指定し、適当な名前(ここでは home_liga.ttf)をつけて保存すると、フォントが出力される。

このフォントは他のソフトウェアから利用することもでき、例えばブラウザで Web フォントとして利用するには、CSS にリスト2.2のように設定し、HTML 文書内にリスト2.3のようにフォントを設定すると、ブラウザで開いたとき、そのフォントで表示した home という文字列が家の記号で表示される(図2.4)。

リスト 2.2. フォントを読み込みリガチャを有効にするスタイルシート

```
1 @font-face {
2  font-family: homeliga; /* フォント名を設定 */
3  src: url(./home_liga.ttf); /* フォントファイルの指定 */
4  font-feature-settings: "liga" 1; /* liga を有効に */
5 }
```

リスト 2.3. フォントを適用する HTML コード

```
1 Click <span style="font-family: homeliga;">home</span> to go to
      the home page.
```

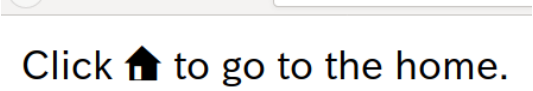


図 2.4. Firefox で家の記号のフォントを表示した場合

2.4.5 GSUB 定義コードの説明

さて、リスト2.1のコードの説明に移る。なお、# は行のそれ以降がコメントであることを表す。

まず、1、2行目にある `languagesystem` は置換等を実行する用字系（ラテン文字、キリル文字、仮名、漢字、など）と言語（英語、中国語、日本語、など）をそれぞれ4文字程度のタグで指定するものである。特殊なタグとして `DFLT`（用字系）と `dflt`（言語）をそれぞれ設定することができ、デフォルトの場合を定義できる。今回はラテン文字だけ使うので、デフォルト用字系 `DFLT` に加えてラテン文字 `latn` を設定し、言語についてはデフォルト言語 `dflt` だけを指定した。

次に、6〜8行目の `feature liga {〜} liga;` は、`liga` というタグで指定される機能、つまりリガチャを定義する。ブラウザなどといったフォントを表示するソフトウェアは、機能を指定するこの4文字のタグ（feature tag）を基準にして、機能の有効無効を切り替えている。例えば、先ほどのブラウザの例で、リスト2.2の4行目 `font-feature-settings: "liga" 1;` は、`liga` というタグに紐付けられた機能をオンにするという設定である。`liga` というラベルの付いたスイッチのように考えてもよいかもしれない。

機能を表すタグはすでに用途が決められたものが登録されており[*10]、作ろうと思えば自分で勝手にタグを作ることもできるが、フォントを扱うソフトウェアがちゃんと取り扱ってくれるのかは分からない。

[*10] `https://www.microsoft.com/typography/otspec/featurelist.htm`

最後に、liga に結び付けられた機能の内容を見ていく。substitute h o m e by home; は、h と o と m と e という名前のグリフがこの順番で並んだとき、home という名前のグリフで置き換えるべし、ということを定義している。

ここで、コードのなかの h, o, m, e, home といったものはすべてグリフの名前であり、文字そのものでないことに気を付ける。GSUB が取り扱うのは文字でなくグリフである。グリフは、文字が最終的にどんな形状で表示されるかを指定した具体的な図形であり、GSUB が処理する単位でもある。

文字コードで表現された文字列は、まずそのコードポイントに対応するグリフへと置き換えられ[*11]、その後それを GSUB が処理するようになっている。グリフには一意に特定できる名前がつけられていて、feature file で指定するのもこのグリフ名である。グリフ名は標準でついているものもあるが、変更することもできるし、新しいグリフを追加することもできる。

さて、何となく GSUB の使い方の輪郭がつかめたところで、次は FizzBuzz の実装にとりかかる。

[*11] これは cmap という仕組みで行われる。

第3章

GSUB で FizzBuzz

3.1　方針

GSUB ではパターンマッチングによりグリフを置換することができるが、これは前から後ろへ行う。そのため、前から後ろへと情報を受け渡すことはいくらでもできるのだが、後ろから前へと情報を受け渡すのには制限がある。この点に留意する必要がある。

