

KSS PC BOOK 2022

2021-05-xx 版 KSS PC Club 発行

目次

はじめに	4
この部誌を支える技術	5
ICPCへのお誘い	10
ワンクリック詐欺サイト解剖してみた	17
ニュートン法で近似をしよう！！	23
キーボードが欲しかったので自作してみた	35

プログラミング言語 Zig の基本文法

50

始めよう、Arch Linux

61

はじめに

このたびは本書をお手に取っていただきありがとうございます。本書は令和4年度けやき祭のためにパソコン部の有志によって作成された部誌です。

古河中等教育学校パソコン部(通称:KSS PC Club)では部員それぞれが主に好きなことをし、競技プログラミングやWeb開発、ゲーム開発など様々なことに取り組んでいますが、本書では部員の興味を持っていることや語りたいことなどについて記事を自由に執筆してもらい、一冊の本に仕上げました。

新型コロナウィルスの影響により、文化祭でパソコン部が作品等を展示するのは2019年以来の3年ぶりであり、久しぶりの作品展示の機会であるのでぜひ特別棟2階奥のパソコン室にもお立ち寄りください。

部誌の制作については今年度からの初の試みであり、自分の言葉で物事を発信する場を設けるという意義があります。このような発信活動の場のひとつである本書を通して、読者のみなさまにも創作・技術に触れる楽しさ、好きなことに接する楽しさを感じてもらえると幸いです。

部長 細島涼雅/Ryoga.exe

お問い合わせ先

本書に関するお問い合わせは部員まで。

免責事項

本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた開発、製作、運用は、必ずご自身の責任及び判断の上で行ってください。これらの情報による開発等の結果について、著者はいかなる責任も負いません。

この部誌を支える技術

Ryoga.exe

はじめに

こんにちは、5期生の Ryoga.exe です。この部誌は今年からの初の試みということもあり折角なので(?)今回はこの部誌を支える技術についてお話ししようかと思います。この部誌については GitHub 上で管理されているため、そのリポジトリも合わせてご覧ください。

[1]

Vivliostyleについて

本書は HTML & CSS で組版ができる Vivliostyle [2] の Create Book [3] というものを使って書かれています。



Vivliostyle

▲図1: Vivliostyle のロゴ

Vivliostyle は CSS 組版という Web 標準技術をベースにした、自動組版システムのオープンソースプロジェクトです。

CSS組版はHTMLとCSSをベースにしているので、普段からHTML/CSSを扱っている人にとってはVivliostyleのプロダクトが比較的手軽に感じるでしょう。

[1] <https://github.com/kss-pc-club/book-2022>

[2] <https://vivliostyle.org/ja/>

[3] <https://docs.vivliostyle.org/#/ja/create-book>

Vivliostyle を選択した理由

まず、複数人で書くことが前提となっていたので原稿等を共有する必要があります。また、見た目などもある程度統一させたいです。そのため、Word 等のソフトウェアは真っ先に候補から外れました。

そして、本を書く上で新たに多くの記法などを覚えるのは大変かつ、脳のリソースの無駄になります。そこで、Markdown で書いて、スタイルをコードで整えることができ、GitHub 上で管理できるといったことが必要でした。

そのため、Vivliostyle を選択しました。(日本語のドキュメントが充実しているという理由もありました。)

VFMについて

Create Book がサポートする Markdown 方言は、Vivliostyle Flavored Markdown (VFM) です。詳細は公式ドキュメント [4] を参照してください。

今回はこの VFM でサポートされている記法について、ざっくりとご紹介します。

```
---
title: セロ弾きのゴーシュ
---

# セロ弾きのゴーシュ

**宮沢賢治**[^1]

ゴーシュは町の活動写真館でセロを弾く係りでした。  

けれどもあんまり上手でないという評判でした。  

上手でないどころではなく実は仲間の楽手のなかではいちばん下手でしたから、  

いつでも楽長にいじめられるのでした。  

ひるすぎみんなは楽屋に円くならんで今度の町の音楽会へ出す  

第六{交響曲|こうきょうきょく}の練習をしていました。

![挿絵]('image/fig334.png') {width=300}

[^1]:  

https://ja.wikipedia.org/wiki/%E5%AE%AE%E6%B2%A2%E8%B3%A2%E6%B2
```

[4] <https://vivliostyle.github.io/vfm/#/ja/vfm>

基本的には GitHub でサポートされている Markdown の方言である GFM (GitHub Flavored Markdown) の記法は使えます。しかし、上のサンプルにはなにか見慣れない記法がありますね。それぞれ説明します。

ルビ

べんり ミライ
 {親文字|ヨミ} とすることでルビが振れます。便利。~~梅やもと書いてミライと読ませることもできますね！~~

脚注(後注)

文章内に [^n] を置きその文の後の箇所で [^n]: hoge とすると脚注をつけることができます。ちなみに、@vivliostyle/theme-techbook のテーマパッケージを使用している場合は hoge とすると本文(ページ)の下に注記をつけることができます。

余談ですが、脚注をつける記法は実は GFM で同様にサポートされていました。

Sass を使う

この節では Sass と一緒にビルド/プレビューする方法をご紹介します。CSS を使ってスタイルを当てるのですが CSS よりも Sass の方がいろいろと楽だったりします。

Create Book でプレビューするとき、直接 .scss ファイルを読み込めないため、スタイルを調整しながらプレビューするといったことが困難です。

これを解決するために、npm-run-all を使います。run-p というコマンドがあり、パラレル実行ができるためこれを活用します。npm や yarn でインストールして

```
$ yarn add --dev npm-run-all
```

package.json の script に以下を追加します。

```
"start": "run-p preview watch:scss",
"preview": "vivliostyle preview",
"watch:scss": "sass --watch scss:css"
```

これで yarn start とすると SCSS のビルドとプレビューが同時にできます。便利。

GitHub Actions でビルドする

この部誌は GitHub 上で管理されており、各部員がブランチを切り、PR を出す...といった感じで書かれています。

しかし、現状どのような見た目になっているかを確認できない部員もいたりします。そのため、GitHub Actions を活用してビルドし、自動で `publish` ブランチに pushされるような仕組みにすることにしました。以下のワークフローを作成することができます。

`.github/workflows/build.yml`

```
name: Build
on:
  workflow_dispatch:

jobs:
  build:
    name: Build
    runs-on: ubuntu-latest
    env:
      TZ: Asia/Tokyo
      GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
    steps:
      - name: Checkout
        uses: actions/checkout@v2.3.4
        with:
          ref: main
      - name: Setup Node
        uses: actions/setup-node@v2.1.5
        with:
          node-version: '14.16.0'
      - name: Get yarn cache directory path
        id: yarn-cache-dir-path
        run: echo "::set-output name=dir::$(yarn cache dir)"
      - name: Cache deps
        uses: actions/cache@v2.1.6
        with:
          path: ${{ steps.yarn-cache-dir-path.outputs.dir }}
          key: ${{ runner.os }}-yarn-${{ hashFiles('yarn.lock') }}
        restore-keys:
          - ${{ runner.os }}-yarn-
      - name: Install deps
        run: yarn install --frozen-lockfile
      - name: Install ghostscript
        run:
          sudo apt-get -yqq install libgbm1 ghostscript
```

```
        sudo apt install poppler-utils poppler-data
- name: Build
  run: yarn build
- name: Build Press-Ready
  run: yarn press-ready
- name: Deploy
  uses: s0/git-publish-subdir-action@develop
  env:
    REPO: self
    BRANCH: publish
    FOLDER: public
    GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

`workflow_dispatch` ではなく `main` ブランチ等への `push` をトリガーにしたほうが楽かもしれないですね。

この部誌をビルドする

前述したとおりこの部誌は GitHub のリポジトリで管理されているのでそれを `clone` してきましょう。

```
$ git clone git@github.com:kss-pc-club/book-2022.git
```

依存関係をインストールし、`yarn build` とするとビルドができます。

