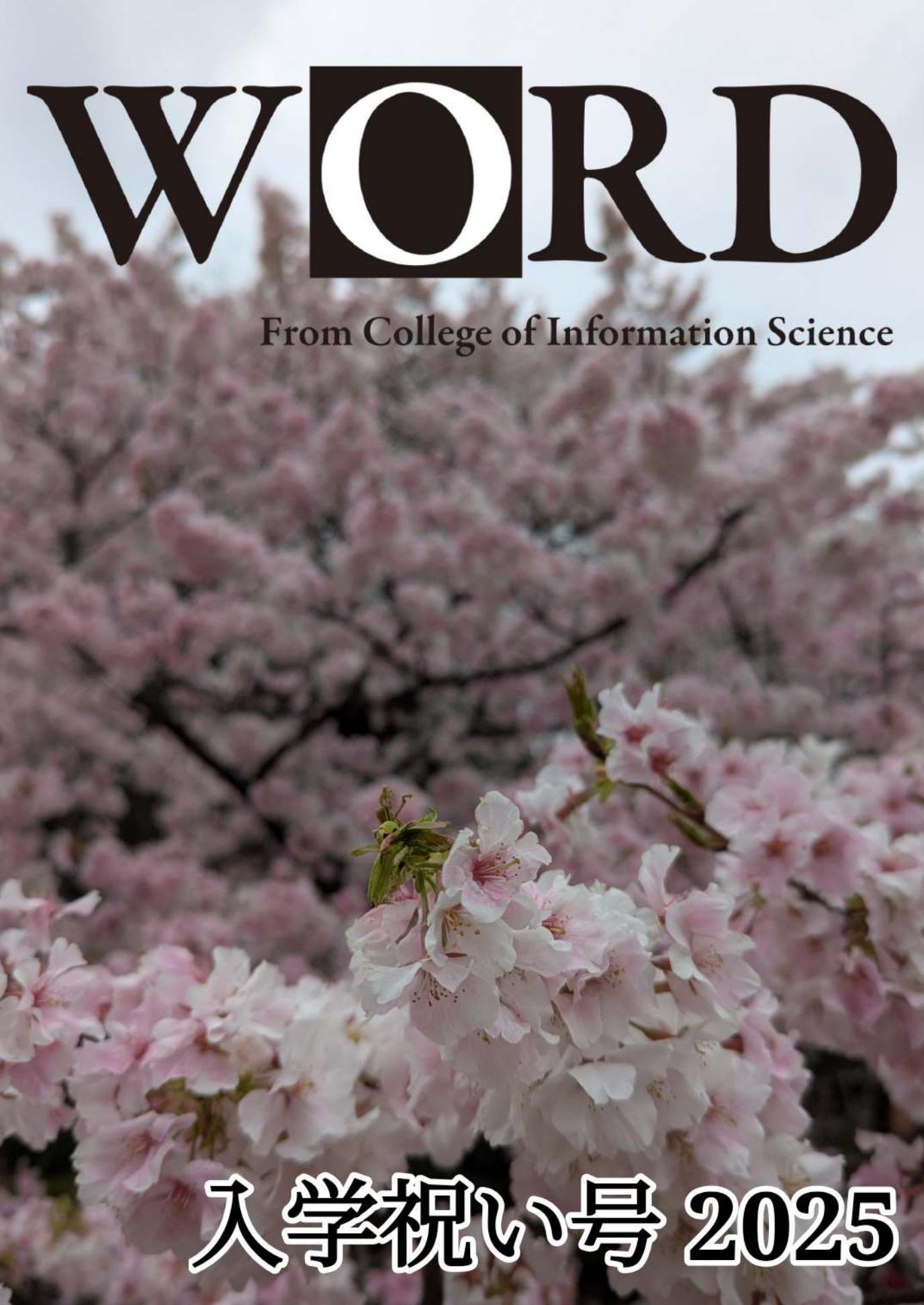
WORD

From College of Information Science



入学祝い号 2025

目次

WebKit のバグを修正した: text-underline-offset の repaint 条件

文　編集部　s7tya

少し前に WebKit のバグを見つけ、PR を送ったら無事マージされました。あまり大したバグではないですが、レビュワーの指摘を踏まえて調べたところ「どうしてこのバグが起きたのか」の話が勉強になったのでまとめておきます。

