D言語 Ver.2023

目次

競技プログラミングで出会った愉快なアルゴリズムたち 3
知っておこう関数ポインタ feat. ずんだもん 7
バッファオーバーフローを検証してみる 15
プログラミングの学習法 21
Web ページの眺め方の違い 24
最近気に入っている VSCode の環境 26
デバイスドライバー 30
アマチュア無線の紹介 32
ざっくり電算 38
作曲を始めたい 40
マウスを買い替えたのでレビューする 43
締め切りを守ろう 45
無題 46
一句 48

競技プログラミングで出会った愉快なアルゴリズムたち

Aozora

皆様は競技プログラミングをご存知だろうか。アルゴリズムの速度が大事になってくるものなのだが、知っておきたいアルゴリズムを書いていくよ。筆者は競プロ初心者であることにご注意いただきたい。あと、競プロのお話なのでアルゴリズムの本質は曖昧です。

- 動的計画法

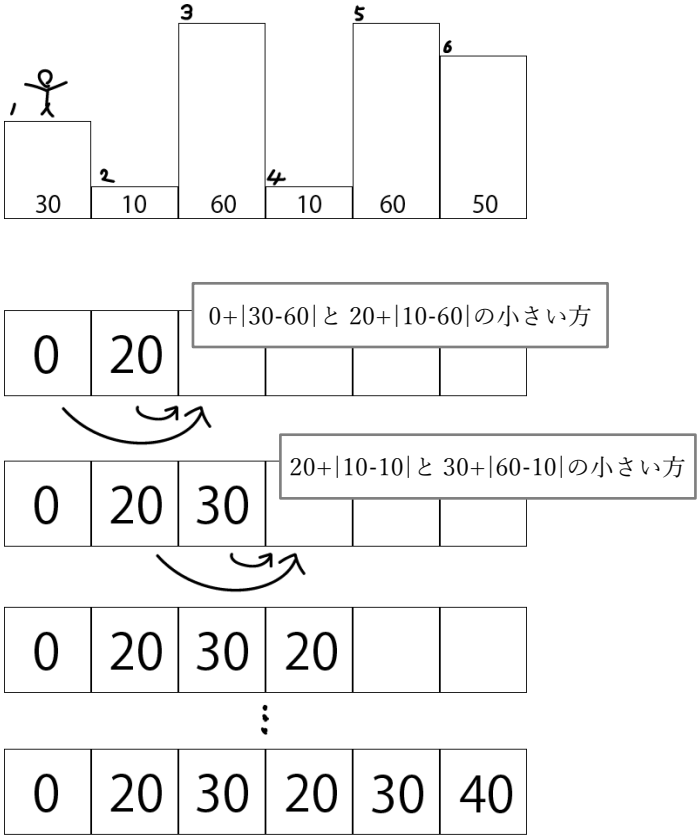
  Dynamic Programming の略らしい。だいなみっくだね。DP は計算したい問題を、より小さい部分問題を利用して計算するということ。とりあえず具体例↓

  Educational DP contest より Frog1

  1~N の番号がついた足場がある。各足場には高さが設定されている。ある足場にいる時、その次かさらに次の足場に移れて、高さの差分のコストがかかる。足場 1 から足場 N にいくまでの最小コストを求めよ。

— 解き方 —

ある足場に注目した時、そこにくるには、1 つ前か 2 つ前の足場にいる必要がある(自明)。なので、この足場までの最小コストは、1 つ前の足場までの最小コスト+この足場に移るコストと、2 つ前の足場までの最小コスト+この足場に移るコストの小さい方となる。これを利用して解こう。



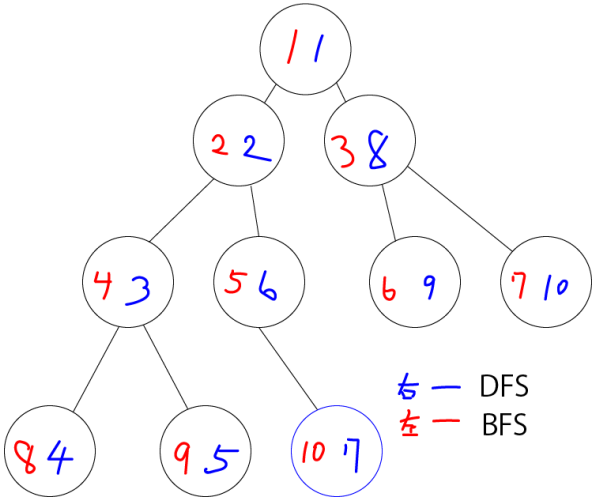
こういうことです。

これはめちゃ簡単な DP の例ですがナップザック問題など有名な問題もたくさんあるのでぜひ調べてね。AtCoder をやっている方は Educational DP contest を一度見てみることをお勧めします。

ここまでで疲れたのであとは概要だけ。気になったら調べてね。

- BFS（幅優先探索）

  Breadth First Search の略。迷路みたいな問題などを解きます。分かれ道が来たら分身して探索する感じ。



- DFS（深度優先探索）

  Depth First Search の略。迷路みたいな問題などを解きます。一旦行き止まりが来るまで進んで戻って、いっていない道があったら行き止まりまで行って、みたいな感じ。