```
$ yarn install
$ yarn build
```

`public/book.pdf` が作られます。

入稿用のデータを `yarn press-ready` というコマンドで作ることができますが、`ghostscript` と `poppler-utils` のインストールが必要なので注意してください。

おわりに

ある程度しっかりとした本を既に知っている技術で作れました。

CSS 組版で本を作りたい！ Markdown で本を書きたい！...そんな方はぜひ Vivliostyle で始めてみませんか？

ICPCへのお誘い

Asa

はじめに

こんにちは、第3期生の土屋です。ネットでは「Asa (@a01sa01to)」として活動しています [1]。2021年3月に古河中等教育学校を卒業し、現在は埼玉大学工学部情報工学科に所属しています。この度、細島部長にお声がけいただき、このような形で部誌の編集に携わることとなりました。

技術系（特にプログラミング関連）の記事がほとんどである、この部誌を読んでくださっている方の中には、「競技プログラミングについて聞いたことがある / すでに参加している」という方がいると思います。そこで、国際大学対抗プログラミングコンテスト（ICPC）に関する記事を書きたいと思います。

ICPCってなに？

国際大学対抗プログラミングコンテスト（International Collegiate Programming Contest・通称 ICPC）[2] は、同じ大学の3人で1チームを作り、プログラミングの問題を解く大会です。中学生・高校生の皆さん、「国際情報オリンピックのチーム参加版」と捉えていただけるとわかりやすいかもしれません。



▲図1: ICPCのロゴ (<http://icpc.foundation/> より引用)

[1] ここまで情報で私のフルネームと誕生日がわかるらしいですよ

[2] <https://icpc.global/> (日本語版: <https://icpc.iisf.or.jp/>)

この大会には世界各国から毎年3万人以上が参加しており、日本からも様々な大学から参加しています。2021年度、私の所属する埼玉大学からも2チーム出場し、私もそのうちの1チームに参加していました [3]。

大会の流れとしては、「国内予選→アジア地区大会→世界大会」といった流れです。情オリに似ていますね。国内予選では3時間で6-7問、アジア地区大会では5時間で10問程度の問題を解きます。情オリとは違い、得点ではなく、解いた問題数で競います。また、アジア地区大会で利用できるコンピュータもチームで1台と制限があります [4]。そのため、1人が解いている間にほかの人がアイデアを出すなどといったチームワークが重要になります。さらに知りたい方は、ICPC日本公式団体に掲載されている「3分でわかるICPC」(<https://icpc.iisf.or.jp/acm-icpc/3min/>)も併せてご覧ください。



▲図2: 2019年度アジア地区大会の様子 (<https://youtu.be/MwD254vH3Pw>)

どんな問題を解くの？

「○○くらいの難易度です！」と言っても個人差があると思うので、実際に出された問題を見てみましょう。

[3] ちなみに私のチームの結果は、予選43位、アジア地区大会11位でした。もう1チームは予選104位で、アジア地区大会に進出できませんでした。

[4] 2020年度・2021年度はオンラインだったため、1人1台でした。なお国内予選では、コードではなく、自分のコンピュータで出た答えを提出します。~~時間制限は実質無限です。~~

2021年度 国内予選 A問題

https://icpc.iisf.or.jp/past-icpc/domestic2021/contest/all_ja.html#section_A より、問題文の概要を書いてみます。

4つのお椀とその中に入ったビー玉がある。1つもビー玉が入っていないお椀があることがあるが、少なくともどれか1つのお椀にはビー玉が入っていることが保証されている。

4つのお椀に対して、空でないお椀が1つだけになるまで、以下の操作を繰り返す。

1. 空でないお椀のうち、ビー玉が少ないものを選ぶ。2つ以上あるときは、一番左のお椀を選ぶ。
2. 選んだお椀以外の、空でないすべてのお椀から、選んだお椀と同じ数のビー玉を取り除く。ただし、選んだお椀のビー玉はそのままにしておく。

さて、最終的にお椀に残ったビー玉の個数はいくつ？

制約

- それぞれのビー玉の個数は、0個以上100個以下。
- 少なくとも1つは0ではない。
- データセットは100個以内。
- 0 0 0 0 は終了の合団。

サンプルの入出力を見てみましょう。

```
10 8 4 6
0 21 7 35
5 45 13 3
52 13 91 78
0 0 0 0  (←入力終了の合団)
```

```
2
7
1
13
```

さて、この問題をあなたならどう解きますか？

まず思いつくのは、単純にシミュレーションしてみることです。

具体的には、ビー玉の残ったお椀が1以下ではない限り、次の手順を繰り返します。

1. お椀を、ビー玉の個数でソートする
2. 一番少ないお椀を選び、そのビー玉の個数を他から減らす
3. ビー玉が0以下になったお椀を削除する

このような方法でも、計算量は高々 $O(T \sum a_i)$ (10^4 くらい) なので、高速です [5]。なお、データセット数を T としています。

ans.cpp

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    while (true) {
        vector<int> a(4);
        for (int i = 0; i < 4; ++i) cin >> a[i];

        if (a[0] + a[1] + a[2] + a[3] == 0) break;

        while (a.size() > 1) {
            sort(a.begin(), a.end());

            for (int i = 1; i < a.size(); ++i) a[i] -= a[0];

            auto itr = remove_if(a.begin(), a.end(), [] (int x) {
                return x <= 0;
            });
            a.erase(itr, a.end());
        }
        cout << a[0] << endl;
    }
    return 0;
}
```

ICPCでは、このような単純な繰り返し処理や条件分岐などができるれば、チームに貢献できます。実際、当時 AtCoder 茶色の私でも、解くことができました。

参考までに、さらに高速な方法をご紹介します。それは、最大公約数を用いる方法です。

[5] 各手順で少なくとも1つは取り除かれるので、最大でビー玉の個数分の手順しか行われません。

実は、それぞれのデータセットの答えは、 a_1, \dots, a_4 の最大公約数になることが証明できます [6]。

これを用いると、 $O(T \log(\min a_i))$ で計算できます。

ans.cpp

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    while (true) {
        int a, b, c, d;
        cin >> a >> b >> c >> d;

        if (a + b + c + d == 0) break;

        int g = a;
        g = gcd(g, b); g = gcd(g, c); g = gcd(g, d);

        cout << g << endl;
    }
    return 0;
}
```

2020年度 アジア地区大会 A問題

本当は2021年度の問題を載せようと思ったのですが、まだ公開されていませんでした...。
(何かの体積を計算させる問題だったと記憶しています)

そこで、2020年度の問題を載せます！ [7]

$N \times N \times N$ の立方体に収まるある立体を、 $x - y, y - z, z - x$ 平面それぞれに射影した図が与えられます。与えられた図に適する図形が存在するか判定してください。

制約: $1 \leq N \leq 100$

[6] <https://icpc.iisf.or.jp/past-icpc/domestic2021/commentaries.html#A>

[7] <https://icpc.iisf.or.jp/past-icpc/regional2020/problems-2020.pdf> すべて英語です。
本番では、機械翻訳は許されません（辞書はOK）。

解法は、 $N \times N \times N$ すべてが埋まった立方体から、「削って」いき、最終的に出来上がった立体が、条件が満たしているかを確認する方法です。

アジア地区大会では、自分のコンピュータではなく、ジャッジ用コンピュータで正誤判定されるため、時間制限が定められています。この問題では2秒ですが、以上の解法は $O(N^3)$ なので、ACになります。

実装は大変ですが、これも問題なく解けると思います。

ans.cpp

```
#include <bits/stdc++.h>
using namespace std;
#define rep(i, n) for (int i = 0; i < (n); ++i)

int main() {
    int n;
    cin >> n;
    vector<vector<bool>> yz(n, vector<bool>(n)), zx(n, vector<bool>(n)),
    xy(n, vector<bool>(n));
    rep(i, n) {
        string s; cin >> s;
        rep(j, n) yz[j][n - i - 1] = (s[j] == '1');
    }
    rep(i, n) {
        string s; cin >> s;
        rep(j, n) zx[j][n - i - 1] = (s[j] == '1');
    }
    rep(i, n) {
        string s; cin >> s;
        rep(j, n) xy[j][n - i - 1] = (s[j] == '1');
    }

    vector<vector<vector<bool>>> ans(n, vector<vector<bool>>(n, vector<bool>(n, true)));
    rep(i, n) rep(j, n) if (!yz[i][j]) rep(k, n) ans[k][i][j] = false;
    rep(i, n) rep(j, n) if (!zx[i][j]) rep(k, n) ans[j][k][i] = false;
    rep(i, n) rep(j, n) if (!xy[i][j]) rep(k, n) ans[i][j][k] = false;

    rep(i, n) rep(j, n) {
        if (yz[i][j]) {
            bool chk = false;
            rep(k, n) if (ans[k][i][j]) chk = true;
            if (!chk) { puts("No"); return 0; }
        }
    }
    rep(i, n) rep(j, n) {
```

```
if (zx[i][j]) {
    bool chk = false;
    rep(k, n) if (ans[j][k][i]) chk = true;
    if (!chk) { puts("No"); return 0; }
}
rep(i, n) rep(j, n) {
    if (xy[i][j]) {
        bool chk = false;
        rep(k, n) if (ans[i][j][k]) chk = true;
        if (!chk) { puts("No"); return 0; }
    }
}
puts("Yes");
return 0;
}
```

最後に

ICPCの問題を見てみましたが、「こんなのが解けるわけがない！」というものではなかったと思います。むしろ、少し考えたうえで、プログラミングの初步である「繰り返し処理」「条件分岐」を使いこなすだけで、最初の問題は解けます。

競技プログラミングをするうえで、ICPCは一生に5回程度しか参加できないため、貴重な体験になることは間違いないです。そしてICPCは、プログラミング力ではなく、チームワークが鍵です。チーム戦で生まれる連帯感というのは、なかなか体験できないと思います。

この記事を通じて、少しでも興味を持っていただけたのであれば幸いです。大学に入学した際には、ぜひともICPCに参加してみてください！！！ [8]

最後までお読みいただき、ありがとうございました！

[8] 参加資格のない方は...順位表実況や過去問などでも楽しめます！

ワンクリック詐欺サイト解剖してみた

張替健太/hnm876_md

はじめに

こんにちは。古河中等教育学校6期生の張替健太です。僕の記事の前に書かれた小難しそうな記事を見て、「プログラミング難しそうだな」とか「変態の領域なのかな」感じている人がいるかもしれません。実際そうですが、Webプログラミングは意外と簡単です。どれほど簡単なのかWebプログラミングの基本が詰まった教科書「ワンクリック詐欺サイト」を見ながら体感していただければ幸いです。

ワンクリック詐欺サイトってなあに

その名の通り、一見ありふれたボタンをクリックした途端に「会員登録完了」や「〇〇日までに料金を払ってね」などの脅しがたくさん出力されるWebサイトのことです。2018年時点では存在していましたが、今もあるんでしょうか。もし引っかかっても、僕のように電話はかけずに無視してブラウザバックしましょう。

ワンクリック詐欺サイト解剖

支払い期限タイマーの表示



▲図1: ハッタリタイマー

こんな感じでクリックした人を焦らせて個人情報入力させようという魂胆です。冷静に見てみるとおもちゃのような仕組みです。

index.html