FizzBuzz 問題では、数字が a) 3の倍数であるか、b) 5の倍数であるかを判定する必要がある。このうち、b) の5の倍数判定は単純で、数字の一番最後の桁だけを見ればいい。すなわち、一番最後の桁が0か5であれば5の倍数という訳である。

a) の3の倍数判定はやや複雑である。ある数字が3の倍数であるかを判定するには、各位の数字の和が3で割り切れるかどうかを判定すればよく、割り切れるのであれば3の倍数である。この判定であれば、先頭からその桁までの各位の数字の和の3の剰余がいくつか、という情報を前から後ろに伝えていくことで、最後の桁で3の倍数かどうかを判定することができる。この情報は、3の剰余がそれぞれ0, 1, 2である場合の3状態を用意すると、図3.1のような状態遷移図で表すことができる。

最後の数字までこの状態遷移を行ったとき、剰余0であれば3の倍数、それ以外であれば3の倍数でないという訳である。

GSUB で状態を表現するには、状態の数だけグリフを登録することになる。0〜9の

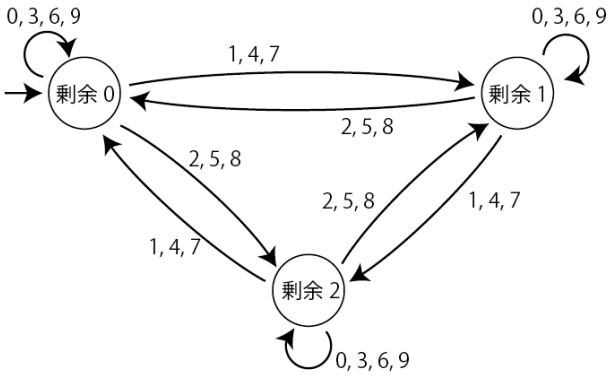


図 3.1. 3の剰余を判定する状態遷移図

数字に3つの状態を持たせる場合、$10 \times 3 = 30$ 個のグリフを追加することになる。

3.2 実装

さて、実際に実装するには次のような手順をとる。

1. 3の倍数か判定（3の剰余を求める）
2. 3か5の倍数であれば、最後の桁を Fizz、Buzz、FizzBuzz のいずれかに適切に置換
3. Fizz、Buzz、FizzBuzz の前にある残った数字を消去

3.2.1 グリフの用意

まず、前章と同様に、不要なグリフを削除したたたき台となるフォントを用意する。手順については2.4.2節を参照いただきたい。

それから、今回は3の倍数を判定するために、3の剰余を表現する3状態に対応する数字グリフを0〜9に対して追加しないといけない。これは `zero, one, two,`

three, four, five, six, seven, eight, nine に対応させて、

剰余0の場合: zero_0, one_0, two_0, three_0, four_0,
five_0, six_0, seven_0, eight_0, nine_0
剰余1の場合: zero_1, one_1, two_1, three_1, four_1,
five_1, six_1, seven_1, eight_1, nine_1
剰余2の場合: zero_2, one_2, two_2, three_2, four_2,
five_2, six_2, seven_2, eight_2, nine_2

という名前のグリフを計30個追加する。

また、“Fizz”, “Buzz”, “FizzBuzz” と表示するためのグリフ Fizz.liga, Buzz.liga, FizzBuzz.liga を追加する。

さらに、数字を消すために、何も表示しないグリフ null を追加する。

以上、合計34個のグリフを追加しないといけないが、前章でやったように一つ一つ追加して名前の変更をしていくのは面倒である。今回は、「名前を指定してグリフを作成」機能を使う。まず、一行に一つのグリフ名を書いたテキストファイルを用意し、FontForge のメニューから エンコーディング (N) 〉 名前を指定してグリフを作成 ((A)... を選択し、開いたダイアログで用意したテキストファイルを読み込むと、指定された名前でグリフが追加される（図3.2）。

