令和 7 年 11 月 11 日
京都工芸繊維大学コンピュータ部

Lime 65

はじめに

コンピュータ部部長の中山です。この度はお忙しい中、本学の学祭にお越しいただき、我々コンピュータ部にご興味・ご関心を持っていただきありがとうございます。そして、この Lime65 号を手に取ってくださったことを、部を代表して厚くお礼申し上げます。この Lime では、部員の活動の一部を記しており、日々の活動の成果や部員個人の趣味など、内容は多岐にわたります。この部誌を通じて、コンピュータ部のことを知り、少しでも我々の活動に興味を持っていただければ幸いです。そして、今回の Lime 作成にあたり編集を担当した樋口さん、記事を執筆した部員各位、ならびに OB・OG の方々を含め、日々我々の活動を支えてくださっているすべての方々への感謝を申し上げ、始めの挨拶とさせていただきます。

令和 7 年 11 月 3 日

京都工芸繊維大学コンピュータ部部長　中山 実玲

目次

1 万博体験レポート

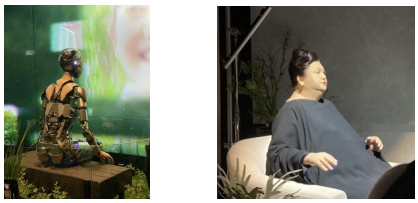
情報工学課程 3 回生 中山 実玲

1.1 はじめに

2025 年 8 月、夏休みを利用して大阪・夢洲で開催中の大阪・関西万博を訪れました。多くの企業や大学が未来の暮らしや医療、都市のあり方を展示しており、技術と人間社会の関係を実感できる内容でした。この記事では、特に印象に残った 4 つのパビリオンを紹介したいと思います。

1.2 いのちの未来パビリオン

「いのちの未来」はモノにいのちを宿してきた歴史、高度な技術で暮らす 50 年後の未来、1000 年後の進化した人間の 3 つのゾーンで構成されており、案内役のロボットたちが私たちを誘導してくれました。特に印象的だったのは、二つ目のゾーンで描かれていたおばあちゃんと孫のストーリーです。このゾーンでは、アンドロイドやロボットが人間と共存する存在として登場し、おばあちゃんが人間としてそのまま死を迎えるのか、それとも孫のために自身の記憶や人格をアンドロイドにコピーして生き続けるのかという選択が描かれていました。難しいテーマでありながら、とても感動的で、思わず涙がこぼれました。



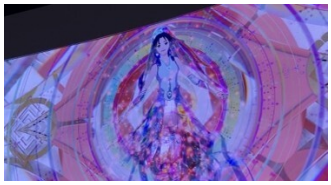
(a) 展示風景 1　　(b) 展示風景 2

図 1.1　いのちの未来パビリオンの展示風景

1.3 飯田グループ×大阪公立大学パビリオン

住宅メーカーと大学が協力し、脱炭素社会を目指す新エネルギーや先進技術を用いた、快適で健康的な暮らしのための「未来の住まい」と「まちづくり」が紹介されていました。構造模型や住宅デザインが展示されており、日常生活に未来の技術が自然に溶け込むイメージを感じました。パビリオンの外観は、特殊加工された西陣織の生地を全面にまとっており、未来と伝統の融合を象徴していました。中に入ると、オリジナルキャラクター「プロトン」が来場者を出迎え、動画を通して未来のまちの姿を紹介していました。また、未来の住宅デザインも紹介されており、実際の生活を身近に想像しやすい展示でした。私がこのパビリオンで特に印象に残ったのは、未来都市の模型です。私は見つけられませんでしたが、その模型には「隠れミャクミャク」が数体潜んでいたそうです。

(a) 外観



(b) キャラクター

(c) 模型

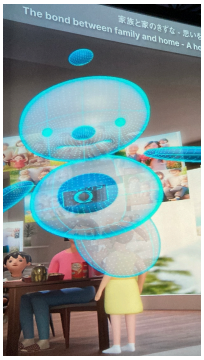
図 1.2 飯田グループ×大阪公立大学パビリオンの様子

1.4 未来の都市パビリオン

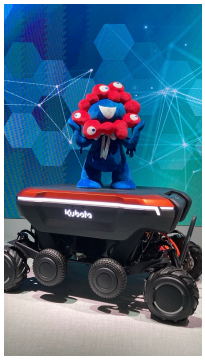
Society 5.0 をテーマとして、複数の展示がされていました。どちらかといえば体験がメインであり、体験コーナーが多数ありました。それに加えて、ショーなども展開されており、来場者をわくわくさせる仕組みが多いパビリオンでした。パビリオン内では、未来の街について、さまざまな予想がされており、私が一番印象的だったのは、食に関するコーナーです。今までの歴史から、私たちの食事が描かれており、現在の食事、これからの未来の食事の予想についても知ることができました。