```
<div>
  <p><span id="hour"></span>hours</p>
  <p><span id="min"></span>minutes</p>
  <p><span id="sec"></span>seconds</p>
</div>
```

timer.js

```
const hour = document.getElementById("hour");
const min = document.getElementById("min");
const sec = document.getElementById("sec");

function countdown(){
  const now = new Date(); //現在時刻を取得
  const tomorrow = new
Date(now.getFullYear(),now.getMonth(),now.getDate()+1); //明日
の0:00を取得
```

```

const diff = tomorrow.getTime() - now.getTime(); //時間の差を
取得（ミリ秒）

//ミリ秒から単位を修正
const calcHour = Math.floor(diff / 1000 / 60 / 60);
const calcMin = Math.floor(diff / 1000 / 60) % 60;
const calcSec = Math.floor(diff / 1000) % 60;

//取得した時間を表示（2桁表示）
hour.innerHTML = calcHour < 10 ? '0' + calcHour : calcHour;
min.innerHTML = calcMin < 10 ? '0' + calcMin : calcMin;
sec.innerHTML = calcSec < 10 ? '0' + calcSec : calcSec;
}

countdown();
setInterval(countdown, 1000);

```

1. HTMLで書かれたファイルのhour,min,secをJavaScriptという言語で取得する
 2. 現在の時刻と明日の0時を取得して差分を出して取得する
 3. modの考え方を利用して時間、分、秒を算出する
 4. 見やすく二桁表示にして、カウントダウンする
- 大体こんな手順です。あんまり怖くないですね。

機密(?)情報の表示

情報が既に抜き取られていて逃げられない…なんてことはありません。端末情報やプラウザのバージョンなんかは開示されているので、誰でも簡単に抜き取ることができます。

kowakunaiyo~.js

```

<script src="platform.js"></script>
<script>
    document.write(platform.name); //Firefox
    document.write(platform.version); //69.0
    document.write(platform.os.toString()); //OS X 10.14
    document.write(platform.layout); //Gecko
</script>

```

上部のように書くと、こんな感じに抜き取ることができます。

会員登録完了

お客様情報

ご利用環境 : Chrome101.0.4951.54OS X 10.15.7 64-bitBlink

▲ 図2: 抜き取り使用例

```
// on IE10 x86 platform preview running in IE7 compatibility
// mode on Windows 7 64 bit edition
platform.name; // 'IE'
platform.version; // '10.0'
platform.layout; // 'Trident'
platform.os; // 'Windows Server 2008 R2 / 7 x64'
platform.description; // 'IE 10.0 x86 (platform preview;
running in IE 7 mode) on Windows Server 2008 R2 / 7 x64'

// or on an iPad
platform.name; // 'Safari'
platform.version; // '5.1'
platform.product; // 'iPad'
platform.manufacturer; // 'Apple'
platform.layout; // 'WebKit'
platform.os; // 'iOS 5.0'
platform.description; // 'Safari 5.1 on Apple iPad (iOS 5.0)'

// or parsing a given UA string
var info = platform.parse('Mozilla/5.0 (Macintosh; Intel Mac
OS X 10.7.2; en; rv:2.0) Gecko/20100101 Firefox/4.0 Opera
11.52');
info.name; // 'Opera'
info.version; // '11.52'
info.layout; // 'Presto'
info.os; // 'Mac OS X 10.7.2'
info.description; // 'Opera 11.52 (identifying as Firefox
4.0) on Mac OS X 10.7.2'
```

上部は抜き取れる情報と抜き取り方みたいなことが書かれています。重要なのはこの内容ではなく、いろんな情報が誰でも抜き取れるんだなあということです。このようなライブラリ (<https://github.com/bestiejs/platform.js>) という誰でも使えるものがあるのでハッキングとかではないんだなと思っておくと気が楽です。

写真を撮ったかのような演出

謎のシャッター音と点滅で「もしかして写真を撮られたのかも」と思うかもしれません
がそんなことありません。演出です。

保存処理中です。処理中に削除せずお待ちください。

次へ



写真を撮影しました

[写真を確認する、もしくは削除する](#)

▲図3: パシャ

今ではWebポリシー改定によってシャッター音を急に鳴らすことはかなり難しくなって
います。できないことはありませんが、実装するためのSPA(シングルページアプリケーシ
ョン)開発者・経験者が少ないらしいので、こちらの解剖はせず、点滅の方の解剖をしてい
こうと思います。

camera.html

```
<div class="camera">
    
using namespace std;

int main() {
}

```

②変数の宣言

今回扱う数字は実数なので、実数の変数を定義し、初期値($=x_1$)としておきます。

```
double ans = 20;
```

これは $x_1 = 20$ としています。また、この初期値は求めたい数より大きければどの数でもかまいません。(自然数に限らず、実数でもOKです。)

③繰り返し

{ }で囲まれている部分を10000回繰り返します。

```
for (int i = 0; i < 10000; i++) {  
}
```

ちなみに下の場合は10回の繰り返しとなります。

```
for (int i = 0; i < 10; i++) {  
}
```

④ニュートン法の計算部分

初めに、ほとんどのプログラミング言語における「=」は等式を意味しているものではなく、代入を意味します。例えば `a = 20` を例にとって考えてみます。数学では `a` と `20` は同じ値であるということを意味していますが、プログラミング言語では「変数 `a` に `20` を代入する」ということを意味します。また、掛け算の記号は `*` ではなく `*` を、割り算の記号は `/` ではなく `/` を用います。

このプログラムでは、`ans` の値を更新していき、値を求めます。

⑤表示

10000回計算が終わった後、`ans`の値を小数第15位まで表示して改行します。

```
printf("%.15f\n", ans);
```

結果

前節で書いたプログラムの初期値を変えることによって値がきちんと求めることができると、収束のスピードを見ていきたいと思います。

また、収束したとは正確な値との差が非常に小さくなった時とします。しかし、その差の基準はいったいどうすれば良いでしょうか？

まず、 x_k は常に求めたい数より大きくなる、また n が増加するにつれて x_k は小さくなつて求めたい数に近づく、という二つの事象から次の x_{k+1} との差が非常に小さいとき、ほとんど近似できた、すなわち、 $(x_k - x_{k+1}) < \epsilon$ (k :自然数、 ϵ :計算精度を決める非常に小さな定数) を満たしたら近似できたということにしましょう。

ここで、1回の計算にかかる時間はほぼ同じと言えるので、上記の条件式を満たすようになるまでに計算した回数が小さいほど収束が早いといえます。

近似できたのが何回目の操作であるかも表示するプログラムの一例を以下に示します。

```
// コード(cpp)
#include <bits/stdc++.h> // おまじない
using namespace std;

// プログラムのはじまり
int main() {

    double ans = 20; // x1の値
    int i;

    // 十分な回数(10000回)繰り返す
    for(i = 0; i < 10000; i++){
        double tmp = ans;

        // ニュートン法の式の通りに値を更新
        ans = ans -((ans*ans) - 2)/(2*ans);

        // 差が0.0000001になったら計算終了
        if(tmp - ans <= 0.0000001) break;
    }

    // ansの値と誤差が小さくなった時のiを表示
    printf("%.15f\n%d\n",ans,i);

} // プログラムの終わり
```

このプログラムを用いることで、計算にかかった回数 = 速度が分かります。

x1の値	計算した回数
2	4
20	7
200	11
2000	14
20000	17
200000	21

初期値が求める値に近いほど収束するスピードが早いことが分かります。まあ、当たり前と言えばあたりまえなのですが。

例えば、ある初期値 x_1 によって得られる x_2 が別の初期値 x'_1 と同じになった時、単純に計算にかかる時間は1回分短くなります。

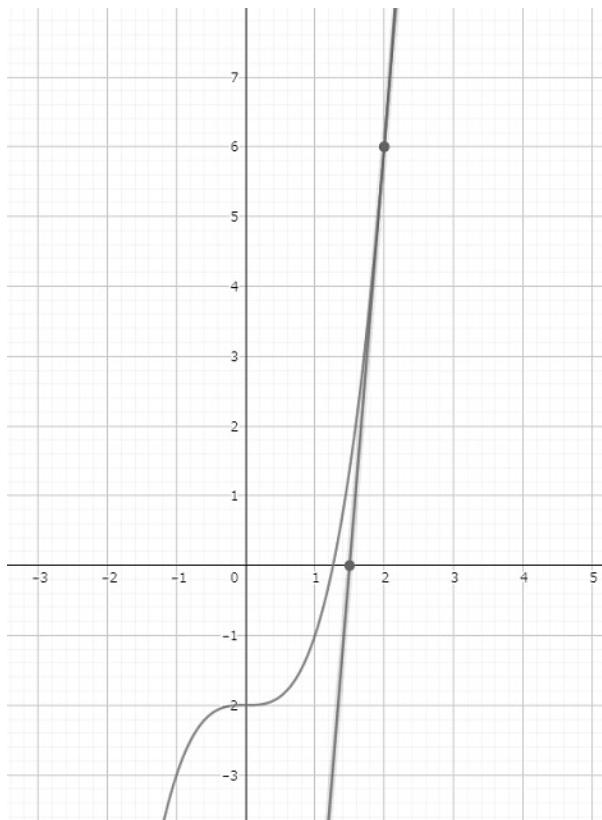
応用方法

実は、このニュートン法は $\sqrt{2}$ の時だけでなく、 \sqrt{n} (n は任意の正の実数)のときにも成り立ちます。

さらにいうと、これは2乗根の時だけではなく1.5乗根、3乗根、4乗根、... m 乗根(m は任意の正の実数)の時にもすべて成り立ちます。

これは、最初に求めた式 $x_{n+1} = x_n - f(x_n)/f'(x_{n+1})$ をもとに、2次式が成り立つことが分かります。

この式の $f(x)$ の次数を変えても、(例えば $f(x) = x^3 - 2$ のときでも)成り立ちます。



▲図3: 3次関数の場合の様子

したがって、 $\sqrt[k]{m}$ (k, m はともに任意の正の実数) の値も同様の方法で近似することができます。ニュートン法の式 $x_{n+1} = x_n - f(x_n)/f'(x_n)$ を用いることは変わらないので $\sqrt[k]{c}$ を求める場合、 $f(x) = x^k - c$ において、少し変えると作ることができます。

また、前節では調べる数字を $\sqrt{2}$ のみに絞って考えているため変数が少なかったのですが、ここで「何乗根であるか？」と「根の中の数字」を表す変数を追加し、新たにコードを書き直します。

コードは以下の通りになります。