  探索順はこんな感じ。

● その他

- ユークリッドの互除法
  最大公約数を高速に求める。
- エラトステネスの節
  素数列挙はこれ。
- 二分探索
  有名なやつ。日常生活でも使える一つ。

他にも知っておいたほうがいいことはありますが、とりあえずコンテストにたくさん出よう！

知っておこう関数ポインタ

feat. ずんだもん

itu

> 本稿で紹介するｃソースプログラムには`#include<stdio.h>`やプロトタイプ宣言が抜けてるからお手元の PC で実行する場合には気をつけて欲しいのだ。
>
> ずんだもんより

僕はずんだの化身、ずんだもん。県内の某高専に通う、平々凡々な高専生なのだ。よろしくお願いしますなのだ！

最近、僕は C 言語の授業で「関数ポインタ」というヤツを習ったのだ！とっても便利だから、みんなに教えるのだ！

関数ポインタとは？

そもそも『関数ポインタ』ってどんなものなのだ？

まずは、パソコンのメモリ領域を想像してみて欲しいのだ。僕たちが基本的に触るメモリ領域は、変数だけが格納された領域。でも、PC 全体のメモリ空間には、変数以外のデータもたくさん詰まっているのだ。

関数も変数と同様、プログラムが実行されると、メモリのどこかにその実体が展開されるのだ。要するに、関数は、変数と同じ、「メモリ空間」に存在し、「メモリアドレス」が存在するということ。変数と関数は、実は似ているのだ！

『関数ポインタ』といっても、変数に対するそれと大きな違いは無いのだ。……つまり、変数がそれぞれアドレスを持っているように、関数だってアドレスを持っているのだ。

百聞は一見にしかず、まずは下のプログラムを見るのだ！

```c
/*絶対値を求める関数*/
int abs(int x) { return x > 0 ? x : -x; }

int main() {
    printf("%p\n", abs); // 出力 => 0x401130
}
```

これを見てわかる通り、な・な・なんと、printf で、関数 abs のアドレスを表示させているのだ。ビックリ仰天なのだ！

（……ところで、printf の%p で、メモリのアドレスが表示できるのは覚えているのだ？）

さて、ここからわかる通り、C 言語は、宣言した関数の「関数名」のみを記述することで、その関数のアドレスを取得することができるのだ。これこそが、『関数ポインタ』の正体なのだ！

ポインタ変数

さて、ここまでで、関数の正体が「メモリ上に存在する」ことはわかったのだ。そこで、このアドレスを「ポインタ変数」に格納してみようと思うのだ。

では、実際に『関数ポインタ』を格納した「ポインタ変数」を作ってみるのだ！

そこで、作ってみたものが、次のプログラムなのだ。

```
1  /*絶対値を求める関数*/
2  int abs(int x) { return x > 0 ? x : -x; }
3
4  int main() {
5      int (*fp)(int);
6      fp = abs;
7      printf("%p\n", fp); // 出力 => 0x401130
8  }
```

んんー、ちょっとややこしいけど、落ち着いてよく見てみるのだ。さあさあ、まずは、`main` 関数の最初の行にご注目！ この行の `int (*fp)(int)`というのは、ポインタ変数 `fp` の宣言なのだ。

この宣言は、「`int` 型の値を受け取って、`int` 型の値を返す関数」という「型」をもった関数ポインタ `fp` を宣言する文。要するに、ただの変数宣言なのだ。(でも、ちょっと、さすがに構文がわかりにくいと僕も思うのだ……)

さて、これでポインタ変数に対して、関数のアドレスを格納することに成功した。ここから、こんな面白いことが可能になるのだ。次のプログラムを見て欲しいのだ。

```c
/*絶対値を求める関数*/
int abs(int x) { return x > 0 ? x : -x; }

int main() {
    int (*fp)(int);
    fp = abs;
    printf("abs(-10) = %d", abs(-10));
    printf(" fp(-10) = %d", fp(-10));
}
```

（このとき、`abs(-10) = 10 / fp(-10) = 10` と表示される）

な・な・なんと、さらにビックリなのだ！ 関数 `abs` が、ポインタ変数 `fp` を使って呼べるようになったのだ。ザッツライト、これこそが『関数ポインタ』のチカラなのだ！

map 関数

それでは、「関数ポインタ」で、できることを教えるのだ。

それは**高階関数**なのだ。（ホントは仕様上、厳密な高階関数は作れないのだ。あくまでも再現だと思って欲しいのだ）

高階関数とは、「関数を受け取る関数」のこと。Python や C#などにも同じ機能が実装されていて、「コールバック関数」なんかでお世話になった人も多いと思うのだ。

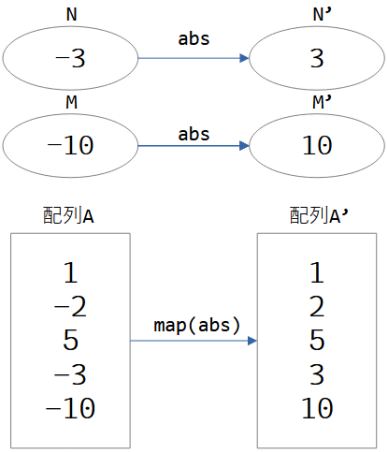
一見、めんどくさそうな要素に見えて、コレはとっても便利な機能なのだ！ これを使うことで、ソースコードがエレガントでロジカルになるのだ！

それでは、これから高階関数を作ってみるのだ。今回は、map 関数というものを作ってみる。使用例は以下の通りなのだ。

（map 関数はデータ構造の Map とは別物だから、一応注意するのだ）

```
1  int addone(int x) { return x + 1; }
2  int square(int x) { return x * x; }
3
4  int array[5] = {1, 2, 3, 4, 5}
5  map(array, addone} // => array = {2, 3, 4, 5, 6}
6  map(array, square} // => array = {4, 9, 16, 25, 36}
```