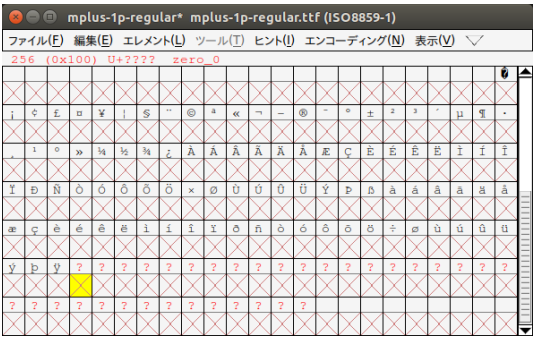
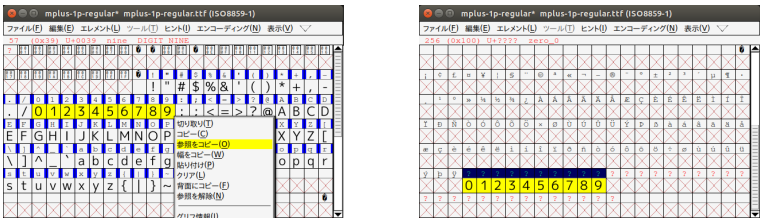


図 3.2. 指定した名前でグリフが追加された状態。

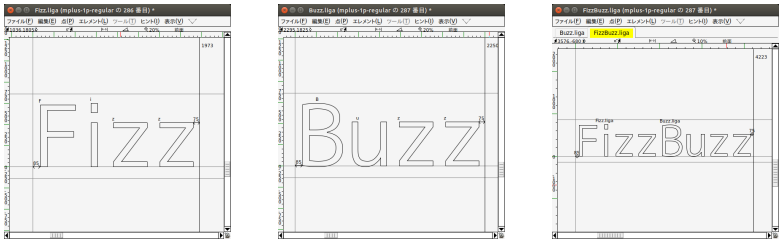
次に、グリフの図形を用意する。ASCII の0から9までのグリフを選択し（図3.3a）、右クリックから 参照をコピー (O) を選ぶ。次に、zero_0 を選択し、右クリックから 貼り付け (P) を選択し図形の参照を貼りつける（図3.3b）。同様に zero_1、zero_2 にも貼りつける[*12]。



(a) 数字グリフの参照をコピー　　(b) 貼り付け

図 3.3. 追加した数字グリフの図形を用意する。

Fizz.liga, Buzz.liga, FizzBuzz.liga については、適当にアルファベットのグリフをコピー&ペーストして作成した（図3.4）。幅は適当に設定してある。



(a) Fizz　　(b) Buzz　　(c) FizzBuzz

図 3.4. 作成した Fizz, Buzz, FizzBuzz のグリフ。

最後に null のグリフを選択し、右クリックの 幅を設定 (W)... から、グリフ幅の設

[*12] 追加されたグリフの順番が異なっている場合には、このように数字をまとめて貼りつけることがうまくいかないかもしれない。その場合は、グリフ名のアンダースコアより前の部分が表す名前と数字の図形を適切に合わせる。