```
// コード(cpp)
#include <bits/stdc++.h> // おまじない
using namespace std;

// プログラムのはじまり
int main() {

    double ans = 20; // 初期値
    double k = 3;    // 何乗根か？
    double c = 2;    // 根の中の数字は何か？

    // 十分な回数(10000回)繰り返す
    for (int i = 0; i < 10000; i++){

        // ニュートン法の式の通りに値を更新
        ans = ans - (pow(ans,k)-c)/(k*pow(ans,k-1));
    }

    // 小数第15位までansの数字を表示する
    printf("%.15f\n",ans);
}

// プログラムの終わり
```

この時、 $f'(x) = kx^{k-1}$ であるから、累乗をプログラムで表現する必要が出てきます。そこで、 a^b を `pow(a, b)` という標準の関数で計算します。

次節では収束条件とその証明を書きます。

収束条件とその証明

収束条件： $f(x_0) > 0$ 、 $f(a) < 0$ 、 $f'(x) > 0$ 、 $f''(x) > 0$ の全てを満たす。

すなわち、 x についての閉区間 $[a, x_0]$ において $f(x)$ が下に凸な単調増加なグラフであり、 $f(x) = 0$ を満たす解が一つある状態です。

証明

x が区間 $[a, x_0]$ にある時、 $f''(x) > 0$ より

$$\int_x^{x_0} f''(x) dx > 0$$

が成り立つ。

計算し、式を変形すると $f'(x) < f'(x_0)$ したがって

$$\int_x^{x_0} f'(x) dx < \int_x^{x_0} f'(x_0) dx$$

が成り立つ。

計算すると

$$f(x_0) - f(x) < f'(x_0)(x_0 - x)$$

ここで $f(a) < 0 < f(x_0)$ が成り立つから中間値の定理より $f(\alpha) = 0$ を満たす α が区間 (a, x_0) に存在する。

また、関数 $f(x)$ は単調増加関数であるから、逆関数が存在する。(区間 $[a, x_0]$ において $f(k) = f(l)$ でかつ $k \neq l$ を満たす数が存在しないものを逆関数という。)

よって $f^{-1}(x)$ を $f(x)$ の逆関数とすると $\alpha = f^{-1}(0)$ と表せる。

ここで $x = \alpha$ とおくと

$$f(x_0) - f(\alpha) < f'(x_0)(x_0 - \alpha)$$

ここで $f(\alpha) = 0, f'(x) > 0, f(x_0) > 0$ より

$$\alpha < x_0 - \frac{f(x_0)}{f'(x_0)} < x_0$$

また、 $x_0 - f(x_0)/f'(x_0) = x_1$ より

$$\alpha < x_1 < x_0$$

これを繰り返すことによって $\alpha < \dots < x_3 < x_2 < x_1$

また、 x は単調減少であり下限があるため数列 $\{x_n\}$ は収束する。ここで数列 $\{x_n\}$ が β に収束するとする。

$x_{n+1} = x_n - f(x_n)/f'(x_n)$ より

$$\lim_{n \rightarrow \infty} x_{n+1} = \lim_{n \rightarrow \infty} \left(x_n - \frac{f(x_n)}{f'(x_n)} \right)$$

ここで $\lim_{n \rightarrow \infty} x_n = \beta, \lim_{n \rightarrow \infty} x_{n+1} = \beta$ であるから

$$\beta = \beta - \frac{f(\beta)}{f'(\beta)}$$

これより $f(\beta) = 0$

関数 $f(x)$ は逆関数を持つため $\beta = f^{-1}(0)$ よって

$$\beta = \alpha \quad (\because \alpha = f^{-1}(0))$$

したがって、数列 x_n は α に収束し、 $f(\alpha) = 0$ となるので

ニュートン法の数列 x_n は「 $f(x_0) > 0, f(a) < 0, f'(x) > 0, f''(x) > 0$ の全てを満たす」とき $f(x) = 0$ の解 α ($a < \alpha < x_0$) に収束する。 ■

おわりに

プログラミングと聞くと、難しくて初心者にはできないものだと考えてしまうかもしれません、この記事のように簡単な計算や、文字を表示させるなど、単純な動作だけなら今すぐにでも実装・実行できます。

それに、今の時代コンパイラはオンライン上のものを使うこともでき、インターネットで検索すれば実装方法も簡単に見つけることができますし、自分でアレンジすることができます。

ぜひ、皆さんも簡単なプログラムから初めて、自分の作りたいものを作りましょう！

参考文献等

1. ニュートン法 wikipedia

<https://ja.wikipedia.org/wiki/ニュートン法>

2. ニュートン法とは？～定義と性質～ - 理数アラカルト

<https://risalc.info/src/newtons-method-properties.html>

3. グラフ作成 Geogebra

<https://www.geogebra.org/calculator>

4. 今回使用したオンラインコンパイラ : paiza.io

<https://paiza.io/ja>

キーボードが欲しかったので自作してみた

Nagaso.cpp

はじめに

知っている人はここにちは。知らない人もここにちは。6期生のNagaso.cppです。今回は初めて自作キーボードに挑戦してみました。元々何かしらのハードウェアを自作してみたいなと思っていて、本当はPCを自作するつもりでしたが予算が足りなさ過ぎて断念しました。代わりに、PCと同じく欲しいと思っていたキーボードを自作することにしました。自作というものはなかなか面白いもので、今回はその楽しさを皆さんにもぜひ知ってもらいたく、ここに記すことにします。

必要なパーツについて

ここでは、今回購入したパーツを紹介します。パーツはすべて台東区にある遊舎工房という自作キーボード専門店の実店舗で購入しました。ホームページから通販で購入することもできます。 [1] パーツ選びにあたっては、ほぼすべて店員の方が手伝って下さいました。ありがとうございました。実店舗だと店員の方に相談すれば間違いないのがいいですね。おすすめです。

さて、キーボードは、**PCB基板**、**スタビライザー**、**キースイッチ**、**キークリップ**、**ケース**、**プレート**の6つのパーツから構成されます。それぞれ併せて解説します。

PCB基板

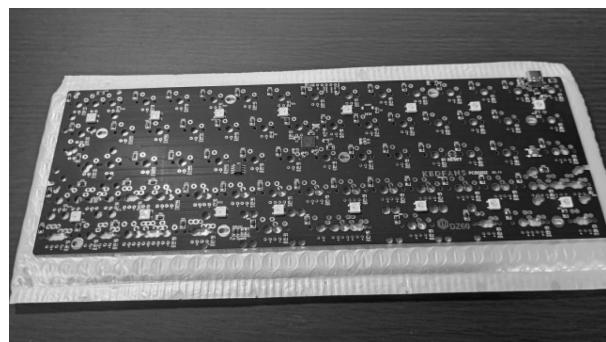
名前通り基盤です。これに後述のパーツを取り付けていきます。基盤は、日本で使われている日本語(JIS)配列用と、海外で使われている英語(US)配列用の二種類があります。今回はUS配列用のもので、普段私たちが使い慣れているものとは若干キーの配列が異なります。また、US配列の中にも配列の種類が二つあり、基板によって対応する配列が異なります。

[1] <https://shop.yushakobo.jp/>

さて、今回は **DZ60 REV 3.0 type-c** というものを購入。よくあるゲーミングキーボードのようにピカピカ光らせることができるタイプになります。また、この基盤は60%サイズとなっていて、テンキー・矢印キー、ファンクションキーなどがないコンパクトなタイプになります。ちなみに、設定をすることでこれらのキーは他のキーに自由に割り当てることができます。

商品ページはこちら↓

<https://shop.yushakobo.jp/products/dz60?variant=37665274855585>



▲図1: PCB基板

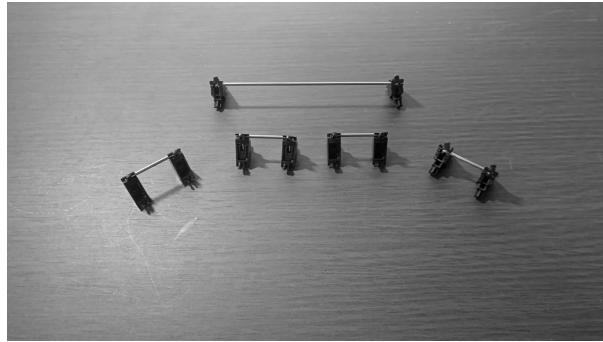
スタビライザー

このパーツは、PCB基板に取り付けることで、スペースキーなどの横に長いキーを支える役割をします。金属棒が使われているため、キーを押下したときに若干金属音が鳴る場合がありますが、高品質なものを使ったり、潤滑剤を塗ったりすることで改善できます。

私はお金がなくてケチりました。

商品ページはこちら↓

<https://shop.yushakobo.jp/collections/all-keyboard-parts/products/a0500st>



▲図2: スタビライザー

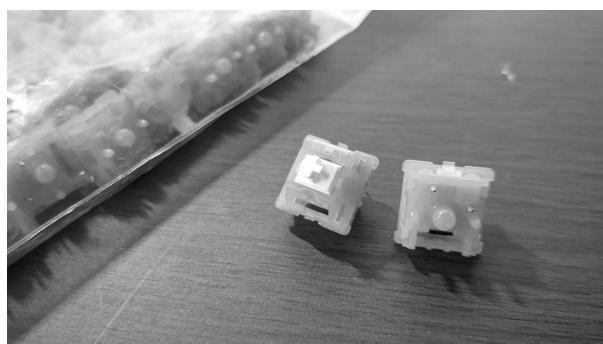
キースイッチ

各キーのスイッチ部分です。今回はメカニカルキーボードと呼ばれる種類のキーボードに使われるものになります。スイッチの種類によって真ん中のスイッチ部分（今回であれば白い部分）の色が異なり、打鍵感も様々なものがあります。Cherry社の「赤軸」「青軸」「茶軸」「黒軸」などが有名ですね。そのほかにも様々な会社が独自のキースイッチを開発・販売しており、その種類数はゆうに100を超えます。そのため、スイッチ選びは自作キーボードの醍醐味の一つと言えます。

今回はTecsee社の**Blue Sky Switch**をチョイス。Cherry社の茶軸に似たばねの重さと、青軸のようなクリッキー感を持ち合わせたもので、実店舗で様々な種類のスイッチを比べたうえで、この打鍵感がとても好きだったので選びました。

商品ページはこちら↓

<https://shop.yushakobo.jp/collections/all-switches/products/3973>



▲図3: キースイッチ

キーキャップ

普段私たちがタイピングをするときに、指に触れる部分です。キースイッチに取り付けます。自作キーボード用のキーキャップもまた様々なデザインがあり、このキーキャップ選びもまた自作キーボードの醍醐味のひとつです。塗装してオリジナルのデザインを作る人もいます。今回は、**PG PBT Macaron Keycaps set** をチョイス。結構おしゃれな感じのものを選びました。

商品はこちら↓

<https://shop.yushakobo.jp/collections/keycaps/products/pg-pbt-macaron-keycaps-set>



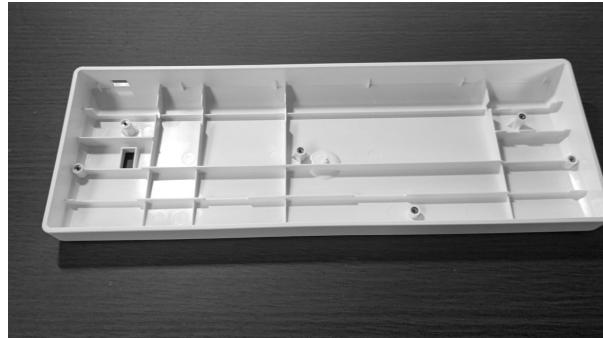
▲図4: キーキャップ

ケース

PCB基板にいろいろ取り付けた後、それをはめ込むケースです。材質はいろいろありますぐ、今回はケチって節約して安めのプラスチックのものにしました。基板の大きさにあったものを買わないと基盤がはまらなくて悲惨なことになるので気をつけましょう。今回の基盤は60%のものなので、60%用のを購入。

商品はこちら↓

<https://shop.yushakobo.jp/collections/case/products/60-plastic-case>