ここでいう map 関数とは、「第一引数で配列を受け取り、配列の中身全てに受け取った関数を適用する」はたらきをもっているのだ！なんだかわかりにくいけど、要するに……



左図は、abs 関数（絶対値を求める関数）を map 関数に適用した例なのだ。

abs 関数は int 値を操作する関数で、本来、配列を直接操作することは出来ないのだ。

ところが、map 関数を使うことで、「abs 関数を配列に対して適用する」ことが可能になるのだ！

これこそが map 関数。たまに使うことがあるから、覚えておいて欲しいのだ！

……と、いうわけで、僕も map 関数を実装してみたのだ！ホレボレする出来なのだ！

```
void map(int array[], int array_size, int (*func)(int)) {
    for(int i = 0; i < array_size; i++) {
        array[i] = func(array[i]);
    }
}
```

やってることは単純で、受け取った配列の各要素に、for ループで関数を適用しているだけなのだ！

あとは、お手元の「int 型を受け取り、int 型を返す」関数をご用意ください。そうすれば、その関数を任意の int 型配列に適用できちゃうのだ！

この出来立てホヤホヤの map 関数を、実際に使ってみるのだ！

```
int triple(int x) { return x * 3; } // 入力値の三倍を返す
int cube(int x) { return x * x * x; } // 入力値の三乗を返す

int main() {
    int array[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    map(array, 10, triple);
    map(array, 10, cube);
}
```

（このプログラムで、配列がどう変化するか、皆も考えてみて欲しいのだ！）

まとめ

いかがでしたのだ？ この記事で、関数ポインタに対する理解が深まったのであれば嬉しいのだ。

最近の言語のほとんどには高階関数が実装されているので、お使いの言語があれば、ぜひそっちも使ってみてほしいのだ！(そういえば、Python や C#などの言語は、「ポインタ」が存在しないにもかかわらず、高階関数が実装可能になっているのだ。これは、きっとナニかのカラクリがあるに違いないのだ！)

お読みくださりありがとうございましたなのだ！ 再見！

おまけ

紙に余白が余ったので、謎かけやります。当ててみてね。

えー、「部下のクビを切りたがる IT 企業」と掛けまして、
「優秀だった頃の自分を懐かしむ高専生」と解きます。

その心は……「かいこ（解雇／懐古）もほどほどに」でしょう。

お付き合いいただき、ありがとうございました。

バッファオーバーフローを検証してみる

yukidoke

1. はじめに

C 言語において、stdio.h で定義されている scanf()関数にバッファオーバーフロー脆弱性が存在することをご存じの方は多いと思います。まぁ…やったことないのでやってみよう！という話です。

2. 環境

OS　　：Ubuntu 20.04.4 LTS (WSL2)
C コンパイラ　：gcc 9.4.0

3. 検証

図 1 に示すようなプログラムを用いて検証を行います。

```c
#include <stdio.h>

int main()
{
    int temp = 0;
    char a[8], b[8];

    scanf("%s", a);
    scanf("%s", b);

    printf("%s¥n", a);
    printf("%s¥n", b);
    printf("%d¥n", temp);

    return 0;
}
```

図１：sample.c

3.1. 文字列に対する BOF

文字列に対する BOF を検証します。図 1 のプログラムを以下のコマンドでコンパイルします。

```
gcc -o normal sample.c
```

生成された”normal”を実行して図 2 に示すような入力を与えます。

```
yukidokeNDC
aka.ty
```

図２：input

すると、図 3 に示すような出力が得られます。

```
yukidokeaka.ty
aka.ty
0
```

図 3 ：normal の出力

b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]
a	k	a	.	t	y	¥0	

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
y	u	k	i	d	o	k	e

図 4：a, b の内容（normal）

配列 a, b の内容は図 4 に示したようになります。起きていますね、バッファオーバーフロー。1 度目の scanf()で起こっています。

配列 a にはヌル文字を考えると 7 文字までしか入りません。しかし、11 文字入力されたので、バッファオーバーフローが起き、配列 b の範囲まで入力されます。その後、配列 b は入力で上書きされます。

スタックを考えると図 5 のようになります（データサイズが正確ではないですが)。

2	char b[8]
1	char a[8]
0	int temp

図 5 ：normal のスタック

スタックは上から下に書き込まれるので a がオーバーフローすると…あれ。本来ならば上書きされるのは int のはずです。

実はコンパイラが int の値を変えないように保護しています。

3.2. int に対する BOF

前項で言及した、int へのバッファオーバーフローをやってみたいと思います。

図 1 のプログラムを以下のコマンドでコンパイルします。

```
gcc -o bof -fno-stack-protector sample.c
```

-fno-stack-protector オプションによって、コンパイラによるスタック保護を外しました。

生成された”bof”を実行して図 2 に示したような入力を与えます。

すると、図 6 に示すような出力が得られます。

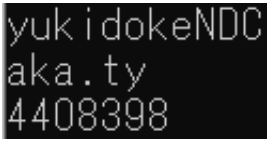


図 6 ：bof の出力

int temp の値が変わっちゃっていますね…。スタック保護がなくな

ったので、プログラムは何も気にすることなくスタックに書き込みます。

スタックは、最終的には図 7 のようになります(こちらも正確ではないです)。

b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]
a	k	a	.	t	y	¥0	

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
y	u	k	i	d	o	k	e

int temp			
N	D	C	¥0

図 7 ：bof のスタック

ASCII コードより、ＮＤＣ¥0 は 16 進値で 4E 44 43 00 となります。バイトオーダーはリトルインディアンなので、temp は 0x0043444E だとわかります。これを 10 進値に直すと 4408398 となって図 6 の出力と一致します。