1.5 大阪ヘルスケアパビリオン

「Reborn」をテーマに、食・運動・予防医療の最前線を紹介していました。AI による健康診断体験、iPS 細胞の展示、ミライ人間洗濯機の紹介、そして未来の食と文化に関するコーナーなど、幅広い分野の内容が揃っていました。さらに「リボーン体験ルート」など



(a) 展示風景 1



(b) 展示風景 2

図 1.3　未来の都市パビリオンの展示風景

の体験型コーナーもあり、来場者が楽しみながら健康について考えられる構成になっていました。展示デザインも明るく開放的で、来場者が気軽に健康への関心を持てる工夫がされていました。私は体験型を予約することはできず、実際に体験はできませんでしたが、iPS 細胞やミライ人間洗濯機などを見ることができました。特に、培養された心筋シートが実際に動いている様子を見て、生きている細胞の存在を実感しました。

(a) 心筋シート

(b) 人間洗濯機

図 1.4　大阪ヘルスケアパビリオンの展示風景

1.6　感想

今回の万博では、住宅・医療・都市など、さまざまな分野が将来の暮らしの新たな形を模索していることを感じました。朝から夜まで会場を巡り、広い敷地と多くの来場者に圧倒されながらも、どの展示も体験を通じて未来社会の方向性を示しており、非常に印象深いものでした。ずっと歩き続けて昼食を取るのも忘れるほど夢中になり、疲れは感じたものの、それ以上に楽しく、未来について考えさせられる良い機会になりました。技術だけでなく、人の暮らしや社会全体の在り方を考えるきっかけとなり、満足感のある貴重な体

験でした。

参考文献

[1] 万博公式サイト: https://www.expo2025.or.jp/

※掲載写真は筆者が会場で撮影したものであり、非営利目的で使用しています。

2 C# 言語を用いた電卓の作成

情報工学課程 3 回生 田中大貴

2.1 はじめに

私は, 物理的な電卓では四則演算を正しい順序で行うことができないことに疑問を抱いていました. 物理的な電卓が正しい順序で演算を処理できないことにはどのような背景があるのでしょうか. 一方で, スマートフォンなどにインストールされている電卓は, 正しい順序で演算を処理できます. 同じ「電卓」なのに, なぜこのような差があるのでしょうか.

私は, 四則混合算を正しい順序で処理できる電卓の開発を通し, これらの疑問を解決することにしました.

2.2 概要

私は, GUI 開発の前に, 開発しようとしているものを実現できそうか知ること, 実現のための手法を学ぶことを目的として CLI 開発を行います. この部誌では, GUI 開発についてのみ説明いたします.

2.2.1 目標

(1) GUI 開発について
- ソフトキーボードキーボードとキーボードの両方の入力に対応すること
- 演算処理の過程を表示すること

(2) 数式処理について
- シャンティングヤード法により, 一般的な数式を逆ポーランド式記法に変換できるようになること
- 逆ポーランド式記法から演算結果を導出できるようになること

2.3 演算処理の違い

四則演算の演算順序は大まかには

(1)「かけ算」と「わり算」 は 「たし算」と「ひき算」 より先に計算する
(2) 左から順に計算する

の 2 つのルールで説明できます.

物理的な電卓では, 今記憶している数と, 新たに入力された数について演算を行うことを繰り返します. つまり, 式全体は見えず, 直近の計算結果とあらなた入力しか見えません. そのため, ヒトが左から順に式を入力する限り, (1) が守られず, (2) は守られます.

一方, スマートフォンにインストールされている電卓では, 入力している部分全体を計算するとどうなるかを入力途中であっても表示する, あるいは, 「=」などのボタンを押して初めて入力全体を見て演算結果を表示することが多いでしょう. いずれにしても, 式の全体を見て, (1) と (2) の両方を満たすように処理をしていることは間違いないでしょう.

$$1 + 2 \times 3 + 4 \tag{2.1}$$

私たちが扱う, (1.1) のような式には優先順位の情報は含まれておらず, 2 ページで示した優先順位に関する条件が必要です. そのため, 優先順位の情報を含む表記法に変換したいと考えます. 優先順位の情報を含む表記法が**逆ポーランド式記法**です. (1.1) を逆ポーランド式記法に変換すると, (1.2) となります. ここでは, 数字や演算子を「,」で区切っていますが, スペースなどで区切ることもあります. また, (1.1) のような通常の式を (1.2) のような逆ポーランド式記法に変換するアルゴリズムを**シャンティングヤード法**といいます. これらについて, 詳しくは [2] をご覧頂けますと幸いです.

$$1, 2, +, 3, \times, 4, + \tag{2.2}$$

2.4　実行

初期画面を図 1.1 として示します.