定値に0を指定して OK ボタンを押し、幅を0にしておく。

以上でグリフの準備は終了（図3.5）、次は GSUB を設定していく。

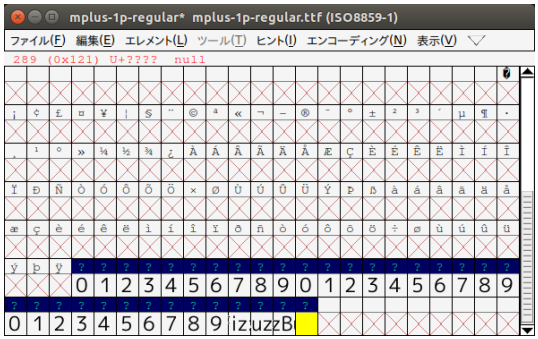


図 3.5. グリフの準備が終了した状態。

3.2.2 GSUB の設定

まず GSUB の設定ファイル全体を掲載する。後程解説するので、解説と見比べながら読んでほしい。

リスト 3.1. `fizzbuzz.fea`……FizzBuzz を実現する OpenType feature file。

```
languagesystem DFLT dflt; #デフォルト用字系・デフォルト言語
languagesystem latn dflt; #ラテン文字・デフォルト言語
#標準の数字
@num_normal=[zero one two three four five six seven eight nine];
#3の剰余の数字
@num_0 = [zero_0 one_0 two_0 three_0 four_0
          five_0 six_0 seven_0 eight_0 nine_0]; #3の剰余が0
@num_1 = [zero_1 one_1 two_1 three_1 four_1
          five_1 six_1 seven_1 eight_1 nine_1]; #3の剰余が1
@num_2 = [zero_2 one_2 two_2 three_2 four_2
          five_2 six_2 seven_2 eight_2 nine_2]; #3の剰余が2
#3の倍数であり、5の倍数でない数字（Buzzに対応）
@num_0_not0mod5 = [one_0 two_0 three_0 four_0
                   six_0 seven_0 eight_0 nine_0];
```

```
15 #数字グリフ全部
16 @num_all = [@num_normal @num_0 @num_1 @num_2];
17
18 #直前の数字に紐付いた3の剰余がそれぞれ0,1,2だったときの置換先
19 @num_after_0 = [zero_0 one_1 two_2 three_0 four_1
20                 five_2 six_0 seven_1 eight_2 nine_0];
21 @num_after_1 = [zero_1 one_2 two_0 three_1 four_2
22                 five_0 six_1 seven_2 eight_0 nine_1];
23 @num_after_2 = [zero_2 one_0 two_1 three_2 four_0
24                 five_1 six_2 seven_0 eight_1 nine_2];
25 #数字以外
26 @chr_notnum = [Fizz.liga Buzz.liga FizzBuzz.liga null];
27
28 lookup check_first_fig_mod3 { #1文字目の数字を3の剰余グリフで置換
29     ignore substitute @num_all @num_normal'; #最初以外無視
30     substitute @num_normal' by @num_after_0;
31 } check_first_fig_mod3;
32
33 lookup check_mod3 { #2文字目以降を3の剰余で置換
34     substitute @num_0 @num_normal' by @num_after_0;
35     substitute @num_1 @num_normal' by @num_after_1;
36     substitute @num_2 @num_normal' by @num_after_2;
37 } check_mod3;
38
39 lookup check_fizzbuzz { #3か5の倍数ならば最後の数字を置換
40     ignore substitute @num_all' @num_all; #最後の数字以外無視
41     substitute [zero_0 five_0]' by FizzBuzz.liga;
42     substitute @num_0_not0mod5' by Fizz.liga;
43     substitute [zero_1 five_1 zero_2 five_2]' by Buzz.liga;
44 } check_fizzbuzz;
45
46 # Fizz,Buzz,FizzBuzzがあれば数字を逐次消す
47 lookup delete_figures_16 {
48     substitute @num_all' @num_all @num_all @num_all
49                @num_all @num_all @num_all @num_all
50                @num_all @num_all @num_all @num_all
51                @num_all @num_all @num_all @num_all
52                @chr_notnum by null;
53 } delete_figures_16;
54 lookup delete_figures_8 {
55     substitute @num_all' @num_all @num_all @num_all
```

```
56                @num_all @num_all @num_all @num_all
57                @chr_notnum by null;
58  } delete_figures_8;
59  lookup delete_figures_4 {
60      substitute @num_all' @num_all @num_all @num_all
61                @chr_notnum by null;
62  } delete_figures_4;
63  lookup delete_figures_2 {
64      substitute @num_all' @num_all @chr_notnum by null;
65  } delete_figures_2;
66  lookup delete_figures_1 {
67      substitute @num_all' @chr_notnum by null;
68  } delete_figures_1;
69
70  feature liga{ #リガチャの設定
71      lookup check_first_fig_mod3; #1. 最初の桁が3の倍数か判定
72      lookup check_mod3;           #1. 2桁目以降が3の倍数か判定
73      lookup check_fizzbuzz;       #2. Fizz/Buzzに置換
74      lookup delete_figures_16;    #3. 16個隣の数字を消す
75      lookup delete_figures_8;     #3. 8個隣の数字を消す
76      lookup delete_figures_4;     #3. 4個隣の数字を消す
77      lookup delete_figures_2;     #3. 2個隣の数字を消す
78      lookup delete_figures_1;     #3. 1個隣の数字を消す
79  } liga;
```

用字系・言語の設定

最初に用字系と言語を設定している。これは前章と同じである。

```
languagesystem DFLT dflt; #デフォルト用字系・デフォルト言語
languagesystem latn dflt; #ラテン文字・デフォルト言語
```