４. 感想

本当に書き換わるんだ…。入力で変数を上書きできるとか、あまりにもガバガバすぎるのでは？と思ったり、思わなかったり。

ま、全体としてはかなり面白かったです。脆弱性に触れるって痺れますよね。

こういうのが好きな人は CTF とかやってみるといいと思います。たぶん。

ちなみに、3.1.のやつでも 3.2.のやつでも対策は簡単です。図 1 の%sを%7s にすればいいです。

プログラミングの学習法

UT

近年、若いころからプログラミング教育をしようという声が高まっている。プログラミングによって作られた製品やシステムが身の回りに有り余るほど存在しているからだ。昔と比べれば、今は中学や高校からスマホを持つのが当たり前のようになってきている。地域や学校のイベントでプログラミング教室が開かれたり、プログラミング学習の本が多く売り出されたり、YouTube をはじめとする動画投稿サイトでのプログラミング動画が増えてきている。プログラミング学習ができるサイトなどもある。SCRATCH という学習サイトなどがその例だ。

しかし、学習する手段がたくさんあったとしても、実際に学習するのは学習したいと思う本人である。小・中学校の授業でのプログラミング学習では、習得できる人もいるが、うまく習得できないという人のほうが圧倒的に多い。どうすれば確実にプログラミング技術を身に着け、課題や要求にあったものを自由自在に作れるようになるのか。いままでの自分の勉強やプログラミング学習で用いていた方法の中で、よかったものを記したいと思う。

その１：自分で確かめていく方法

この場合は、教科書や参考書、本など（以下「教材」と称する）を用いていく。まずは、教材の手順通りに学習をして、サンプルプログラムなどがあれば実際に書いていく。教材の内容を覚え、教材に記されているプログラムを実行し指示通りに動くか確認する。あとはそれを繰り返して、教材を進めていく。

このままでも十分技術を身に着けていけるが、教材の内容だけしか覚えられないため、それを応用する場面でうまく対処できなくなってしまう。これを避けるために、ひと工夫を加える。方法としては、教材で学習しているときに疑問に思ったことを記録する。そして、その疑問を解決させるために、新たな教材を使って調べたり、先生や詳しい人に質問したり、自分で試してみたりする。これを行うことで、自分の疑問が解消され、教材に書かれたこと以外のことを学習することができる。自分の知りたいことを自分のものにできるため、その知識は扱いやすく、応用しやすくなる。

その２：サンプルを改造していく方法

サンプルというのは、プログラミング学習サイトに投稿されている作品や、教材に記されているプログラムなどをさす。まずは、これをそのまま実行する。サンプルの動きの流れなどが分かったら、どんな風にプログラムが書かれているのかを見る。動きの流れを照らし合わせて、どの部分のプログラムが実際の動きと対応しているのかを判断する。もちろん、わからない部分は調べてよい。だいたい理解できてきたら、改造してみる。動く速さを変えたり、ステージやオブジェクトの色を変えたりするといったささいなことで良い。そして実行してみる。エラーが起き

たら、前の状態に戻して改造し直したり、もう一度調べたりする。うまく実行できるようになったら、どういう風に変えたらそうになったのかを考える。文字の内容を変えたのか。数字の値を変えたのか。書く順番を変えたのか。不安であれば似たような改造をほかの部分でも行う。

自分の思うように改造できるようになれば、その部分の知識が増えるため、技術が増えていく。うまくいかなくても、何回もいろんな方法を試していくことが重要である。改造が難しいと思う人には、教材などの練習問題がおすすめである。教材には学んだ知識を試す問題があり、解答や解説を通じて知識の理解を深めることができる。

今回は 2 つ説明したが、ほかにも方法はたくさんあり、人によって合うものが違うと思う。まずはできそうなものから取り組んでみて、自分にとってやりやすい方法で学習していってほしい。

Web ページの眺め方の違い

lime

普通の人の場合

新しいスマホ欲しいな～
「iPhone14 値段」で検索っと…
うわ、たっか。Android にしよ
⇒ **Web ページを開かない**

意識高い人の場合

やっぱり iPhone 欲しいよな～
「iPhone 公式」で検索！
公式ページあった…
iPhone14、NEW って書いてある、購入ヨシ！
⇒ **Web ページを開く**

考え込む人の場合

スマホ 2 台持ってるけど、最近こっちの赤いスマホ調子悪いな
新しく iPhone 買おうかな、「iPhone14」と…
!!これは！スマホが宙に浮いている
犯人はどんなトリックを使ったんだ

⇒ **驚くけど分からない**

裏技を知った人の場合

公式ページに来てみたんけど、iPhone14 が浮いてるね
お、そんなになめらかに動けるのか、いい術式使ってんじゃん
ま、どんな術式かは御生憎様、目がいいもんで…
スゥー、大丈夫、デベロッパーツール、最強だから
まあ大丈夫でしょ 10 秒で…
ナニコレ？

⇒ **やっぱり分からない**

---

10 秒でわかる天才オタクの場合

iPhone の Safari 使ってみたいな
やはり iPhone 浮いてますね〜
カラーを変えると、ほぉ〜なるほど
transform の matrix する感じですか
…iPhone ページ無限恒久永遠推し!!!