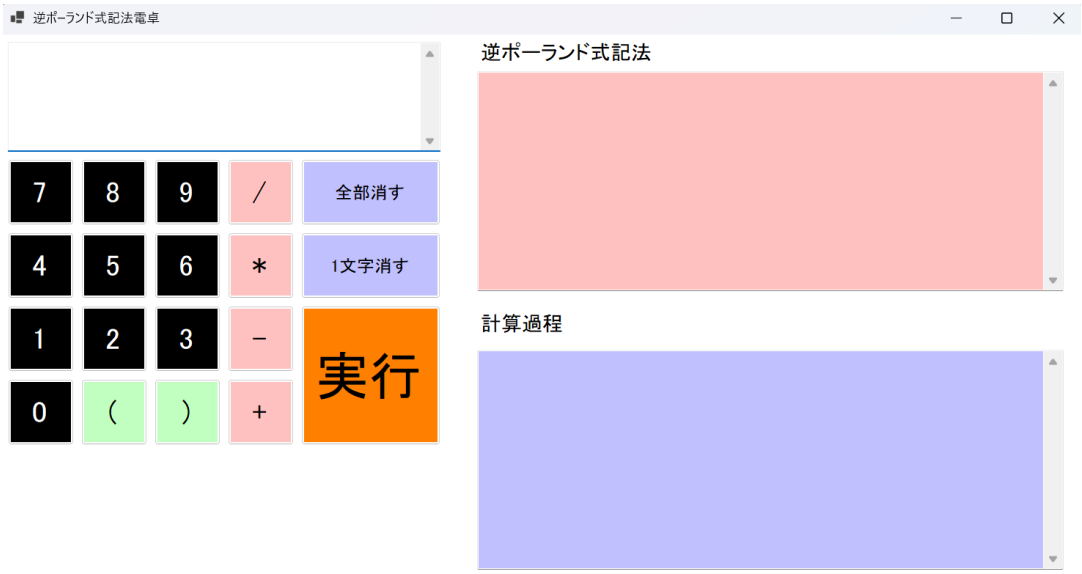


図 2.1　初期画面

数式は, ソフトキーボードまたは, パソコンのキーボードにより入力します. 「実行」ボタンを押すことで, 入力された数式をシャンティングヤード法により逆ポーランド式記法に変換する処理と, 逆ポーランド式記法から演算を行う処理が実行されます. 逆ポーランド式記法に変換する過程は赤色のテキストボックスに, 演算処理の過程は青色のテキストボックスに出力されます.

なお, 「$--1+2$」のように, 符号が 2 つ続く数式や, 「$((1+2)*3+4$」のように括弧が対応していない数式の入力エラーを検出する機能を実装しています.

例えば, (1.1) を処理する様子を図 1.2 として示します.

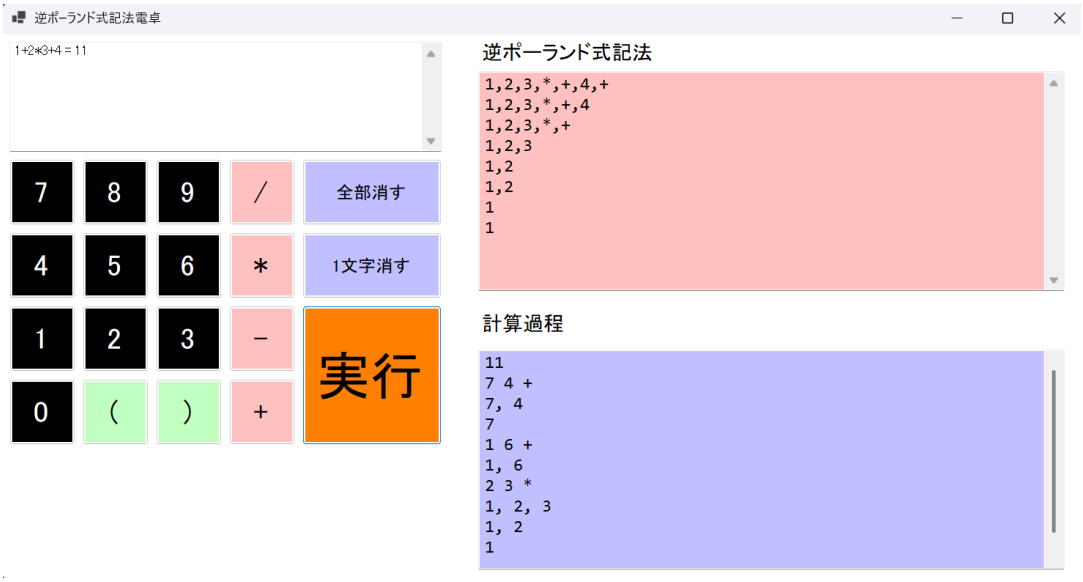


図 2.2　(1.1) の処理結果

2.5　改善できる点

[3] この電卓では, 整数のみを扱っています. そのため, 小数についての演算をできるようにしたくなります. 安直に考えると, 単精度や倍制度の浮動小数点数の変数に値を格納すればよい, という発想になりそうですが, この方法では不十分とされています.

そこで, 0.6 ではなく, 6×10^{-1} のように, (整数) $\times 10^{(\text{整数})}$ の形で記憶します. このとき, 仮数と指数の 2 つの整数の組として記憶するようにします. もちろん, この方法でも正しく処理できない計算は少なくありませんが, 正しく処理できるものを増やすことができます.