▲図5: ケース

プレート

キースイッチを固定・安定させるための部品です。



▲図6: プレート

また、今回の制作にあたっていくつか工具が必要なので、それも紹介します。

はんだごて一式

キースイッチを基板に取り付ける際、はんだづけが必要 [2] なため購入しました。セラミックヒーターで、かつ温度調整が可能なものが望ましいです。こて台が付属していない場合は、安全のため購入しておくことをお勧めします。

[2] はんだづけが不要なものもありますが、基盤が対応している必要があります。今回は非対応です。

はんたは、プリント基板用の直径0.8mmのものを購入しました。あとは、失敗したとき用に吸い取り線があると安心です。



▲図7: はんだごて

その他工具

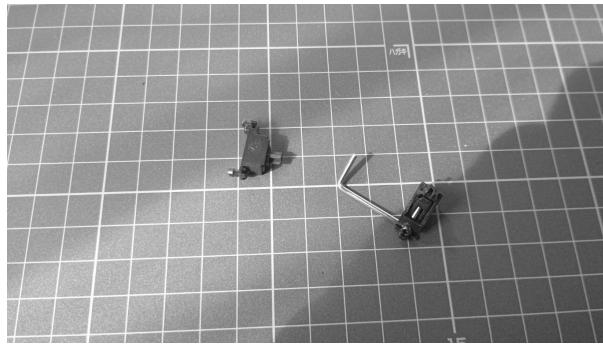
潤滑剤とカッターマットを購入。潤滑剤は筆で塗るタイプが好ましいです。また、ドライバーが基板のねじ止めの際に必要です。ドライバーは家にあったのでそれを使いました。

組み立て

さて、いよいよ組み立てに入ってきます。まず最初に、基板にいろいろ取り付けていきます。

スタビライザーに潤滑剤を塗る

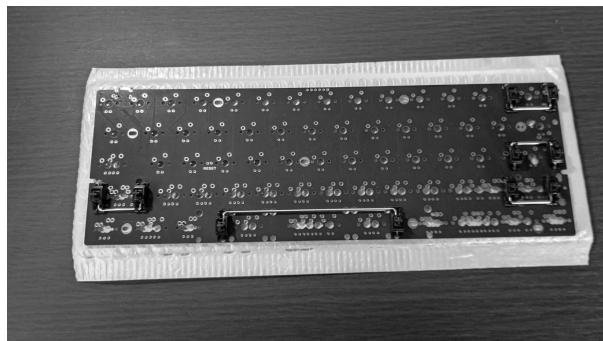
前述の通り、タイピング時に金属音が鳴るので軽減するため、一度スタビライザーを分解してから、金属部分に潤滑剤を塗っていきます。本当はキースイッチも分解して潤滑剤を塗るといいそうですが、キースイッチを分解する道具を買っていなかったので今回は塗っていません。~~一度潤滑剤を倒してしまいあわや大惨事になりかけたのは内緒~~



▲図8: スタビライザーの分解

スタビライザーを基板に取り付ける

スタビライザーのプラスチック部分の爪を、基板の小さな穴に押し込みます [3]。取り付ける位置は基板の取扱説明書などに書いてあるのでそれを参考にします。意外と力がいるので、基盤を折りそうになりますが、折らない程度に思い切りよく押し込むと上手くいきます。スタビライザーは、取り付け位置によって上下の向きが異なるので注意します。



▲図9: スタビライザー取り付け

プレートとキースイッチを基板に取り付ける

上下の向きに気を付けつつプレートを基板に取り付け、キースイッチを取り付けていきます。キースイッチの裏側にある5本の爪と基板の穴の位置が合うように、また取り付け位置は基板の取扱説明書やプレートの穴を参考にします。

[3] ねじ止めタイプの場合はねじ止めします。今回は爪タイプです。



▲図10: プレート取り付け



▲図11: スイッチ取り付け

キークリップを仮で取り付ける

特に横に長いキーは、スタビライザーやキースイッチの位置を間違えると隣のキーと干渉してキークリップがはまらない可能性があるため、一部を仮で取り付けてみて干渉しないか確認します。大丈夫そうなら、外して次の工程に移ります。



▲図12: キャップ仮設置

ケースイッチをはんだ付けする

基板の裏から出ている、各スイッチの2本の金属の爪にはんだ付けをしていきます。はんだ付けをしたのは前期生の技術の授業でやったとき以来なので、コツを掴むのに苦労しました。~~というかコツ掴んでない~~

こて先が熱すぎるとはなんだが蒸発？したので、適宜こて台のスポンジで冷やすようにするといい感じになります。



▲図13: はんだ付け

ケースにネジ止めして、キークリップを取り付ける

基盤を付属のねじを使ってケースにねじ止めして、各スイッチにキークリップを取り付けていきます。キークリップの穴の部分を、ケースイッチの先端の突起に押し込んでめます。日本語配列とは配列が異なるので、「USキーボード 60%」等でググって配列を確認

しながらはめていきます。スタビライザーとキースイッチの両方が下に来るスペースキー、Enterキーなどは、キースイッチとスタビライザー両方の突起がキークリップの穴に取り付けられるようにします。

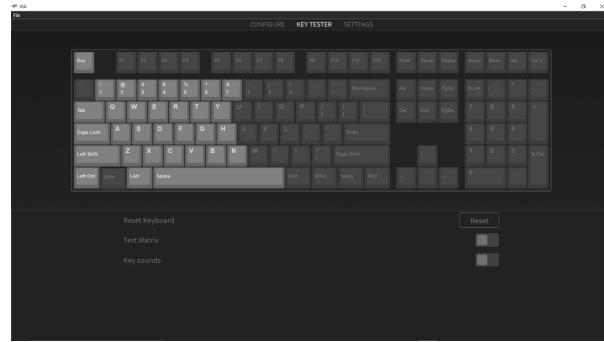


▲図14: キャップ取り付け

PCにつないで動作確認をする

今回は、キーボード側の端子がType-Cなので、それに対応したコードを使います。家に数本余っていたのでそれを使います。100均のを使うと接続が安定しなかったので、ちゃんとしたものを使っています。基板のLEDが光って、PC側で入力がしっかりとできていればとりあえずOKです。光っている様子は後ほど。

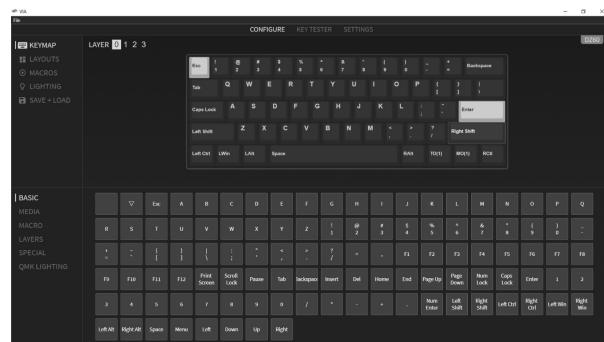
入力ができるかの確認は、ネットでもできるし、専用のソフトもあるのでそれを使うといいでしょう。今回はVIAと呼ばれるソフトを使いました。画面上のタブから**KEY TESTER**を選択し、任意のキーを押すと、認識されればそれに対応するキーが赤色に変わります。今回は、右シフトキーなどが矢印キーとして認識されていたため、その修正をしていきます。



▲図15: キーテスト

キーマップを変更する1

内部的に認識されているキーの配置（キーマップ）を変更し、実際の配置との齟齬をなくしていきます。先ほど使用したVIAで修正することができます。画面上のタブから**CONFIGURE**を選択します。画面上半分にあるキーボードのイラストから、変更したいキーを選択し、画面下半分のキー一覧から、そのキーに新たに対応させたいキーを選択すると、キーマップを書き換えることができます。



▲図16: キーマップ変更

キーマップを変更する2

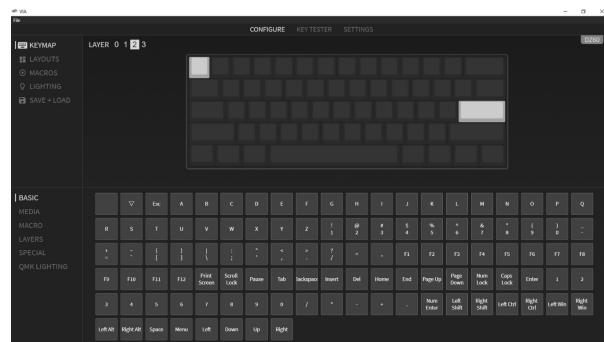
60%キーボードには、ファンクションキー（F1、F2...）や矢印キー等が存在しないため、このままではただの使いにくいキーボードになってしまいます。そこで、キーマップにレイヤーを追加して、これらのキーを使えるようにします。この作業が、自作キーボードの一番のミソだと思います。

レイヤーを編集する

画面左上の**LAYER**の数字を選択することでそのレイヤーの編集画面に移ることができます。先ほどの修正過程で書き換えたのはレイヤー0ですので、レイヤー1を選択します。すると、画面上半分のキーボードのイラストが、何も書いていないまっさらな状態になります（画像はレイヤー2）。これに、先ほどの修正過程と同じ手順でキーを割り当てていきます。今回割り当てたのは、左側の主要なキーと、矢印キー、記号キー、Print Screenキー、Deleteキー等です。

レイヤーを移動するキーを設定する

このままでは一つのレイヤーしか使えないのに、複数のレイヤーを移動できるようにキーを設定していきます。使用するのは**MO()**キーと**TO()**キーです。**()**の中身がレイヤーの番号と同じものを使います。MO(n)キーはそのキーを押している間n番目のレイヤーに移動し、TO(n)キーは一度押すとずっとn番目のレイヤーに移動したままになります。レイヤー0の右のMENUキーとFNキーは特に使わないのでこれに割り当てます。レイヤー0からレイヤー1に移動させたいのでどちらも**()**の中身が1であるものを使います。また、TO(1)を使用したときにレイヤー0に戻れるようにしたいので、レイヤー1にTO(0)を割り当てておきます。これでレイヤーを活用して、存在しないキーを使えるようにすることができました。



▲図17: まっさらなレイヤー



▲図18: レイヤー設定

完成!

これにて初の自作キーボード、完成となります。ちなみに、光り具合もVIAで調整することができます。

初めてのハードウェア自作でしたが、ほぼ何事もなく無事完成させることができてとても嬉しく思います。この記事も今回製作したキーボードで書きましたが、打鍵感も打音も最高で、大満足です。~~財布の中身は空っぽになりましたが~~。そしてなにより、ずっと挑戦してみたかった自作が実現したのが本当に夢のようで、とても楽しかったです。次はお金をためてPC自作にも挑戦してみようかな。

キーボードの自作は、突き詰めれば何も技術はいらず、お金さえあれば誰でも気軽に作れると思うので、皆さんもぜひ挑戦してみてください。



▲図19: 完成



▲図20: 完成2

余談

キーマップの書き換えについてですが、当初はキーマップをウェブサイトで作成し、それをファームウェアと呼ばれる、ハードウェアの処理をつかさどる部分に書き込む方法で行っていました。その書き込みでやらかしてしまい、キーは効かず基板も光らずで、ただのクソデカ文鎮と化してしまいました。何時間も格闘したのに直らずへこんでいたところ、その状態から、我らがPC部部長Ryoga.exeがほんの数十分で直してくださいました。部長には感謝してもしきれないので、この場を借りてお礼させていただきます。ありがとうございました。ちなみに原因は単に書き込みがうまくいっていなかっただけでした。タメガ

余談2

パーツを店舗で購入した際、レシートの代わりに領収書を受け取ったので、キースイッチがなんのものなのかわからなくなってしまいました。小一時間インターネットを探し回っても見つからなかったので、自作キーボード界隈では有名なYouTuberのららぽん^[4]さんという方のTwitterのDMに突撃したところ、何と一瞬で見つかりました。この記事を書ききたのもららぽんさんのおかげなので、この場を借りてお礼させていただきます。ありがとうございました。

[4] https://twitter.com/rarapon_exe

Special Thanks

- 遊舎工房
- ジョイフル本田
- rarapon/ららぽん
- Ryoga.exe
- この記事を読んでくださった方々

プログラミング言語 Zig の基本文法

榎本舷希

はじめに

こんにちは。6期生の榎本です。パソコン部では、主に競技プログラミングをしたり、低レイヤーの勉強をしたりして遊んでいます。

さて、今回は Wikipedia を眺めていて発見した「Zig」^[1]と呼ばれるプログラミング言語を紹介したいと思います。

Zig is 何？

Zigは、GitHubにおいて、オープンソースで開発されているプログラミング言語です。2015年ごろから開発が開始された比較的新しい言語で、現在はversion 0.91が最新版となっています。

Zigは、堅牢で最適かつ再利用可能なソフトウェアを維持するためのプログラミング言語ならびにツールチェーンとされています。

Zig is a general-purpose programming language and toolchain for maintaining robust, optimal, and reusable software.

具体的には、メモリ不足などのエッジケースでも正確に動作し、プログラムが最適に動作/実行できるように記述され、同一コードが異なる多くの環境で動作するとされています。^[2]

[1] <https://ziglang.org>

[2] <https://ziglang.org/documentation/0.9.1/#toc-Introduction>



▲図1: ロゴ

Zig を使うには?

Zig コンパイラは、公式サイト [3] で配布されているバイナリをダウンロードし、パスを通すか、パッケージマネージャを通じてインストールすることができます。一部パッケージマネージャは過去のバージョンのみ対応しているものがあるので、注意が必要であるほか、最新版(master version)にはバグが含まれている場合があります。