1　バグの概要と修正

「ホバー時に text-underline-offset を変えようとしても Safari では正しく機能しない」というバグです。どうやったら動くか色々と試行錯誤していると、どうやら以下のサンプルの「OK」の例のように他のプロパティーも一緒に変更すれば正常に動作するようでした。

```
<a href="https://webkit.org" id="ok">OK</a>
<a href="https://webkit.org" id="ng">NG</a>
```

```
a {
  text-decoration: underline;
  text-decoration-skip-ink: none;

  &:hover {
    text-underline-offset: -5px;
  }
}

#ok:hover {
  color: orange;
}
```

ブラウザがどのように機能しているかはほとんど知らなかったのですが、挙動から、再描画をするかどうかの条件に `text-underline-offset` が含まれていないのではないかと予想できたので再描画の判定を行っていそうな箇所を探し始めました。しばらく探すうちに `RenderStyle::changeRequiresRepaintIfText` というピッタリな関数が見つかり、ここに `text-underline-offset` を考慮するよう変更を加えたところ正常に動作したため、PR を送りました。ここからはなぜこのバグが発生したかについて、今の段階での私の理解を紹介します。

2　(前提) レンダリングの処理の流れ

まず前提として、ブラウザでは図 1 のような順で Web ページを描画します。今回のバグに関係があるのは layout と paint の部分で、layout で要素の位置やサイズを定め、その後に paint で実際に結果を描画します。Web ページの内容に変更があった時、それが layout に関わる場合は再度 layout が実行 (reflow) され、そうでない場合は再度 paint が実行 (repaint) されます。

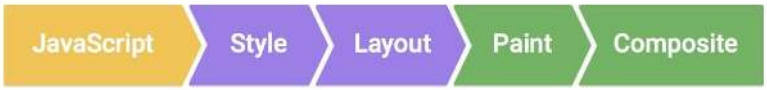


図 1　The full pixel pipeline, illustrated.[*2]

3　下線の位置の変化とトリガされる処理

テキストに下線を追加した状態から下線の位置を移動した場合にトリガされる処理は 図 2 のようになっています。(テキストの周囲に表示されている点線が bounding box をイメージしたもの)

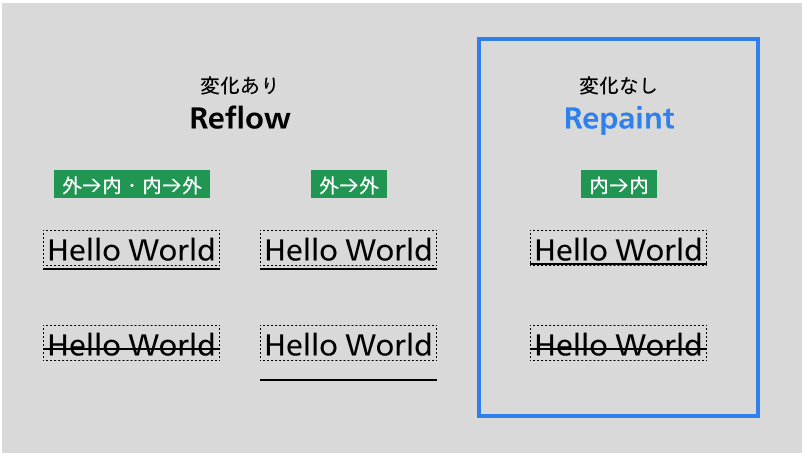


図 2　下線の位置の変化とトリガされる処理の対応図

外側から内側、内側から外側や、外側から外側へ移動した際には下線が移動したことで要素からはみ出している部分が変化し、reflow が実行されます。[*3]それに対し、内側から内側の例でははみ出している部分に変化はありません。そのため、reflow はトリガされず、repaint で処理される必要があります。しかし冒頭で説明したように repaint されるべきかを

[*2] https://web.dev/articles/rendering-performance

[*3] 下線は ink overflow と呼ばれる、要素からはみ出しているが layout に関係ない装飾要素の一つだと紹介されているのですが、WebKit では変更があった場合には reflow を行っているようでした。