それでは, 小数を正しく入力できたか否かをどのように判定すればよいでしょうか. ここでは, 決定性有限オートマトンの状態遷移図を考え, それに対応するプログラムにより, 適正な形式で入力されたか判定するものとします. 状態遷移図を図 1.3 として示します.

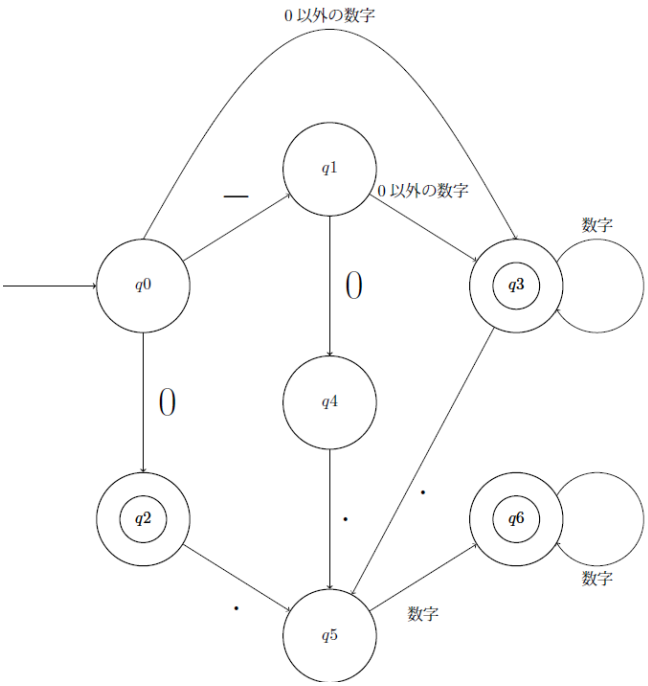


図 2.3　小数を受理するクラスの決定性有限オートマトンの状態遷移図

2.6　おわりに

自分で電卓を作ることで, 電卓の内部でどのように処理が行われているかを知ることができました. その際に, 物理的な電卓とスマートフォンにインストールされている電卓それぞれで行われる処理の差異を, 背景とともに学びました. そして, これまで「どういうわけか処理の手順や結果が異なる」とだけ考えていた段階から, より深く知ることができました.

まだまだ改善するべきところがある, と思われたかもしれませんが, 最後までお読み頂き, ありがとうございました.

参考文献

[1] 具体例で学ぶ数学,2019,”計算の順番のルールと例題（+-×÷とカッコ、累乗）”,2025 年 9 月 10 日閲覧 https://mathwords.net/junban

[2]【逆ポーランド記法】 なぜ、電卓は小学生レベルの計算を間違えるのか？【ずんだもん解説・ゆっくり解説】,2025 年 9 月 11 日閲覧 https://www.youtube.com/watch?v=iGLOXFjTmO0

[3] プログラムはなぜ小数の計算をミスるのか？,2025 年 9 月 17 日閲覧 https://www.youtube.com/watch?v=2pyWu8MVqdA

3 AIで作るLaTeX回路ツール

電子システム工学過程 3 回生 近藤 偉希

3.1 はじめに

今年の前期の実験から、LaTeX を使ってレポートを書くようになりました。LaTeX とは、数式や図表を美しく整えられる便利なツールです。電子系に所属しているので、回路図を描く機会も多く、LaTeX で回路図も描きたいと思いました。しかし、LaTeX で回路図を描くには、circuitikz というパッケージを使用する必要があり、部品の位置をすべて座標で指定しなければならないため、複雑な回路になると非常に手間がかかります。例えば、図 3.1 のような回路図を描こうとすると、コード 3.1 を数十行にわたって書かなければなりません。