⇒ **公式ページが推しになる**

結論　かっこいいページはすごい！
みんなも高専生探偵になって Web ページの正体を暴こう！

最近気に入っている VSCode の環境

maru

こんにちは、部長です。みなさん VSCode はご存知ですか？VSCode とは天下の Microsoft 様が開発し、無償で提供している神エディタです。操作性や機能性も抜群。これを知らない・使ってない人は今すぐ使い始めましょう。

この記事は VSCode 初めて使う初心者向けの記事……ではなく、最近使い始めて全然自分なりの環境を作っていない人向けに自分のお気に入りの環境を紹介します。

1. JetBrainsMono

こちらの JetBrainsMono とは、JetBrains 社がオープンソースで公開しているめっちゃ見やすくてかっこいいフォントです。このようなフォントです。

```
#include <stdio.h>
int main() {
    printf (“Hello World!”);
}
```

めっちゃ見やすいっすよね。あとこのフォントにはリガチャという機能があり、これもめっちゃ良いです。このようなもの……と言いたいと

ころでしたが、なぜか Word では対応していなかったのでご自身で調べてみてください。

2. One Dark Pro

次は One Dark Pro という VSCode のテーマです。こちらはもともと Atom エディタで使われていたテーマだそうです。また、VSCode で最もダウンロードされたテーマの 1 つらしいです。とてもスタイリッシュなデザインで結構好きです。

3. clang-format

これは C 言語のコード整形を行ってくれるソフトです。VSCode だと C/C++拡張機能を入れると一緒についてきます。そのままでも十分使えますが、ひと手間加えるともっと良くなります。開いているフォルダのルートに.clang-format というファイルを作成し、設定を記述します。

```
BasedOnStyle: Google
IndentWidth: 4
ColumnLimit: 0
AlignConsecutiveAssignments: true
AlignArrayOfStructures: Right
```

1-3 行目はそのままなので割愛します。気になるのは 4, 5 行目です。

AlignConsecutiveAssignments とは連続した代入の'='を揃えてくれるものです。Latex 数式の'&'でイコールを揃えるのに似ていますね。これを自動でやってくれます。

AlignArrayOfStructures とは二次元配列を超綺麗に整形してくれます。例えばこんな感じです。

```
// 整形なし
int table[][] = {
    {3, 5, -7},
    {11, -2, 132},
    {-33, 5, 2}
};

// 整形あり
int table[][] = {
    {  3,  5,  -7},
    { 11, -2, 132},
    {-33,  5,   2}
};
```

めっちゃ綺麗だと思いませんか？これを知ったときとても感動しました。おすすめです。

4. Code Runner

最後に VSCode でよく使う Code Runner のコマンドをほんの少しいじります。まず変更前と変更後を見比べてみましょう。

変更前

```
cd $dir && gcc $fileName -o $fileNameWithoutExt &&
$dir$fileNameWithoutExt
```

変更後

```
cd $dir && gcc $fileName -o $fileNameWithoutExt
&& ./$fileNameWithoutExt && rm $fileNameWithoutExt
```

このような感じです。ここで注目してほしいのが、最後のコマンド『rm $fileNameWithoutExt』です。コードをコンパイルしプログラムを実行させたあと、実行ファイルを削除しています。これの何がいいのか。勘の良い人ならお気づきかもしれません。そうです、私は A 型です。ただただ実行ファイルがディレクトリに含まれているのが気に食わないのです。これをすることに意味があるのか、私にはわかりません。でもこれをすると私は気持ちいいです。A 型の皆さん、ぜひこの設定をしましょう。

さいごに

ここまで読んでいただきありがとうございました。またどこかでお会いしましょう。

デバイスドライバー

sakaue

デバイスドライバーとは

マイコン制御に近しいもので違うもの。PC に USB 挿して読み書きしたり、スマホと繋いで画面共有だったり…PC と接続するものほぼ全てにデバイスドライバーというプログラムが働いています。デバイスドライバーをインストールしなくても OS に元から入っているものが多く、キーボードやマウスなど何も気にせず挿せます。他はデバイス購入時についてきたり、ホームページでダウンロードしたりと。しかし、デバイスドライバーを開発する機会など無いでしょう。古すぎるデバイスを買ってデバイスドライバーが動かない、大企業に就職してデバイスを開発する、などプログラマーでもやらないでしょう。しかし、私はそんな特殊な状況下でデバイスドライバーを 1 人で作ろうとしました。そんな経験を話していきます。

デバイスドライバーの中身

デバイスドライバーはまずこのデバイスがなんなのか、定義してファイルに保存しなければなりません。そのために、接続時のレスポンス(帰ってくる信号)で個体番号を判別して、それと結びつける詳細情報を記したファイルを置いておく必要があります。次にその個体番号や詳細情報からそのデバイスにアクセス出来るライブラリを作ったりソフトを作ったりしないといけません。それに対応してデバイス側にもプログラムをしておく必要があるわけです。そこではマイコンのような知識や USB

の送信データ構造などの知識もちょっと必要だったり…つまり自作デバイスドライバーなんかはデバイスに対応したものとなることが多いわけですね。

デバイスドライバーの大変なところ

デバイスと対応させなきゃいけないのにまだデバイス側をいじっているわけではない私が言えることではないですが、デバイス側の設定とOS側での設定と合わせるのが難しいです。また、USBポートをC言語のWinAPIでいじっているのでハンドル(デバイスのアドレスみたいなもの)の扱いが難しいです。しかし、普段私たちが使うUSBポートの仕組みを知ることが出来ました。