判定する処理には `text-underline-offset` が考慮されていなかったため、再描画されずホバーしてもスタイルが変化しなかったようです。

4　おわりに

下線を要素内から要素内に移動させるような状況は確かに多くないと思うのでこれまで気づかれなかったのも理解できます。小さいですが、ブラウザのレンダリングの流れや変更の反映方法について学ぶきっかけになる面白いバグでした。また、WebKit への PR ははじめてだったので専用のコマンドを使って git 操作を行うなど WebKit 独自のシステムに触れることができたのもよかったです。

個人的に Next.js が嘘だと思う理由

文　編集部　みずあめ

1　はじめに

この記事は非常に強い思想が含まれる可能性があります。注意してご覧ください。

1.1　とりあえず Next.js 使うのやめませんか？

現在 LLM の発展により、Cline や V0 といったサービスでとりあえず Next.js を採用するみたいな流れを X 上で多く観測しています。これは本当に正しいのでしょうか？

自分が Next.js を採用する場面は以下の場合のみです。

- すぐに破壊するプロジェクトである
- オンプレでホストする
- 潤沢な資金があり、かつ、小中規模までの発展しか望めない

逆に以下の場合だと採用しません。

- 極力コストをかけたくない
- ユーザー数が小〜大規模まで変わっても柔軟に対応したい

Vercel にデプロイする場合は商用利用の場合、開発人数 1 人につき月 20 ドルかかる上、課金要素が多々あります。[*1]。仮にお金を払ったとしても、様々な制限がかかっています[*2]。

個人的な感想としては、Web における Python なのではと思っています。(要するに、保守性がなく、すぐに破壊するというコンテキストです)

1.2　フロントとバックエンドをくっつけるの本当ですか？

個人的に一番懐疑的なのが、`use client` と `use server` です。オンラインゲーム並みの常時 WebSocket と UDP の通信、それに伴う整合性が必要ならフロントとバックエンドを同時に作るのも理解できるのですが、Web でそれをやるのはオーバーエンジニアリングだと考えています。

1.3　サーバーレスに逆行

先ほどの話と関係してくるわけですが、AWS Lambda や Cloudflare Workers といったサーバーレスアーキテクチャが主流のこの時代に、スケーリングするために一々インスタンスを立てるのも、Docker コンテナを立ち上げるのも環境破壊だと思います。

[*1] https://vercel.com/docs/pricing
[*2] https://vercel.com/docs/limits

1.4 スケーリングとベンダーロックイン問題

Next.js を手軽にデプロイするとなるとどこを使うことになるでしょうか？ Vercel が有力な選択肢の 1 つだと思います。一番の問題ポイントはここです。Next.js で作成したプロジェクトを手軽にデプロイしようとすると、Vercel にベンダーロックインされる羽目になります。OpenNext というものもありますが、破壊的仕様変更が頻繁に起こる Next.js で本当にそれに依存していいのでしょうか？

1.5 おすすめのフレームワーク

ログイン機能等がないただのサイト、ブログなら Astro を使いましょう。

Linuxが動く最小セットのRISC-V CPUを自作する

文　編集部　Arata

1　はじめに

最近、CPU を作ることにハマっています。普段はソフトウェアばかり触っていて、ハードウェアにはあまり興味がありませんでしたが、知人との会話をきっかけに自作 CPU の世界に引き込まれてしまいました。今回、作っていた CPU が Linux を動かせる程度のものになったので、その開発過程を紹介します。

2　作るもの

CPU を自作するには、まず ISA（命令セットアーキテクチャ）を定める必要があります。ISA とは、CPU がどのような命令を持ち、それらはどのようなバイト列で表現され、どのような動作をするのかを定めた仕様です。今回は ISA に RISC-V を採用しました。RISC-V はシンプルで理解しやすい命令セットを持ち、Linux のような OS を動かせる汎用性を備えています。また、コンパイラやシミュレータなどの周辺ツールも活発に開発が進められており、開発環境が整っているのも魅力です。

RISC-V の命令は基本命令と拡張命令から構成されます。基本命令には整数演算や比較、分岐などの基礎的な命令が含まれています。拡張命令には乗算・除算命令、浮動小数点数演算命令など、特定の機能を拡張する命令群が含まれています。