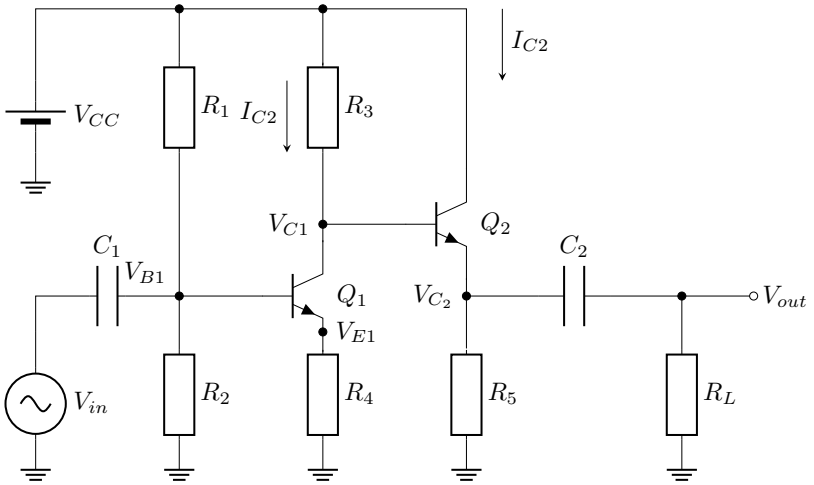


図 3.1 LaTeX（circuitikz）で描いた回路図

```
1 \draw (3, 6.25) to[european resistor, l={$R_4$}, label distance
      =0.02cm] (3, 5);
2 \draw node[npn] (N1) at (3, 7) {} node[anchor=west] at (N1.text)
      {$Q_1$};
3 \draw (-1, 6) to[sinusoidal voltage source, l={$V_{in}$}, label
```

```
    distance=0.02cm] (-1, 5);
\draw node[circ] (N5) at (1, 7) {} node[anchor=south east] at (N
    5.north west){$V_{B1}$};
\draw[-stealth] (2.5, 10) -| (2.5, 9)
  node[midway, below left, yshift=-5pt] {$I_{C1}$};
```

ソースコード 3.1 circuitikz での回路図コード

これはほんの一部ですが、実際にはこのような命令を何十行も書く必要があります。そこで、マウス操作で簡単に回路図を描き、それを LaTeX のコードに変換するツールを作成してみようと思いました。

3.2 AI コーディングを使った理由

私にはアプリの開発経験がなく、どのように作ればよいかわかりませんでした。特に回路図を描くためのマウス操作のアプリを作るには多くのコードを書く必要があり、独学で一から作るのは難しいと感じました。そこで、AI にコードの生成を手伝ってもらうことで、効率的に開発を進められるのではないかと考えました。また、AI コーディングが一般的になりつつある現代において、情報系ではない自分がすべてを一から学ぶよりも、AI をうまく活用して「作りたいものを形にする」力を身につけることが重要だと感じました。AI を使えば、基本的な文法の理解さえあれば、専門知識がなくてもアプリを開発できると考えました。開発には、OpenAI の Codex を使用しました。当時最も高性能とされ、世界的にも注目を集めていたからです。

3.3 開発の流れと工夫した点

開発言語には Python を選びました。自分が基本的な文法を理解している言語の中で、最もアプリ開発に向いていると感じたからです。

開発の進め方は、まず設計書を簡単にまとめ、AI に最低限の機能だけを持つアプリを生成してもらい、そこから少しずつ機能を追加していくという方法を取りました。この形にした理由は、最初から完全な設計書を作るのは大変であり、また AI が初めから完璧なプログラムを生成できるとは限らないと考えたためです。さらに、最初にすべての機能を一度にお願いしてしまうと、アプリの骨格が曖昧なまま複雑になり、意図しない方向に進んでしまう可能性があると感じました。

プログラムがうまく動作しないときは、まず AI に聞いて、それでも解決できないときは自分で調べて修正しました。

工夫した点は、AI に指示を出す際にできるだけ細かく内容を分けて伝えることです。大きな機能を一度にお願いすると、コードが動かなくなったり、構造が理解しづらくなることがありました。そのため、一つひとつの小さな機能を個別に生成・確認しながら、AI

が出力するコードを自分でも理解できるよう意識して開発を進めました。

3.4　完成したツールの紹介

完成したアプリの起動画面と、実際に回路を描いた例を図 3.2, 3.3 に示します。マウス操作で部品を配置し、配線をつなぐだけで回路図を作成できます。作成した回路図は、ボ

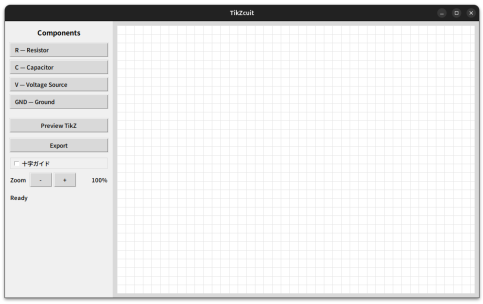


図 3.2　起動画面

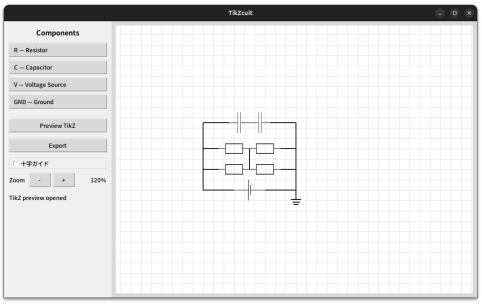


図 3.3　回路を描いた画面

タンを押すだけで LaTeX コードに変換できます。そのコードをコンパイルすると、図 3.4 のような図が生成されます。このように、位置を細かく指定しなくても、マウス操作で簡