まとめ

直接的にデバイスドライバーを作った経験が生きることは無いでしょう。しかし授業でやるよりよっぽど実用的で新しいことをたくさん知ることができます。好奇心がある方はやってみると面白いと思いました！

アマチュア無線の紹介

JJ1URG

周りがパソコンクソ強人間ばっかりなので…あの…大した話はできないんですけどね…

この記事は内容は不十分であったり、誤解を招く表現が使用されていたりする可能性があります。そんな世界もあるんだ程度に読み進めていただけると幸いです。

１．初めに

アマチュア無線という世界をご存じでしょうか。アマチュア無線とは、アマチュア無線技士の資格により運用される無線です。

２．そもそも無線にはどんなものがあるのか

無線には、無線従事者免許という資格があります。これは大きく

1. 総合無線従事者
2. 海上無線従事者
3. 航空無線従事者
4. 陸上無線従事者
5. アマチュア無線従事者

に分かれます。
そしてそれぞれの資格による無線局が開局され、世界にただ一つのコールサインを国からもらい、運用されます。
アマチュア無線局は、この中のアマチュア無線従事者により運用できる

ようになる無線局です。

3．アマチュア無線ってどんな感じ？

3.1．ほかの無線局と違う点

アマチュア無線局が他の無線局と決定的に違う点に、業務に利用できないという点があります。また、他の無線局では基本的に周波数が決まっているのに対し、アマチュア無線局はアマチュアバンドと呼ばれるアマチュア無線のために定められた範囲の周波数のなかで運用されます。(正確にはその中でも色々制限がありますけどね。)

また、資格が要らない無線(ライセンスフリー無線などと呼ぶ)と比較して、出力を高くできる点や、アンテナの選択肢が多い点などがあります。

3.2．アマチュア無線の主な楽しみ方

アマチュア無線で最もメジャーな楽しみ方は、たまたま聞こえてきた「どなたか交信しませんか？」という旨の電波をキャッチして自分がそれに応答するか、自分がそのような内容の電波を出して応答を待つような楽しみ方です。
相手が応答するまで自分が発した電波が相手に届いているかどうかの緊張感、また相手がこちらに応答してくれた時の面白さがアマチュア無線にはあります。そして、見ず知らずの電波越しの音声とのんびり楽しんだり、とにかく交信数を稼いだりする楽しみがあります。

また、アマチュア無線には QSL カードなるものが存在します。これは交信が成立したとき、交信したことを証明するためにお互いに送りあう証明カードです。アマチュア無線は、いろんな人から QSL カード

を集めるという楽しみがあります。しかし、私も含めてQSLカードを作っていない人は、QSLカードの交換なしに交信することももちろんあります。

また、突然聞こえてきた誰かと交信するだけでなく、アンテナや、場合によっては無線機すらも自作し、自分で作ったもので電波がどのくらい飛ぶのかを試すという楽しみもあります。アマチュア無線は、特にアンテナに関しては自作しても何の問題もないので、どんどん自作できます。

もちろんこれらだけではなく、業務以外であれば、様々な実験を行うことができます。

４．無線技術の簡単なお話

ここからは、軽く技術面の話を書いてみたいと思います。

４.１.電信

音声を搬送波に乗せて送信するのとは違い、搬送波をモールス符号にのっとって断続的に送信する、CW(電信)というモードがあります。他の音声通信よりも簡易的で、電気通信の世界で最も伝統的なモードです。CW は、現在アマチュア無線以外の業務無線ではほぼ使われなくなってしまいましたが、アマチュア無線では依然として現役バリバリに使われています。

他の音声通信と比べて送れる情報に制限はありますが、とにかくトンとツーを認識できれば交信が成立するので、比較的弱い電波でも遠くの人と交信することができます。

４.２．周波数による違い

無線は、周波数によっても様々な特徴があります。

電磁波は波であるので、周波数ごとに固有の波の長さがあります。電波を出すためには、基本的に波長の半分の長さか、1/4 の長さのアンテナが必要になります。

ところで、周波数には区分が決められています。アマチュア無線の電信、音声通信で主に使われている区分は、3M～30M Hz を指す短波(HF)、30M～300M Hz を指す超短波(VHF)、300M～3G Hz の間を指す極超短波(UHF)です。

４.２.１. VHF/UHF

VHF と UHF は特徴が似ているので、ここではまとめて説明します。VHF/UHF は、基本的に見通し距離にのみ届きます。しかし、波長が 10m から 10cm と比較的良心的で、無線機も小さくて、”比較的”（←ここ大事）安いので、難易度が低く、初心者がまず通る区分になります。主に FM が用いられ、ゆったりとお話を楽しむ方が多くいらっしゃいます。

４.２.２. HF

HF の特徴は、とりあえず遠くまで飛ぶことです。上空にある電離層とよばれる層と地面を反射して、日本全国に簡単に飛びます。主に SSB や電信が使われます。HF は波長が 100m から 10m と割と大変な長さであり、市販自作関係なく電波を飛ばすためにはアンテナの長さを細かく調整しなければなりません。無線機も VHF/UHF の無線機と比べて大きく、高いです。しかし、難易度が高い分日本全国に飛ぶということで、アマチュア無線の醍醐味を多く教授できる周波数なのではないかと思います。