```
$ export PATH=$PATH:~/Zig #ダウンロード後にパスを通す
$ brew install zig #もしくはhomebrewなどのパッケージマネージャを使用する
```

Vim や Vscode 用拡張機能や LSP も存在するので、GitHub やそれぞれのエディタのリポジトリなどを確認すると良いでしょう。

これ以降の解説は 2022 年 5 月 17 日時点の公式ドキュメントや最新の安定版である Zig version0.9.1 をもとにしています。Zig はメジャーリリースに達しておらず、今後仕様変更の可能性があります。

また、サンプルとなるコードは `sample.zig` として作成・実行しています。

Zig コマンド

Zig コマンドのサブコマンドとして使用する形式になっています。主なコマンドの一覧

- `zig run sample.zig`: `sample.zig` の実行ファイルを生成し、すぐに実行する
- `zig test sample.zig`: `sample.zig` の test コードを実行する
- `zig build`: `build.zig` の記述をもとにビルドを行う
- `zig fmt sample.zig`: `sample.zig` にシンタックスフォーマットをかける

[3] <https://ziglang.org/download>

このほかにもさまざまなコマンドがあります。詳しくは、`zig --help` コマンドを参照のしてください。また、サブコマンドのオプションも多数存在し、こちらは `zig [sub command] -h` を参照してください。

Hello,World!

さまざまなプログラミング入門書やリファレンスの初めには、`Hello,World!` と出力するプログラムが載っています。では、それに倣って Zig でも `Hello,World!` を出力してみましょう。