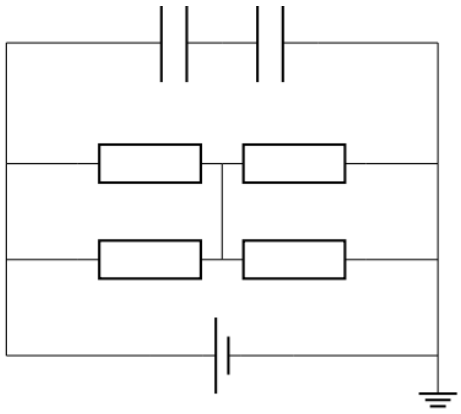


図 3.4　図 3.3 の回路図をアプリで LaTeX のコードにして実行した結果

単に回路図を描けるようになりました。

3.5　まとめ

今回の開発では、AI コーディングを活用して簡単な LaTeX 用の回路図作成ツールを完成させることができました。アプリの開発経験がなく、基本的な文法しか知らなくても、AI の力を借りることで、ある程度実用的なツールを作れることを実感しました。今後も AI を活用して、自分の勉強や暮らしが少しでも快適になる実用的なアプリを作っていき

たいです。

4 記録に残す計算トレーニング

情報工学課程 3 回生 田中大貴

4.1 はじめに

私の趣味の 1 つに, 暗算があります. もちろん, 難しい暗算はできませんが, 平易なものを素早く暗算できるようにトレーニングできるツールを開発したいと考えました. 暗算が趣味である状態は今に始まったことではないので, 過去に, GUI つきのアプリケーションを開発しました. しかし, このツールでは, 結果を記録に残せません. 結果を記録に残すことができるプログラムをコマンドラインで実行できる形式で開発しました.

この記事では, プログラムの内容も説明しますが, 実行の仕方に重きを置いて説明します.

4.2 概要

このプログラムは, C 言語で記述されています. はじめにかけ算のみ出題されるプログラムを作成しましたが, 後からその他の演算も出題されるプログラムを作成しました. そのため, この記事では, 両方のプログラムについて説明します.

これらのツールは, 使用するツールの名前, 難易度, 問題数をターミナルに入力して使います. 詳しい使い方は, 「実行」で説明します.

4.3　実行

初期画面を図 4.1 として示します．この例は，すべての演算に対応したプログラムのものですが，かけ算だけのプログラムでもほとんど同じです．

図 4.1　開始画面の例

さて，このプログラムでは，実行した結果を `No.XX-c.txt` と `No.XX-cf.txt` というファイルに記録しています．どちらのファイルも，そのまま見ることは想定されていません．しかし，`No.XX-cf.txt` は，決められた手順で処理することで，PDF に変換できるようにしています．

なお，かけ算のプログラムでは，実行した結果を `No.XX.txt` と `No.XXf.txt` というファイルに記録しています．また，`No.XXf.txt` は PDF 変換ができるようになっています．

図 4.1 の続きを計算した例を図 4.2 として示します．

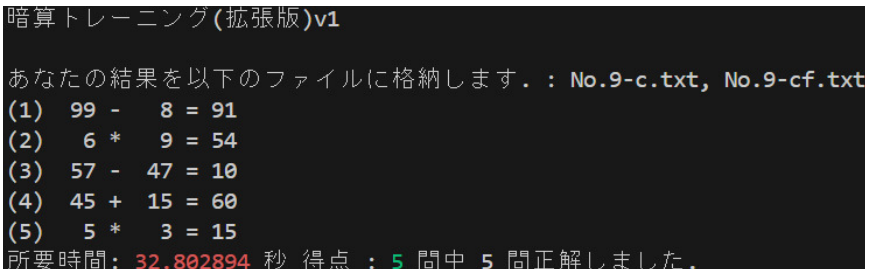


図 4.2　開始画面の例

4.3.1　始め方

実行するときは，ターミナルに指定された文字列を入力します．

全ての演算を出題してほしいときは，`./ ct` **［難易度］［問題数］** と入力します．

かけ算だけを出題してほしいときは，`./ mt` **［難易度］［問題数］** と入力します．

但し，[ ] で囲った部分は，数字に置き換えて下さい．例えば，ct を用いて難易度 10 の問題を 5 問を解きたいときは，`./ ct 10 5` と入力してください．

4.3.2 演算

全ての演算を出題するプログラムでは, たし算, 引き算, かけ算, 商, 剰余の 5 種類の演算を扱います. たし算と引き算は, 普通の計算なので, 説明することはありません.

かけ算は, 記号が $\times$ ではなく, 「$*$」という記号に変わっています. 但し, 計算は $*$ を「$\times$」に読みかえたものと同じです. 例えば, 図 4.2 の (2) は, $6 * 9$ ですが, 6×9 と読みかえて, 答えは 54 です.

商は, 記号が「/」で表されます. 例えば, 9 / 6 であれば, $9 \div 6$ の結果が 1 あまり 3 なので, 商は 1, と考えられます. また, $9 \div 6 = 1.5$ から, 答えの整数部分を考えて 1 とすることもできます.

剰余は, 余りのことです. 記号は「%」で表されます. 例えば, 9 % 6 であれば, $9 \div 6$ の結果が 1 あまり 3 なので, 余りは 1, と考えられます. また, 割り算の定義 $9 = 6 \times 1 + 3$ から, 余りを考えて 3 とすることもできます.