５．終わりに

ここまで長々とアマチュア無線について書いていきました。アマチュア無線を始めるためには、まず免許を取り、無線機とアンテナを購入し、開局申請を総務省に行わなければなりません。
正直お金はかなり消えます。難易度も高いです。HF を運用したり、アンテナなどを自作したりするためには、電気回路、特に交流についての知識が不可欠となります。しかし、かなりの知識が必要だからこそ、アマチュア無線を通して勉強できるものはかなり大きいです。近年は資格を取得するために、JARD による講習会もあり、比較的簡単に資格を取得することができます。

この記事で書いたことは、アマチュア無線や無線技術のほんの一端にすぎません。長くなってしまい端折った内容もあります。また、正直近年アマチュア無線は人口が減ってきて、だんだん廃れてきています。これを機に、少しでもアマチュア無線について興味を持っていただけたなら幸いです。

最後になりますが、こんなカスみたいな文章を読んでくださり、ありがとうございました。

ざっくり電算

卵食い倒れ

「電算って頭良さそう（粉みかん）」という浅はかな考えで入部した一年E科の私が4ヶ月ほどで何を学んだかをざっくりまとめさせていただくコーナーです。

最初期

「何から初めてなにすればええの(＾ω＾)」

って感じでひたすらブラウザ版の C 言語環境で本とパソコンとにらめっこしておりました。しかしながらプログラミング初心者の私では hello world の文字を出すのや簡単な計算すら一苦労。「まーじでこれマトモに出来るようになるまで何年掛かんだ」と思いつつも頑張って時には回数を指定した計算をやらせようとするが無限に計算し続けて結果がすんごいことになったこともありました。

部内4分割

その後部内で幽霊部員が多すぎる問題を解決しようとした先輩の計らいにより競プロ、web、サーバーサイド、ゲーム開発の 4 つの班に分けられました（これ実質 4 つ部活出来てね？）。そしてサーバーサイド班の所属した私は分かったか分かってないか微妙な理解度ではあるものの Linux を学び初めました。

成果

未だにC言語は扱えずLinuxも知識が浅いですがLinuxに関してはE科の授業電子メディア工学序論のなる授業で「あっ、ここ部活でやったところだ!!」と言う進研ゼミ的展開もあったのでやってよかったなぁと思いました。ちなみに電子メディアのテストは赤点でございました。(^Д^)ﾌﾟｷﾞｬｰ

作曲を始めたい

イイジマ

こんにちは、イイジマといいます。私はピアノを習っているのですが、クラシックしか弾いてこなかったんですね。でも、クラシックとか誰も興味ないし聞かないです。だから、ポップスの練習とかをしていたんですが、作曲もかっこよくねとなりました。

でも、何から始めればいいかわかんないし、てかどうやって曲作るのかも知りません。それで自分なりに、伴奏をつけるところから始めてみたら結構作曲してる気分になれたので、その体験記を書いていこうと思います。
音楽経験なしの方を対象に書いていくつもりです。

1. ジュラシック・パークを見る
スティーヴン・スピルバーグの傑作映画です。まずはこれを見ました。

2. 感動する
感動します。

3. メインテーマを耳コピする
感動したので、これを弾けるようになりたくなって、YouTube の動画を聞きながらメロディーだけ耳コピしました。

4. 伴奏をつける
大変でした。作曲に最重要なメロディーは与えられたとはいえ、メロディーを生かす伴奏を作るのは時間がかかりました。

５. 終わり

達成感を得て終わりです。

詳しく書いていきます。

映画を見て感動する経験は誰しもあると思うので割愛します。次に、メロディーの耳コピですが、これはいろいろ方法があると思います。絶対音感がある人はそのままでいいし、なければ、楽譜を見てしまってもいいと思います。音感を鍛えたい人は、一音一音確認していきましょう。楽器を持っていなくても、スマホ用のピアノアプリはありますし、こつこつ埋めていけるはずです。

ちなみにですが、楽譜エディタは MuseScore を用いました。五線譜で書けるのがいいし、打ち込んだ音楽を細かく再生できて便利でした。あと何より無料です。DTM ソフトの方がつかいやすければそっちでもいいと思います。

続いて苦戦した伴奏ですが、マジで何もわからなかったので勘でやってしまいました。濁ったらやり直して、メロディーに対してうるさすぎたら控え目にして…を繰り返して、なんとか 1 ページ書き終えました。

どうしても濁りが消えないときは、まずメロディーに対して一音与えてみて、いい感じだったらもう一音…とやっていきました。

もっちゃインクリメンタルです。

はじめは気づいてなかったのですが、裏拍で入るところがあったりして、初心者の自分には難しいところもありましたが、完成です。

僕の作成した楽譜は GitHub[1]で公開してあるので、良かったら YouTube に上がっているものと聴き比べてみてください。金管楽器とは違う良さがあると思います。

全部で五時間くらいかかりました。疲れました。

右手でしか弾けなかった曲がいい感じに両手で弾けるようになっ

[1] https://github.com/linakoi/JURASSIC

て最高でした。もっと速く、正確な楽譜が作るには、音楽理論を学ぶ必要があるので、すこしずつ学んでいきたいですね。

あと、メロディーも作曲したい。

読んでくださりありがとうございました。

マウスを買い替えたのでレビューする

yamazaki

最近マウスを買い替えたのでレビューしたいと思います。

「HyperX Pulsefire Haste Wireless」について詳しく見ていきます。最大 100 時間持続するバッテリーを備えていながら、本体重量 61g と軽量設計です。付属の USB ドングルと延長アダプタにより通信の安定性を確保でき、バッテリー充電方法もシンプルで、汎用性の高い USB Type-C 端子を備えています。