グリフクラス定義

3〜26行の部分では、クラスの割り当てを行っている。`@num_normal` など、`@` で始まる名前はグリフのクラスを表す。クラスの内容については後で触れることにして、ここではクラス定義のやり方と使い方についてみていく。

クラス定義は `@` クラス名 `=[`グリフ $a_{␣}$ グリフ $b_{␣}$ グリフ $c_{␣}\cdots$`];` のように、`=` の左

側にクラス名、右側にクラスに含まれるグリフ名を空白文字で区切って [〜] の中に記述し、定義の終わりにセミコロン ; をかく。

置換を定義する substitute にはクラスを指定でき、複数の置換の定義を簡潔に書くことができる。次に挙げるリスト3.2の例は、a を A に、b を B に、c を C に、d を D に、e を E に置き換える。置換先にクラスを指定する場合、クラスの要素の数は置換元と等しくなっている必要がある。

リスト 3.2. いくつかの文字を大文字へ置換する例

```
@lowercase = [a b c d e];
@uppercase = [A B C D E];
substitute @lowercase by @uppercase;
```

また、置換先は一つのグリフを指定することもでき、置換元のクラスのいずれかのグリフが置換先のグリフで置き換えられる（リスト3.3）。

リスト 3.3. 0以外の数字を全部0に置換する例

```
@figs = [one two three four five six seven eight nine];
substitute @lowercase by zero;
```

さて、しばらく飛ばして、70〜79行目の liga の定義をみていく。

```
feature liga{ #リガチャの設定
    lookup check_first_fig_mod3; #1. 最初の桁が3の倍数か判定
    lookup check_mod3;           #1. 2桁目以降が3の倍数か判定
    lookup check_fizzbuzz;       #2. Fizz/Buzzに置換
    lookup delete_figures_16;    #3. 16個隣の数字を消す
    lookup delete_figures_8;     #3. 8個隣の数字を消す
    lookup delete_figures_4;     #3. 4個隣の数字を消す
    lookup delete_figures_2;     #3. 2個隣の数字を消す
    lookup delete_figures_1;     #3. 1個隣の数字を消す
} liga;
```

ここにはいくつかの lookup なるものが並んでいる。これらの lookup はそれぞれが置換を定義したものであり、上から下にこの順番で実行される。lookup は feature liga の定義より前で定義されている。次はこれら lookup を実行される順に見ていこう。

3の剰余の計算

最初の2つの lookup は数字が3の倍数かを判定するものだ。なぜ2つに分かれているかというと、先頭の数字を3の剰余の情報をもたせたグリフに置換するのと、そのグリフを足掛かりに3の剰余の情報を後に伝えていくのとを別々に行っているからである。

```
lookup check_first_fig_mod3 { #1文字目の数字を3の剰余グリフで置換
    ignore substitute @num_all @num_normal'; #最初以外無視
    substitute @num_normal' by @num_after_0;
} check_first_fig_mod3;

lookup check_mod3 { #2文字目以降を3の剰余で置換
    substitute @num_0 @num_normal' by @num_after_0;
    substitute @num_1 @num_normal' by @num_after_1;
    substitute @num_2 @num_normal' by @num_after_2;
} check_mod3;
```

check_mod3

実行順とは逆になるが、先に check_mod3 の方から見ていこう。この中で出てくる @num_normal は標準の数字を、@num_0, @num_1, @num_2 はそれぞれ3の剰余が0, 1, 2である数字を表すクラスである。@num_after_0, @num_after_1, @num_after_2 クラスは置換先を定義する。

ここで出てきた substitute は今までのものとは少し異なっていることに気付いただろうか。これは、文脈依存の置換（contextual substitution）といって、グリフの並びが条件にマッチする場合のみ、アポストロフィ' がついたグリフを置換するというものである。文脈依存の置換は、前から後ろに順番に条件をみて置換をおこなっていくが、それ以降の置換の条件で前のグリフを参照する場合、現在置き換えた後のグリフが使われることに留意する。この仕組みにより、単純な記述で3の剰余の情報を後ろへと受け渡していくことができる。

要するに、substitute @num_0 @num_normal' by @num_after_0; は、「『3の剰余が0の数字』（@num_0）に『標準の数字』（@num_normal）が続く場合、