特に Linux を動かすことを目指す場合、以下の命令セットを実装する必要があります:

- RV32I：32bit の基本命令
- M 拡張：乗算・除算命令
- A 拡張：アトミック命令
- Zicsr 拡張：CSR 命令
- Zifencei 拡張：メモリバリア命令

さらに、以下の周辺デバイスも実装する必要があります:

- CLINT：割り込み（特にタイマー割り込み）
- UART：シリアル通信（文字の入出力に使用）

以上の命令セットと周辺デバイスを実装することで、Linux を動かせる最小セットの CPU が完成します。

3　基本命令セットの実装

まず最初に RV32I と Zicsr 拡張から実装します。この段階では「RISC-V と Chisel で学ぶ はじめての CPU 自作」[*1]という本が非常に参考になります。図書館情報学図書館に所蔵されているので、まずは借りて読んでみるとよいでしょう。この本の第 21 章まで実装を進めると、RV32I の一部と Zicsr 拡張が完成します。残りの RV32I 命令も 8 命令程度なので、これまでの実装を参考にすれば簡単に実装できます。

興味があればパイプラインの実装まで進めてもよいですが、のちのデバッグが大変になるのである程度の覚悟が必要です。筆者はパイプライン関連のバグ修正に 1 週間を溶かしました。ベクトル命令や独自命令の実装は省略しても問題ありません。

ちなみに本書の内容が難しいと感じる場合は、情報科学類開講の論理回路や論理システム演習を履修すると良いかもしれません。

4　Coremark を動かす

Coremark[*2]は組み込み向けのベンチマークプログラムです。ですが、ここでの目的はベンチマークを取ることではなく、Coremark を動かすのに必要な CPU の機能を実装することです。いきなり Linux を動かすことを目標にするとデバッグが非常に困難になる可能性が高いので、まずは Coremark を動かすことを目標にします。

4.1　M 拡張の実装

M 拡張は乗算・除算命令を追加する拡張です。Coremark がベンチマークとして乗算・除算を行うので、この実装が必要です。基本的にはすでに実装されている加減算命令と同様の手順で実装できます。命令によって扱う整数が符号ありか符号なしかが異なるので、注意して実装します。

4.2　UART の実装

UART はシリアル通信を行うためのデバイスです。Coremark はベンチマークを行った後に、その結果を文字で出力します。したがって文字を出力する機能が CPU に必要で、そのために UART を実装します。厳密に実装をすると各種レジスタやバッファリングの実装が必要になりますが、Coremark を動かすだけであれば、レジスタに書かれた値をそのまま標準出力に出力するだけで十分です。

ここまでの実装が完了すれば、CPU で Coremark を動かすことができるようになります。

5　Linux を動かす

さて、ここまでの実装が完了したら、いよいよ Linux を動かすことを目指します。この段階では、原因究明の難しいバグに遭遇する可能性が高いです。それらのバグはテストでも発見が難しい場合があり、デバッグには多大な時間がかかるでしょう。

[*1] https://gihyo.jp/book/2021/978-4-297-12305-5

[*2] https://github.com/eembc/coremark

そのため、シミュレータを使ってデバッグを行うことをお勧めします。Spike や QEMU などのシミュレータと動作を比較しつつ、挙動が異なる場合には修正する作業を繰り返します。ただし CPU の状態をすべて比較するのは大変に時間がかかるので、プログラムカウンタの値や注目したいレジスタの値のみを出力し、それをプログラムで比較するようにすると効率的です。

なお Chisel で実装した CPU を動かす場合、ビルドに若干時間がかかりますが、Verilator を使うと実行速度が高速で便利です。

5.1 Linux のビルド

自作 CPU の上で Linux を動かすためには、自分で Linux をビルドする必要があります。デフォルトでは Linux は CPU に多くの機能が実装されていることを期待しますが、自作 CPU にはそれらの一部しか実装されておらず、機能不足で動作しないためです。したがって、Linux のビルド設定を変更して機能を減らし、自作 CPU でも動作するようにします。