4.3.3 難易度

入力する「難易度」について説明します. ここでいう難易度を上げると, 計算式に出てくる数が大きくなります. そのため, 必ずしも難しくなるわけではありません.

何回も取り組む中で調整しても良いですが, そのような時間がない場合もあると思いますので, どういう処理で数を決めているかを見て, 難易度をどうするか決めてください.

たし算と引き算は, 出てくる数の上限は難易度の 2 乗です. 例えば, 難易度が 10 なら, 上限値は 100 となります.

かけ算は, 答えが大きくなりやすいので, 出てくる数の上限は難易度と同じです. 例えば, 難易度が 10 なら, 上限値は 10 となります.

商と剰余は, 出てくる数の上限は難易度の 2 乗です. 但し, 割られる数 (左の数) のみ難易度の 2 乗が上限となり, 割る数の上限は難易度と同じです. 例えば, 難易度が 10 なら, 2 桁の整数を 1 桁の整数で割る問題が多く出題されます.

4.3.4 記録

PDF 化するには, 以下の手順を踏みます. 松ヶ崎祭当日は, プログラム ct, mt を実行して頂いた方には, 私のパソコンで以下の手順を踏むことで, PDF データをお渡しすることができます.

(1) `No.XX-cf.txt` または `No.XXf.txt` という名前のファイルを Windows のフォルダ (ディレクトリ) にコピーまたはダウンロードします.
(2) Excel で (1) で用意したファイルを開きます. このとき, **ファイル¿開く¿この PC** から, 先ほどのファイルを指定します. このとき, ドロップダウンリストにより, 探すファイルを, **すべてのファイル**に変更します.
(3)「テキストファイルウィザード」が開きますので, データ形式は, 「**コンマやタブなどの区切り文字によってフィールドごとに区切られたデータ**」を選び, 次へを押します.
(4) フィールドの区切り文字は, 「**その他**」を選び, テキストボックスに「|」を入力してください.
(5) 完了を押します.
(6) タイトルの行を含めてすべての出力をコピーします. CalcMultiplyRes.xlsx の「入力」の A1 セルを選択し, 貼り付けることで, 「結果用紙」に結果を表示できます. 「結果用紙」を表示したまま,「ファイルの種類」を「PDF」に変更して保存します.

4.4 改善できる点

問題の難易度調整をもう少し上手くやることができると考えています. また, ソースコードについては, 無駄があり, 改善ができると考えています.

4.5 おわりに

自分で使う用に作った暗算のトレーニングのツールを, 松ヶ崎祭で公開するために改変したものなので, 改善できる点が多くあったかと思います. そんな中で最後まで読んで頂き, ありがとうございました.

5 画像をテキストで書く（SVG 入門）

情報工学課程 4 回生 石倉 朋佳

5.1 はじめに

この記事では、初めて画像をテキストで書いたという入門者の新鮮な感想をお届けします。「画像をテキストで書く」の定義次第では初めてではないのですが、少なくとも SVG を書くのは初めてでした。これまで、Android アプリ開発をする中で SVG や XML で書かれたファイルを使う機会がありましたが、ベクター画像で中身はテキストなんだな（書くの大変そう）、くらいのイメージで素通りしていました。ふとソースコードを書くことで画像を描きたいと思ったので、SVG について調べて書いてみました。

5.2 SVG について

SVG (Scalable Vector Graphics) とは、XML ベースのソースコードでベクター画像を記述するフォーマットです。テキストやラスター画像の表示、透過、アニメーションなども記述できます。

コンピュータでの画像の表現方法にラスター画像とベクター画像があります。ラスター画像とは、ピクセルが並んだもので表現する画像であり、拡大すると曲線などにギザギザ（ジャギー）が見られます。ベクター画像とは、図形の集まりで表現する画像であり、拡大縮小や高画質に強いです。

SVG は、HTML や CSS、JavaScript と合わせて Web で使われたり、Android アプリ開発で使われたりします。例えば、ロゴやアイコン画像に使ったり、UI を記述したりすることができます。

5.3 初めての SVG

SVG の例をソースコード 5.1 と図 5.1 に示します。これらは、同じ redstart.svg という SVG 形式のファイルを、ソースコード 5.1 はテキストエディタなどでテキストとして開いた結果であり、図 5.1 はブラウザなどで画像として開いた結果です。