『標準の数字』(@num_normal)を『置換先0』(@num_after_0)で置き換える」といった意味合いになる。この置換先については、前の桁までの各位の和の3の剰余に応じて、現在の桁までの各位の和の3の剰余が計算される様に設定する。要するに、図3.1で示した状態遷移を実装すればよい。実際の置換先の設定は表3.1のようになる。19〜24行目の @num_after_0, @num_after_1, @num_after_2 はこの表の内容を定義している。

表 3.1. 置換先の設定。3の剰余の情報を持つ数字は、剰余を下付きで表示した。

前の数字＼今の数字	0	1	2	3	4	5	6	7	8	9
剰余0(@num_0)	0_0	1_1	2_2	3_0	4_1	5_2	6_0	7_1	8_2	9_0
剰余1(@num_1)	0_1	1_2	2_0	3_1	4_2	5_0	6_1	7_2	8_0	9_1
剰余2(@num_2)	0_2	1_0	2_1	3_2	4_0	5_1	6_2	7_0	8_1	9_2

check_first_fig_mod3

前述したとおり、check_mod3 は3の剰余の情報を持つ数字に後続する場合を定義しているため、最初の数字を3の剰余の情報をもつものへと置換して足掛かりとしないと動かない。この足掛かりを作るのが check_first_fig_mod3 である。

check_first_fig_mod3 には最初に ignore substitute @num_all @num_normal'; と書かれている。@num_all は、標準の数字と3の剰余の情報をもつ数字を合わせた全数字グリフを表すクラスである。これは、@num_normal クラスのグリフの前に @num_all クラスのグリフが来る場合、@num_normal に対しての置換を行わないという指定で、数字が前に来ない場合、つまり数字の先頭のみ置換が実行される。この置換は、前の数字において3の剰余が0であるときと等しいので、@num_normal を @num_after_0 で置き換えている。

実際に「1234567890987654321」という数字列に check_first_fig_mod3 と check_mod3 を適用させると、それぞれ図3.6、図3.7のようになる。

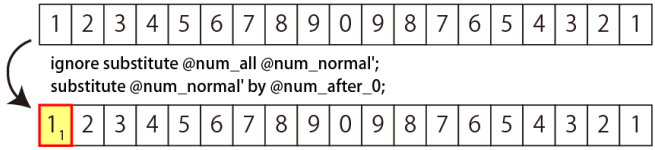


図 3.6. check_first_fig_mod3……最初の桁の数字を3の剰余の情報をもつものに置換する。

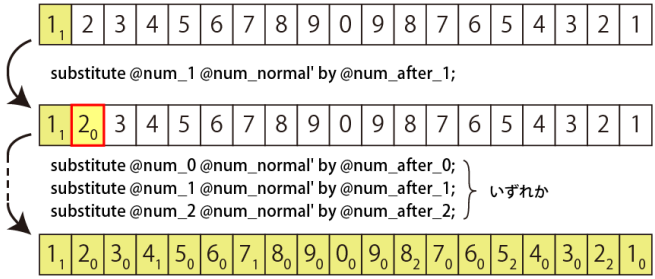


図 3.7. check_mod3……2桁目以降の数字を3の剰余の情報をもつものに置換する。

check_fizzbuzz

次に、32〜44行目の check_fizzbuzz を見ていく。

```
lookup check_fizzbuzz { #3か5の倍数ならば最後の数字を置換
    ignore substitute @num_all' @num_all; #最後の数字以外無視
    substitute [zero_0 five_0]' by FizzBuzz.liga;
    substitute @num_0_not0mod5' by Fizz.liga;
    substitute [zero_1 five_1 zero_2 five_2]' by Buzz.liga;
} check_fizzbuzz;
```

最初の ignore substitute @num_all' @num_all; で、後ろに数字が来る場合を無視し、最後の数字だけ置換される様に指定している。

次に substitute [zero_0 five_0]' by FizzBuzz.liga; であるが、

[zero_0 five_0] という表記は、この中のいずれかという、クラス表記と同様の意味をもつ。ここでは、3の剰余が0でかつ最後の桁が0または5という、要するに15の倍数である場合を示している。この場合、最後の数字を“FizzBuzz”を表示するグリフ FizzBuzz.liga に置換する。