Linux のビルドには Buildroot[*3]を使うことを強くおすすめします。Buildroot は Linux のビルドを一部自動化するツールで、ユーザーが作成した設定に従ってビルド作業を行います。`qemu_riscv32_nommu_virt_defconfig` というプリセットの設定をベースにするとよいでしょう。ファイルシステムやデバイスツリー blob は Linux のイメージに埋め込む設定にしておくと、読み込む必要のあるファイルが 1 つだけになるので便利です。そのほか、mini-rv32ima[*4]の設定を参考にしてもよいでしょう。

5.2 A 拡張、Zifencei 拡張の実装

A 拡張はアトミック命令を、Zifencei 拡張はメモリフェンス命令を追加します。

嬉しいことに、今作っている CPU はシングルコアであるためこれらの実装は非常に簡単です。アトミック命令はメモリからの読み取り、演算、メモリへの書き込みを 1 命令で行います。ここまでに実装した命令はメモリからの読み取り、ないしはメモリへの書き込みを同時には行わないはずなので、アトミック命令のためにメモリへのアクセスを 1 段増やす必要があります。

メモリフェンス命令はメモリアクセスの順序を保証するための命令です。パイプラインを実装していなければ NOP 命令（何もしない命令）と同じで問題ありません。

5.3 CLINT の実装

CLINT は割り込みを扱うデバイスです。Linux はタイマー割り込みを使って実行中のプロセスを切り替えるため、この実装が必要となります。より具体的には、以下の機能が必要です。

- `mtime` レジスタ
- `mtimecmp` レジスタ

[*3] `https://buildroot.org/`

[*4] `https://github.com/cnlohr/mini-rv32ima/tree/master/configs`

- タイマー割り込み

`mtime` レジスタは現在の時刻を保持するレジスタで、`mtimecmp` レジスタは次の割り込みを発生させる時刻を保持するレジスタです。タイマー割り込みは `mtime` レジスタが `mtimecmp` レジスタを超えたときに発生し、割り込みハンドラが呼び出されます。

以上の機能の実装が正しく行われれば、Linux を動せる最小セットの CPU が完成し、実際に Linux を動かすことができるようになります。とはいえ、筆者がそうであったように、CPU をバグなしに正しく実装することは難しい作業です。Linux のイメージを読み込ませたが、そもそも何も出力されなかったり、カーネルパニックが発生したり、あるいはパニックするがログが出力されなかったりといった問題が発生することがあります。そのような場合は、根気強くシミュレータとの比較・デバッグを行い、原因を突き止めることが重要になります。

6　さらなる拡張

Linux が動く最小セットの CPU が完成したら、もちろんさらに実装を進めることもできます。

6.1　FPGA で動かす

ここまでは実装した CPU をシミュレータで動かしてきましたが、結局既存の CPU の上で動いているので、ハードウェアを触っている感覚がありません。そこで、FPGA を使って自作 CPU を動かすというのも面白いでしょう。FPGA は論理回路をプログラムできるデバイスで、調べると先人が自作 CPU を FPGA で動かしている例が多く見つかります。情報科学類開講の論理システム演習で高価な FPGA を扱えるほか、情報科学特別演習でもハードウェアに詳しい先生方と話せるので、そういった機会を活用すると良いと思います。

6.2　MMU の実装

現在の Linux は MMU がないため NOMMU で動いており、メモリ保護の機能がありません。これはつまり、あるプロセスが他のプロセスのメモリを読み書きできるということで、現代の OS としては非常に危険です。MMU を実装することで、これを防ぐメモリ保護の機能を実現できます。

6.3　CPU の検証

RISC-V では、CPU の検証を行うための RISCOF[*5]と呼ばれるフレームワークが開発されています。せっかく作るのであれば RISC-V もどきではなく、RISC-V の仕様に準拠した CPU を作りたいという方は、こういったフレームワークを使って CPU の検証を行うと楽しめると思います。

[*5] `https://github.com/riscv-software-src/riscof`

6.4 CPU の高速化

現代の CPU はパイプライン化やスーパースカラ化、アウトオブオーダー実行といった高速化のための技術を多く実装しています。パイプラインや分岐予測は高速化技術の中では比較的実装が簡単であり、自作 CPU にも実装しやすいでしょう。