サイズ感は万人向けで、傾斜の作り方はやや背の高い Zowie FK、両サイドは本体後部の膨らみを除いてほぼ水平となっています。あらゆるグリップスタイルで干渉しづらく持ち方を選びませんが、この手の形状は持ち方が定まりづらく、強いフィット感が得られづらいことには注意。

ビルドクオリティは可もなく不可もなくで、メインボタンのシェルとホイールの取り付け精度が甘いです。どちらもフィーリングに悪影響なだけで、パフォーマンスを損なうほどではないので、価格相応のクオリティということで納得はできます。

スイッチはうまく調整されています。特にメインボタンは押下圧と跳ね返り感のバランスがうまく取れており、とても優れたクリック感をもたらしています。センサーは明らかに体感できるほどの DPI ズレが生じているため、ファームウェアアップデートによる改善が待たれるところ。

結論として、Pulsefire Haste Wireless はいくつかの懸念点を抱えていますが、価格を考えればよく出来たゲーミングマウスです。付属品も充実していながら、有線マウスと変わらない価格で入手できるので、安価で軽量ワイヤレスマウスを探している方の有力候補になります。

締め切りを守ろう

ice_adu

１．結論

　締め切りを守っていればもっとまともな文章が書けました！！！！ネタならいっぱいあるのに！！！！まぁ全面的に悪いのは私なんですが

　みなさんはぜひぜひ締め切りを大切にしてあげてください。締め切りまであと数秒しかありません。ここまでごらんいただきありｇ

無題

wataru

提出締め切りギリギリになり、急いで書き始めたものの何を書けばいいのか思いつかないので、意味の分からないことを書くとします。

高専に入学してからというものきゅうりをおいしいと感じて、赤点がとりたくないとつくづく思います。今日この頃、きのうは白湯をのみました。とても固いと感じて海を溶かしました。

家に柿の木にあり本当にありがたいと思っている矢先に祖母が家を訪ねてきたので隣人に挨拶をしに行きました。そうしたら隣人の家に人の形をした。それはそうとして明日で花火をしたのですけど久しぶりに友達とさよならができるので果てしなくうれしいと思いたかった。

今もこうやって奇人のふりをしてじダイビングしているとミュータントタートルズな気がしてとても怖くなってきます「寝起きヤシの木」には種多様性とポリコレに配慮されているとありえないフェミニストであり機能も充実していました。「ジョジョ」は「アブドゥル」が最も好きなんですよ。この木なんの木って一目が両全なんですよ。「デルトロ」のピノッキオ見ましたか？いいですよねあれ。長いな。最近ヴィーガンが亡くなりましたよね。飢餓で。どうでもよくないんですけど枕元ってどこなんですかね。久しぶりに友人と会った花火がした。寒くなると熱いし、腐っていくと思うのですよ。あたまが痛くなってきました。もうｔｙほっとがんばるとします。最近「くしゃがら」とか「ドグラ・マグラ」を読んで楽しいと感じています。

積み木てえ楽しいですけどどんなこともないよねってお母さんに言っ

たのですよ。うるさい言って諭されましたけどカチンときてハンマーを握ったんですよ酢飯と一緒にかねとかねがふってきたせいでなんかもっとつかれるよねっておかさんにはなしたんですよ。そしたらお前には光線があってるよすぺ住む野菜やばいな子ｗれ笛田人間関係ってむずいよね、そんなこともないんじゃん。拝読ありがとうございました。

一句

映像の　調子が悪い　ホラーかな　　ZIMMAD

貞子でも出てくるんですかね…

とりあえず　音割れさせて　ナンボでしょ　　JJ1URG

音割れポッターでも好きなのでしょうか…？

赤点が　ひとつ、ふたっつ　みっつよつ　　itu

あるある

あとがき

この度はＤ言語 2023 年度版を手に取っていただきありがとうございました。

今回の Ｄ 言語は内容が盛りだくさんでしたね。おそらく過去一だったのではないのかと思います。しかも全部ためになるような記事だったり面白い記事だったり、作ってよかったなって凄く思います。

自己紹介が遅れました、部長の maru と申します。特にこれといった特徴なんてないのですが、まぁ毎日頑張って生きています。

そういえば、今年度の電算部は部員の人数が過去最高らしいです。30 人くらいかな、とても多いです。大変喜ばしいことなのですが、ここでぶち当たる壁が一つありました。私 1 人で 30 人も管理できるわけがありません。

これに困った私は改革を行います。4 つの班分けをしました。なんとこれがとてもいい。1 班 3〜8 人程度、そして班長を置くことで労力の分散、私はとてもハッピーになりました。もちろん部長としての仕事はやっていますよ。ちゃんとやっています、多分…。

改めまして、ここまで読んでいただき本当にありがとうございました。ぜひ次回のＤ言語をお楽しみにお待ちください！

D 言語 2023 年度版

2023 年 8 月 19 日　　初版

著者　Aozora, ice_adu, itu, JJ1URG, lime, maru, sakaue, UT, wataru, yamazaki, yukidoke, ZIMMAD, イイジマ, 卵食い倒れ
校閲　yukidoke, itu, JJ1URG, イイジマ
表紙　maru
編集発行　群馬高専電算部
Twitter (X)　@GNCT_densan

誤字脱字等ありましたら、Twitter（@GNCT_densan）までご連絡ください。

群馬高専　電算部