```
const std = @import("std");

pub fn main() void{
    std.debug.print("Hello,{s}!\n", .{"World"});
}
```

出力結果

```
$ zig run sample.zig
Hello,World!
```

まず、1行目の `const std = @import("std")` では基本的な機能やアルゴリズム、データ構造などを提供する Zig Standard Library（標準ライブラリ）を組み込み関数である `@import` 関数によって取り込んでいます。続いて、`pub fn main() void` では、`void` 型の `main` 関数を定義しています。この関数がエントリーポイント（最初に実行される）になります。`main` 関数のソースコードは、標準出力に `Hello,World!` と標準出力に出力をすることを表しています。`std.debug.print` 関数の二つの引数はコンパイル時に評価され、第一引数の `{s}` は第二引数の値（ここでは `World`）に置き換えられます。`\n` はエスケープシーケンスであり、改行を示しています。なお、出力は `std.log` 関数や `std.debug.print` 関数によっても行うことができます。

コメント

ソースコード中には機能の説明やメモなどの解説のためにコメントを書き込むことがあります。C 言語などでは複数行にわたってコメントを書くことが許されていますが、Zig は複数行に渡ったコメントは記述できません。

```
const std = @import("std");

pub fn main() void {
    // コメントは"//"から始まり、その行の最後までが範囲になる
    std.debug.print("Hello, world!\n", .{});
}
```

変数(整数・小数など)

Zigにおいて、変数は `(var|const) 識別子 : 型 = 値` の形で表されます。`const` 変数は不变の値であり、再代入が許されません。再代入により値を変えたい場合には、`var` 変数を使用します。`:型` ではその変数の型を示します。Zigではその型のビット数を表します。なお、型推論が可能な場合、省略することが可能です。

- 符号付き整数型: `i8, i16, i32, i64, i128`
- 符号なし整数型: `u8, u16, u32, u64, u128`
- 浮動小数型: `f16, f32, f64, f80, f128`
- その他: `bool, noreturn, void, type`

など上の整数型のほかに、任意のビット幅の整数を `i` もしくは `u` 識別子の後に数字をつけることで作成することができます。例えば、`u45` は符号なし 45bit 整数を表しています。

```
const std = @import("std");

pub fn main() void {
    var n: u8 = 50;
    n = n + 5; //nはvar変数なので代入可能

    const pi: u32 = 314159; //piはconst変数なので再代入不可かつ符号なし整数型なので負の数は表現できない

    const negative_eleven: i8 = -11;

    std.debug.print("{} {} {}\n", .{ n, pi, negative_eleven });
}
```

出力結果

```
$ zig run sample.zig
55 314159 -11
```

配列

Zigにおいて、配列は [要素数]要素型 の形で表されます。

```
const a = [3]i64 {100,200,300};
```

なお、要素数を推論できる場合には `_` を用いることができ、上の配列は以下のようにも表せます。

```
const b = [_]i64 {100,200,300};
```

配列の要素へのアクセスは `配列[添字]` で行うことができます。ただし、添字は C 言語や C++ などと同じく 0 から始まります。また、配列の要素数は `配列.len` で取得できます。Zig の配列には面白い性質があり、`++` 演算子で配列どうしの連結を、`**` 演算子で配列を繰り返すことができます。

```
const std = @import("std");

pub fn main() void {
    var some_primes = [_]u8{ 1, 3, 2, 3, 5, 8, 13, 21 };

    some_primes[0] = 1;
    const first = some_primes[0];
    const fourth = some_primes[3];
    const length = some_primes.len;

    std.debug.print("First: {}, Fourth: {}, Length: {}\n", .{
        first, fourth, length,
    });

    const ko = [_]u8{ 2, 0 };
    const ga = [_]u8{ 1, 2 };
    const koga = ko ++ ga;

    const wa = [_]u8{ 202 } ** 3;
```

```

std.debug.print("KOGA: ", .{});
for (leet) |n| {
    std.debug.print("{}\n", .{n});
}

std.debug.print("WA: ", .{});
for (bit_pattern) |n| {
    std.debug.print("{}\n", .{n});
}

std.debug.print("\n", .{});
}

```

出力結果

```

$ zig run sample.zig
First: 1, Fourth: 3, Length: 8
KOGA: 2012, WA: 202202202

```

文字列

Zig では、文字列をバイト配列として扱います。C 言語と似ていますね！（実際は違うのですが…ここではそういうことにしておきます）

```

const a = "kss-pc-club";
const b = [_]u8{'k','s'**2,'-','p','c','-', 'c','l','u','b'};

```

文字列は変数 a のように `" "` で囲むことで表すことができます。文字列をバイト配列として表しているので、a は b のようにも表すことができます。個々の文字はシングルクオート(`' '`)、文字列はダブルクオート(`" "`)で囲っていることに注意しましょう。これらは区別され、別のものです！

分岐(if)

Zigにおいて分岐は以下のように書くことができます。C や C++と似たような構文ですね。

```
if(条件式){
    //条件式がtrueの時の処理
}else{
    //条件式がfalseの時の処理
}
```

Zig では条件式には真偽値のみを書くことができ、他の数値や他の型のデータを書くことができません。許されるのは、`true` もしくは `false` のみです。なので、以下のコードはコンパイルエラーになります。C や C++ ではエラーとならず、実行することができますが、Zig ではできません。注意しましょう。

```
const std = @import("std");

pub fn main() void{
    const a = 1;
    if(a){
        //処理
    }
}
```

出力結果

```
$ zig run sample.zig
./sample.zig:4:8 error: expected type 'bool', found
'comptime_int' #出力結果
```

また、以下のようにも書くことができます。Rust の構文に似ていますね。

```
const c:i8 = if(a) 100 else -100;
```

繰り返し(while/for)

Zig には `for` と `while` のループがあります。`while` は条件ループ、`for` はイテレータを用いるループに使うことが多いです。

while

Zig の while はいくつかの書き方があります。なお、ループカウンタは while の外で宣言する必要があります。

```
const std = @import("std");

pub fn main() void{
    //ケース1:while(条件式){本体}

    var i:u32 = 1;
    while(i<10){
        std.debug.print("{} ",.{i});
        i += 3;
    }

    std.debug.print("{}\n",.{});

    //ケース2:while(条件式):(更新式){本体}
    //更新式は、ループの終わりに実行されます

    var j:u32 = 1;
    while(i<10):(i+=3){
        std.debug.print("{} ",.{i});
    }

    //ケース1とケース2は同じ結果を出力する
}
```

出力結果

```
$ zig run sample.zig
1 4 7
1 4 7
```

ケース 1 は C 言語でいう `while(条件式){}`、ケース 2 は C 言語でいう `for(;条件式;更新式)` のような形になっています。また、ループの途中で `continue` 式が登場した場合は、ループ内のそれ以下の処理を飛ばしループの初めに戻ります。また、ループの途中で `break` 式が登場した場合は、ループを終了し、次の処理に移ります。

for

for ループは配列に対して反復処理をするために使用されます。for は以下のよう構文を持ちます。また、while ループと同様、`continue` や `break` を使用することができます。

```
const std = @import("std");

pub fn main() void{
    const array = [_]u8{'k','s','s'};

    for(array)|character|{
        std.debug.print("{u} ",.{character});
    }

    //forループでは配列のindexを持つこともできる

    for(array)|character,index|{
        std.debug.print("no{d} is {u}",.
{@intCast(u32,index),character})
    }
}
```

出力結果

```
$ zig run sample.zig
k s s
no0 is k
no1 is s
no2 is s
```

サンプルコード中の `std.debug.print` 関数の{}の中に `u` と書いていますが、これは第二引数が UTF-8 の文字であることを表しています。もしこれをしない場合、文字コードが output されます。(この場合は筆者の環境においては、`107 115 115` と出力されました。)

関数

いままでも `main` 関数に代表されるようにさまざまな関数を使ってきましたが、使い方を詳しくみてみましょう。

```
const std = @import("std");

fn score(a:i32,b:i32) i32{
    return a * 10 + b * 30;
}

pub fn main() void{
    std.debug.print("{} {} {}\n",.{score(20,30),
        score(50,10),
        score(20,10),
    });
}
```

出力結果