7 おわりに

以上が自作 CPU で Linux を動かすための大まかな手順です。そのほか、自作 CPU を進めるにあたって参考になる資料を以下に挙げます。

- RISC-V の仕様: `https://riscv.org/specifications/ratified/`
- 「マイクロプロセッサの設計と実装」の実験資料: `https://exp.mtl.t.u-tokyo.ac.jp/2024/b3exp/-/wikis/home`
- 「スタンダードセル向け LSI 設計」の実験資料: `https://www.ddna.cs.tsukuba.ac.jp/lecture/lsi_design/index.html`
- 無から始める自作 CPU: `https://vlsi.jp/LetsMakeCPU.html`

あなたもぜひ、自作 CPU を始めてみてはいかがでしょうか。

オタクへのキックスタート

文　編集部　whatacotton

新入生の皆さん、ようこそ情報科学類へ！
私は whatacotton と言います。今日はオタクへのキックスタートと題して、よりコアなオタクへの入り口としてディープな世界に飛び込もうとしている方に向けて、簡単なイントロダクションをさせてもらおうかと思います。ちなみに私は Web とか Linux とか FPGA のオタクです。

1　サブ PC を買う

実験用 PC を買いましょう。実験用 PC を買うことによって気兼ねなくシステムの中をいじり回すことができます。単純に開発機として持っておくのもありです。一番の利点は、サブ PC の環境がいくら壊れても、メイン PC があれば授業には支障が出ないところです。さらに Linux だと Web 版 Office しか使えないので、授業で強制的に Office を使わないといけないときにも実験用 PC として使い分けていればなんとかすることができます。一応 LibreOffice などの代替品も存在しますが、私はあまり使ったことがないです。また、入学後は全学計算機にアクセスすることができるため、これを遠隔で操作することができる RDP(Remote Desktop Protocol) を使用して Office を使うという選択肢もあります。

実験用 PC を買うと、より柔軟に自分に合わせた環境を作ることができてしまうので、正直そっちがメイン PC となってしまうことが多いのですが、フォールバック先があるのとないのでは全然安心感が違います。中古 PC の相場は高くても 5 万円くらいのものであれば十分だと思います。きっと入学祝いで買ってもらうような PC よりは安いはずです。
スペックは、ストレージは SSD で、CPU は `Intel Core i5 7th Gen`辺り、RAM は 8GB、ゴリゴリ GUI を使うなら 16GB あれば望ましいといったところでしょうか。さらに、このような PC はグラフィックボードなどが載っていないので、大抵が軽いです。家と大学間を行き来するとなると、軽い PC のほうが絶対楽です。最初は慣れるまで 2 枚持っていかないといけないですが、実験用 PC がメインになっていくにつれて、最初に持っていた PC は家で文鎮と化します。

2　OS を入れてみる

2.1　そもそも OS を入れるとはどういうことなのか？

私も最初はどういうことなのかよくわからなかったのですが、実は Windows が動いているその PC は必ずしも Windows しか動かないわけではないのです！ Macbook でも Intel チップが入っている PC なら Windows が動いてしまうらしいです。このように PC 本体とその上で動く OS は必ずしも一対一ではないのです。

自作 PC を作ったことがある人は見たことがあるかもしれませんが、起動時に特定のキーを押すことによって入れる UEFI(古い PC なら BIOS) というシステムにアクセスすることによって、どこから OS を起動するかを指定することができます。どこからというのは PC に接続されている HDD・SSD や USB ポートに刺した USB からなどです。つまり USB に OS が入っていれば USB から OS を起動することもできます。

OS をインストールする流れ

Windows でも Linux でも OS をインストールする流れは同じです。

1. 入れたい OS のインストールイメージをダウンロードする (だいたい `.iso`ファイル)
2. 中身を空にした USB を用意する (空にしなくても良いが、データが全部消えます)
3. `balenaEtcher`や、Windows なら `rufus`、Linux なら `dd`コマンドを用いて USB にイメージを書き込む
4. UEFI(BIOS) に入り起動順序 (Boot Order) を USB が HDD などよりも先に起動するように変更する
5. USB から OS が立ち上がったことを確認してインストールガイドに従ってインストールする

以上になります。

基本的に OS をインストールする場合は、インストール先の媒体に保存されていたデータが全て消えてしまいます。よって、OS を引越しする前に大事なデータを別の場所に避難させてあげましょう。