同様に substitute @num_0_not0mod5' by Fizz.liga; は、3の倍数でかつ5の倍数でない場合を定義し、3の剰余が0である0と5以外の数字 @num_0_not0mod5 を“Fizz”を表示するグリフ Fizz.liga に置換する。

最後に substitute [zero_1 five_1 zero_2 five_2]' by Buzz.liga; は5の倍数でかつ3の倍数でないときの置換を定義し、3の剰余が0でない0または5の数字を“Buzz”を表示するグリフ Buzz.liga に置換する。

実際に check_fizzbuzz は図3.8のような置換を行う。

図 3.8. check_fizzbuzz……3または5の倍数であれば最後の数字を Fizz, Buzz, FizzBuzz のいずれかに適切に置換する。

最終的に残ったのが 47〜68行目にある lookup 群、delete_figures_16, delete_figures_8, delete_figures_4, delete_figures_2, delete_figures_1 であるが、これらは最後の数字が Fizz, Buzz, FizzBuzz のいずれかに置換されていた場合に、それ以前の数字を何も表示する要素のないグリフ null で置き換えて消すためにある。前述したように、GSUB 自体が前から後ろへと置換を行っているので、情報を後ろから前へと伝えるのは制限があり、「後ろの文脈を見て置換する」という lookup を何回も適用することで、ある程度は後ろから前に情報を伝えることができるが、定義した回数より多い場合には対応ができない。実際、これでは33桁以上の場合は消しきれないが、上限があるのはどうしようもない。

クラス @chr_notnum は、Fizz.liga, Buzz.liga, FizzBuzz.liga, null

の4つのグリフを指す。まず delete_figures_16 が、@chr_notnum の16個前の数字を null に置き換える。次に delete_figures_8 が、@chr_notnum の8個前の数字を null に置き換える。同様に delete_figures_4, delete_figures_2, delete_figures_1 が、@chr_notnum の4, 2, 1個前の数字を null に置き換えていくと、最大31個までの数字を消すことができる。これは、図3.9のように動作する。

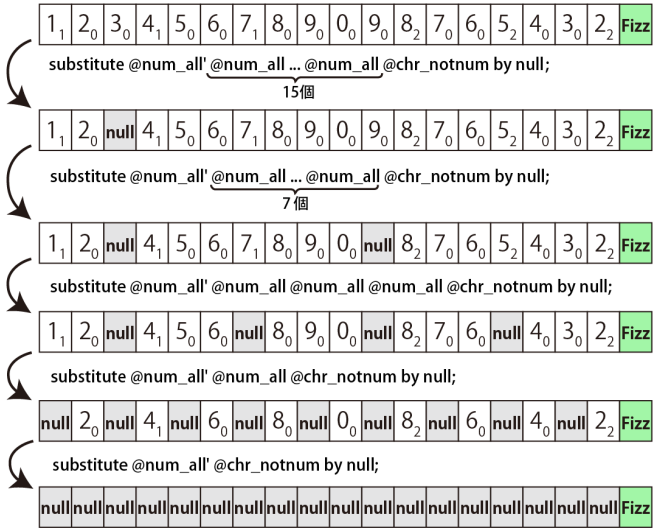


図 3.9. delete_figures 群……Fizz の前の数字を null で置き換える。

このようにして、FizzBuzz を GSUB で表現することができる。

3.2.3 OpenType feature file の適用

前章の場合と同様、FontForge のメニューの ファイル (F) 》 Merge Feature Info... でリスト3.1のファイルを取り込むと、メトリックウィンドウで動作を確認することができる（図3.10）。