```
$ zig run sample.zig
1100 800 500
```

上のサンプルコードでは `score` 関数を作成しました。関数は、`fn` 関数名(引数) 戻り値の型`{}` というようになります。`score` 関数では、`a` と `b` の引数をとり、戻り値は `i32` 型になっています。また、引数は 識別子:型 というようにかくことができ、ここでは `a`、`b` ともに `i32` 型の変数になっていることがわかります。ところで、`main` 関数には `pub` というキーワードが書かれていますが、`score` 関数には書かれていません。`pub` 属性は、他の Zig ファイルから `@import` された時に利用される関数・変数につけるものです。ただし、`main` 関数に限っては `pub` 属性をつけるルールになっているので `pub` をつけています。

おわりに

今回は、Zig の基本的な文法を紹介しました。本当はもっと書きたかったのですが、計画的なさから締切を過ぎてしまい、これ以上記事を書く時間がないのでこれでおしまいにします。すみません。しかしこれだけでは Zig の機能がほとんど伝わっていないので、以下に紹介していない主な機能（特徴）を記します。

- C 言語と ABI 互換がある
- C 言語のプログラムを組み込むことができる
- 厳しい型制約、型推論がある
- コンパイラは C,C++ をコンパイルできる
- webassembly 向けのビルドのサポートがある
- などなど

私自身、勉強途中なためあまり詳しく解説ができず、またこのような記事を書くことが初めてのため拙い文章になってしまいました。お見苦しい部分が多かったと思いますが、最後まで読んでいただきありがとうございました。また、パソコン部を設立し、発展させてくださった諸先輩方、そしてこの部誌の発行ならびに部の各種運営を行なってくださっている部長に感謝申し上げ、結びといたします。

参考文献

- zig Language Reference / 英語 (<https://ziglang.org/documentation/master>)
- ziglearn.org / 英語 (<https://ziglearn.org/>)
- ziglings / 英語 (<https://github.com/ratfactor/ziglings>)
- オープンソースプログラミング言語 Zig まとめ / 日本語
(<https://qiita.com/bellbind/items/f2338fa1d82a2a79f290>)

余談

実は日本語で Zig を解説している記事/サイトが 5 個くらいしかないので。どなたか読者で日本語の記事を書いてくださる方はいませんか…

始めよう、Arch Linux

香風智乃

はじめに

皆さんのパソコンでは、何の OS が動いているでしょうか？

恐らく、ほとんどの人が Windows と答えるでしょう。あるいは macOS ユーザーも居るかもしれません。いずれにせよ、GNU/Linux ディストリビューション（いわゆる “Linux” と呼ばれるもの）をデスクトップ用途における主力 OS として使用している人は、この部誌が主に対象とする読者層においてもなおひと握りであろうと思われます。

本記事では、そんな GNU/Linux ディストリビューションの 1 つである “Arch Linux” (`/a:rtʃ 'li:nəks/`) にスポットライトを当てて、その魅力をお伝えしたいと思います。



▲図1: Arch Linux のロゴ

Arch Linux を Arch Linux 足らしめている要素とはなんでしょうか？私が思う Arch Linux の特徴を以下に書き出してみます。

シンプルであること

KISS 原則という言葉があります。これは Keep It Simple, Stupid の略で、雑に意訳すれば「シンプルにしておけ、この間抜け」といった意味合いの言葉です。プログラミングの世界ではしばしば引き合いに出されるこの言葉は、Arch Linuxにおいても、その一番の特徴であると断言できるほどに非常に強く意識されています。

例えば、Arch Linux にはデスクトップ環境が同梱されていません。GUI を使いたければインストール時に自分でデスクトップ環境をインストールする必要があります。

「いやいや、デスクトップ環境は誰だって必要だろう」

そう思った方もいるかもしれません。しかし、GNOME、KDE Plasma、Cinnamonといった風に群雄割拠するデスクトップ環境の中からどれか1つを押し付けることはユーザーにとって不利益になる可能性があります。また、sshして使うサーバーを作りたいだけだからデスクトップ環境は要らないというケースも十分にあります。Ubuntuにおいては、このようなニーズはデスクトップ環境を差し替えた派生OSやUbuntu Serverといったディストリビューションによってカバーされます。しかし、もとよりデスクトップ環境を同梱せずユーザーの選択に委ねるという形の“シンプルさ”を選択したArch Linuxは、たった1つのエディションでそれら全てのニーズをカバーできるのです。それが、Arch Linuxの目指す“シンプルであること”なのです。

そのシンプルさ故に、Arch Linuxは玄人向けのディストリビューションであると説明されることがあります。しかし、それは全くの間違いです。Arch LinuxにUbuntuに見られるようなデスクトップ環境、GUIツール、そして様々な自動化が無いことは、決して対象ユーザーが上級者に限られることを意味しません。なぜなら、Arch Linuxには他のディストリビューションに類を見ない素晴らしいWikiが存在するからです。

充実した Wiki の存在

ArchWiki (<https://wiki.archlinux.jp/>) は、Arch Linuxそのものと並んで Arch Linux コミュニティの価値の中核を担う、非常に重要な存在です。ArchWikiには、Arch Linuxのインストールからリポジトリに収録されたパッケージの利用方法まで、ありとあらゆる情報が集約されたWikiです。Arch LinuxにGUIツールや自動化がデフォルトで付属しない代わりに、このWikiを読むことで大抵の問題は解決するようにできています。

また ArchWikiに書いてあることは他のGNU/Linuxディストリビューションでも共通する内容が多く、全てのLinuxユーザーにとって価値ある資料です。

最新であること

Arch Linuxでは、ローリングリリースと呼ばれるリリースモデルを採用しています。これは、(Windows 11とかUbuntu 21.04といった)ナンバリングをせずに、持続的に最新のパッケージを提供し続けるモデルです。なのでArch Linuxにバージョン番号はありません。年にn度の大規模アップデートといったものすらなく、日々パッケージをpacman(パッケージマネージャ)から更新していれば常に最新の環境を利用できます。新しいもの好きの人にとってこれほど愉快なことはありません。ほぼいつでも最新バージョンのアプリケーションが使えるのですから。

AUR による無限の拡張性

Ubuntuなどのディストリビューションを使っていて、リポジトリに収録されていないソフトウェアが必要になったことはありませんか？ そういった時、大抵は PPA (Personal Package Archive) を apt に追加してインストールしたり、あるいは手動で実行ファイルをダウンロードして配置するとか、ソースコードからビルドするといった手段を取ることになります。

しかし、Arch Linuxにおいてはそのような悩みとは永遠に決別できます。Arch Linuxには AUR (Arch User Repository) という、本流のリポジトリとは別の、コミュニティベースの（誰でもパッケージを追加できる）リポジトリがあります。AUR ヘルパーと呼ばれるソフトウェアを用いることで、この AUR に収録されている膨大な数のパッケージを本流のリポジトリからパッケージをインストールするのと同じように簡単にインストールできるようになります。これによって、Arch Linux は公式リポジトリのみでは到底カバーできないような無限の拡張性を得られるのです。

余談: Windows と比べた時のメリット

ここまででは Arch Linux と他の GNU/Linux ディストリビューションとの比較でしたが、Windows と比べた時のメリットについても少し述べさせていただきます。

安定感

先述の Arch Linux のシンプルさとも繋がりますが、Arch Linux が基本的にシステムの様々な部分を隠蔽せずにユーザーが直接設定することを志向していることは対照的に、Windows は多くの設定を GUI 経由で行うことを前提として設計されており、その実体が隠蔽されていて扱いにくい側面があります。しかし他方でコマンドプロンプトからの操作を必要とする設定や、複数のパラメーターに同時に作用する設定があり、これらは得てして設定の不整合に繋がりがちです。

また、Windows は NTFS の設計の都合上、特にファイルのメタデータへアクセスする際のパフォーマンスが悪く、大量のファイルが収容されているフォルダを開くなどの操作をした際にシェル (explorer.exe) やシステム全体が顕著に不安定になります。これに対して GNU/Linuxにおいて主に使われる ext4 や btrfs といったファイルシステムは比較的 I/O パフォーマンスに優れており、このような問題が発生する頻度は低いです。

パッケージ管理の楽しさ

最近の Windows では winget というパッケージマネージャが使えるようになったことで以前と比べれば状況は改善しているものの、それでもインストーラーを基本とした Windows の伝統的なバラバラのパッケージ管理は残っています。この点において Windows が Linux に対して優位に立つことは絶対に無いでしょう。

高いカスタマイズ性

Windows にはカスタマイズ不可能な部分が多くあります。システムフォントやウィンドウを操作した時の挙動はさることながら、エディションによっては運用の根幹に関わる Windows Update の可否すら自由に決められません。GNU/Linux ディストリビューションではこのようなことは一切なく、ユーザーが望むのであれば（デスクトップ環境にもよりますが）見た目や挙動を非常に幅広くカスタマイズできますし、パッケージマネージャを明示的に操作しない限り勝手にアップデートのために再起動されることもありません。

このように、Arch Linux は Windows に対しても、他の GNU/Linux ディストリビューションに対しても、非常に優れた側面を持つディストリビューションなのです。

おわりに

ここまで拙文を読んでいただきありがとうございました。他の方の記事と比べると技術色はかなり薄くなってしまったかと思いますが、本記事を通じて 1 人でも Arch Linux やその他の GNU/Linux ディストリビューションに興味を持ってくれる人がいれば嬉しい限りです。

私が Arch Linux を知ったのは 2020 年でした。初めはインストールの段階からコマンドラインで操作する必要があるという硬派な雰囲気に怖気づいて手を出せずにいたのですが、それまで使っていた Ubuntu と Windows 10 Insider Preview にも飽きつつあったので思い切ってメイン PC にインストールしてみたのです。それからおよそ 2 年。途中で Windows 11を入れたりもしましたが、やはり Arch Linux の快適さには敵わず、今も普段使いの OS として活躍しています。本記事では Arch Linux の素晴らしいところについて語らせていただきましたが、これに私の主観が多分に含まれていることは疑いようがありません。ですから、ぜひ自分に合った OS を探してみてください。私にとっての Arch Linux のように、皆さんがそれぞれ“とりあえず Windows”ではない、本当に使いたいと思えるような運命の OS に出会えることを願っています。

最後に、PC 部のかねてよりの目標であった部誌の発行を実現し、また私にその記念すべき第 1 号への寄稿の機会を与えてくださった部長に敬意と感謝（そして 3 度も締切を踏み倒したことへの謝意）を表明して、本記事の結びとさせていただきます。

KSS PC BOOK 2022

2022年5月x日 初版発行

著 者 KSS PC Club

印刷所 xxxxx

© 2022 KSS PC Club