ソースコード 5.1 は参考文献 [1] および [2] のコードを参考に作成しました。

```
1  <?xml version="1.0"?>
2  <svg xmlns="http://www.w3.org/2000/svg" width="100" height="100"
       viewBox="0 0 100 100">
3    <circle cx="50" cy="58" r="39" stroke="#777" fill="#f60" stroke
       -width="2"/>
4    <ellipse cx="50" cy="18" rx="23" ry="17" stroke="#777" fill="#
       fff" stroke-width="2" />
5    <ellipse cx="50" cy="32" rx="23" ry="19" fill="#000"/>
6    <ellipse cx="32" cy="25" rx="3" ry="5" fill="#777" />
7    <ellipse cx="68" cy="25" rx="3" ry="5" fill="#777" />
8    <polygon points="36 29, 50 25, 64 29, 50 31" fill="#777" />
9    <line x1="30" y1="91" x2="24" y2="99" stroke="#777" stroke-
       width="2" />  <line x1="70" y1="91" x2="76" y2="99" stroke
       ="#777" stroke-width="2" />
10   <line x1="17" y1="99" x2="29" y2="92" stroke="#777" stroke-
       width="2" />  <line x1="30" y1="99" x2="29" y2="92" stroke
       ="#777" stroke-width="2" />  <line x1="83" y1="99" x2="71"
       y2="92" stroke="#777" stroke-width="2" />  <line x1="70" y1
       ="99" x2="71" y2="92" stroke="#777" stroke-width="2" />
11 </svg>
```

ソースコード 5.1　SVG の例（redstart.svg をテキストとして開いた結果）

図 5.1　SVG の例（redstart.svg を画像として開いた結果）

実行環境は Windows11 で、特に新たな環境構築も必要なく、以下の 2 手順だけです。

1. テキストエディタ（メモ帳）で SVG のソースコードを書き、保存した。
2. 作成した SVG ファイルをブラウザ（Chrome など）で開いた（画像を確認した）。

表示を拡大縮小してもラスター画像のようなジャギーは現れないことを確認しました。最初に試したのはコピペのコードだったので、図 5.1 の鳥（ジョウビタキ）のベクター画像こそが自分にとって初めて SVG で書いた画像です。楕円と多角形と直線を使って結構いい感じにできました（自画自賛）。コードを書きながら図形を配置したり微調節したりするのは楽しかったです。しかし実際に使っていくとなるとやはり Inkscape や Adobe Illustrator などのソフトウェアを使うのが現実的かなと実感できました。

次に、書いた SVG 画像を LaTeX 文書にとりこみました。実行環境は WSL Ubuntu

24.04 と Inkscape 1.2.2 、TEXLive 2023 でコンパイルと PDF の生成は pLATEX と dvipdfmx で実行しました。TEXWiki[3] によると、SVG は dvipdfmx がサポートしていない形式であるため、Inkscape を用いて SVG を PDF に変換してとりこみました。

1. Inkscape をインストールした。`sudo apt install inkscape`
2. 作成済みの SGV 画像を PDF 画像に変換した。
   `inkscape redstart.svg --export-filename=redstart.pdf`
3. LATEX の文書を作成した。graphicx パッケージを用いて画像をとりこむときと同様にして PDF 画像をとりこむように書いた。
4. LATEX 文書をコンパイルして、PDF の文書を生成した。

PDF に変換した画像が文書にとりこまれていることを確認できました。色々なサイズでとりこんでみてもきれいな線で表示されたのはとても感動しました。

5.4 おわりに

1 年生の C 言語勉強会のときに初めて `Hello, World!` を出力したときの喜びと同じような喜びを味わいました。さらに調べる中で、SVG を使ってできることやその周辺の技術など、面白そうなことがたくさん見つかり、とてもわくわくしています。Android アプリ開発をするときの表現の幅が広がりそうですし、他の場面でも使いたいです。

最後に、自分にとってコンピュータ部に入って良かったなと思うことの一つは、やったこと無いものや知らないものに色々と手を出しやすくなった、出すようになったことです。日々の活動の中で少しでもできることが増える喜びを伝えられたなら幸いです。

参考文献

[1] 杜甫々「とほほの SVG 入門」とほほの WWW 入門, 最終更新日 2018 年 1 月 28 日, 最終閲覧日 2025 年 10 月 18 日, https://www.tohoho-web.com/ex/svg.html

[2] 碓井「SVG をソースコードだけで書く方法」大塚ビジネスサービス, 最終更新日 2022 年 9 月 8 日, 最終閲覧日 2025 年 10 月 18 日, https://www.otsuka-bs.co.jp/web-creation/blog/archive/20220908-01.html

[3]「dvipdfmx/画像のとりこみ」TEXWiki, 最終更新日 2024 年 9 月 22 日, 最終閲覧日 2025 年 10 月 18 日, https://texwiki.texjp.org/?dvipdfmx%2F%E7%94%BB%E5%83%8F%E3%81%AE%E3%81%A8%E3%82%8A%E3%81%93%E3%81%BF

編集後記

今回編集を担当しました樋口です。ここまで目を通していただきありがとうございました。 前号に続き、2 回目の編集となりました。 前回の経験を踏まえ、先輩方が残してくださった編集データを参考にしながら、編集を進めてまいりました。 執筆者の皆さんのご協力のおかげで、今回も無事に部誌を完成させることができました。心より感謝申し上げます。

令和 7 年 11 月 6 日
編集担当　樋口 舞子

Lime Vol. 65
令和 7 年 11 月 11 日発行 第 1 刷

発行 京都工芸繊維大学コンピュータ部
Web サイト: http://www.kitcc.org/
電子メール: question@kitcc.org