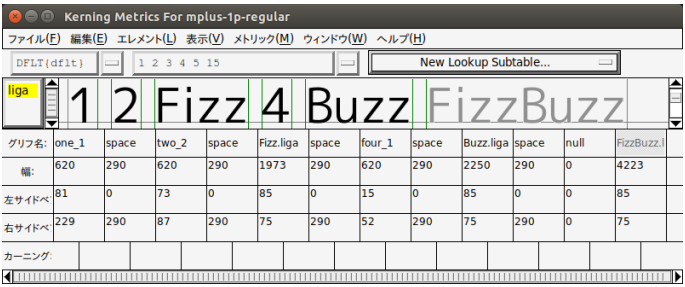


図 3.10. FizzBuzz を GSUB で実現したフォントの動作を FontForge のメトリックウィンドウで確認してみる。

これもフォントとして出力し、前章と同様にブラウザ（Firefox）で表示してみると図3.11のようになり、正常に動いていることが確認できる。

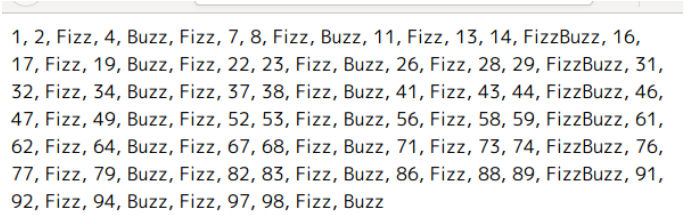


図 3.11. FizzBuzz を GSUB で実現したフォントの動作をブラウザで確認する。

第4章

おわりに

OpenType 機能自体は複雑な用字系や美的に優れた組版をするための要請で実装された（のではないかと思う）が、それによってアイコンフォントや FizzBuzz といったものが実現でき、非常に多彩な表現力をもっていることが垣間見れる。

実は OpenType 機能以外にも FizzBuzz を実現することができるような機能もフォント仕様の中に存在しているのだが、それらは対応する OS や環境がまちまちであり、一方で OpenType 機能は国際化対応をするにあたって必須であることから、現在ほとんどすべての OS でサポートされ、多くのプログラムで利用できる。

本文中に日本語ではあまり出番がないと書いたが、日本語においても GSUB などの OpenType 機能は利用されていて、その最たるものが縦書き時のグリフの切替である。小書き仮名の「っ」「ょ」「い」などや、棒引き「ー」などは、横書き時と縦書き時では、書かれる位置や方向が異なっているので、これを縦書き時には縦書き用の形に切り替えるという仕組みが多くの日本語フォントに GSUB を利用して実装されている。フォントを作る機会があれば、活用してみると面白いのではないかと思う。

GSUB を活用したフォント

最後に GSUB の表現力をフル活用したフォントをいくつか紹介する。

■Timepiece Rounded　http://timepiece.inostudio.de/

アナログ時計を表示するフォント。22:33:41 のような文字列を図4.1のようなアナログ時計に変換することができる。

図 4.1. Timepiece Rounded の表示例。
http://timepiece.inostudio.de/より。

■FF Chartwell　https://www.fontshop.com/families/ff-chartwell/

棒グラフ、折れ線グラフ、円グラフ、レーダーチャート等が作成できるフォントで、例えば棒グラフであれば a+50+35+95+15+65 といった文字列を入力してグラフを表示することができる（図4.2）。

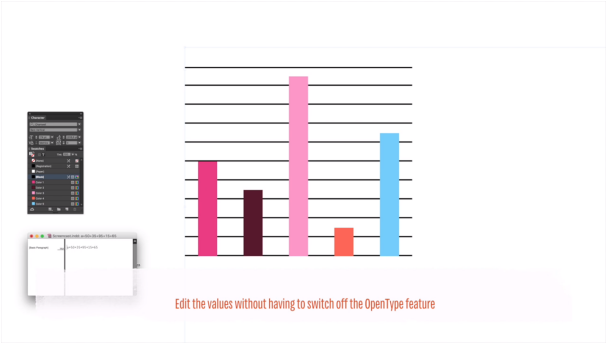


図 4.2. FF Chartwell の棒グラフの例。
https://vimeo.com/154854315 より。

あとがき

フォント島に配置されたのでフォント本を作るしかないと思って作ってみた。結局は自分のブログ記事の焼き直しである。たぶん、多くの人が求めるフォント本とは異なっているのだろうけれど、フォントで使われている技術の紹介として興味をもってもらえたらありがたい。

奥付

書名	フォントで FizzBuzz する方法(1): OpenType 機能
発行	ヒュアリニオス
著者	にせねこ (@nixeneko)
発行日	2017年12月31日(コミックマーケット93)
連絡先	http://hyalinios.hatenadiary.com/
サポートページ	http://hyalinios.hatenadiary.com/entry/c93-gsub