ちなみに後述しますが、Arch Linux などの一部の OS では 5 番の工程がコマンド実行での手作業になります。

2.2 君も Linux を使ってみないか

では実験用 PC に何の OS をインストールするのか。そのまま自作 OS などに極振りしてもよいとは思いますが、やはりここは Linux を入れることをお勧めします。

皆さん Linux はご存知でしょうか？ Linux は Windows、macOS に並ぶ第三の OS であり、オタクにとってはとてもぴったりな OS なんです。

Linux は色々なディストリビューションがあります。ディストリビューションとは、Linux を実用的に使うことができるように電源管理や Linux を起動するシステムまわりをいい感じにしてくれるものです。Linux はそれ自体は最低限の機能を提供しますが、ディストリビューションがしてくれているような最適化はしてくれません。頑張れば Linux の環境をディストリビューション非依存で構築することもできます[*1] が、あまり現実的ではありません。

ここで有名なディストリビューションをいくつか挙げます。

[*1] `https://www.linuxfromscratch.org/`

- Debian
- Ubuntu
- Fedora
- Arch Linux
- Manjaro
- EndeavourOS
- CentOS
- AlmaLinux
- Rocky Linux
- NixOS

  どれが優れているかという話を始めると宗教戦争になるのであまり言及したくないですが、初心者なら Ubuntu が良いのではないかと思います。ちょっとカッコつけるなら Fedora か Manjaro あたりでしょうか。私は Arch Linux から始めましたが、初動で大分手こずったので頑張りたいなら止めないがお勧めはしないといった感じです。

Ubuntu は簡単にインストールすることができ、Windows ユーザーでも簡単に使うことができると思います。私は Ubuntu を使う際には KDEというシステムが好きなのですが、デフォルトだと Gnomeというシステムが入っているので、派生版の Kubuntu というディストリビューションを使っています。

Linux のメリットは以下のようなものがあります。

- 先述したようにディストリビューションがたくさんあるため、自分好みのディストリビューションを選ぶことができる
- パッケージマネージャというアプリケーションを管理してくれるシステムを用いることで、コマンド一つで大概のアプリケーションをインストーラなしで入れることができる。アンインストールも同様にコマンド一つでできる
- GCC や Bash など、日々開発する上で重要になってくるアプリケーションが使える
- **無料** (一部ディストリビューションでは有料サポートをしているものもあります)
- 画面の見た目を好き勝手にカスタマイズできる (Unix Porn で検索してみてください)
- 基本的になんでもできるため、コンピュータに対する理解が深まる (エラーがとても詳しく出力されたり、エラー文をそのまま検索するだけで解決できたりする)

またデメリットは以下が挙げられます。

- Microsoft Office など macOS、Windows 向けの一部ソフトが使えない (ブラウザ版なら使えますが機能が足りないこともあります。)
- コミュニティによって開発されているため挙動が不安定なことがある
- システムへの理解度を高めようとすると学習コストが高い
- 当然のようにたくさんエラーが出るため、開発そっちのけで OS の復旧や原状回復

に回らないといけない時がある

2.3 OS を入れなくても

Windows の場合は WSL(Windows Subsystem for Linux) と呼ばれるアプリケーションを用いることで Linux を Windows 上で使用することができます。現在は WSL 2 が一般的です。Windows を消したくはないが、Linux を使ってみたいという人はぜひ WSL から経験を積んでみてください。

またインストーラの USB に入っている Linux も通常の Linux と何ら変わらないで、インストーラだけ作って USB 越しで Linux を楽しむというのも手段としてはあると思います。

3 シェルを触る

Arch Linux だと、最初のインストールのタイミングからターミナルを触ることになるのですが、Ubuntu などは触らなくてもなんとかなったりします。しかし、Linux を触る以上はターミナルに慣れていく必要があります。一種のシェルなのである Windows のコマンドプロンプトを触ったことがある人は少なくないと思いますが、このあたりの話は少しややこしくなるので割愛するのですが、Linux では Bash、Zsh と言ったようなシェルというアプリケーションをターミナルというアプリケーション越しで実行するといった感じです。macOS でも「ターミナル」というアプリケーションを起動することで Zsh にアクセスすることができます。macOS の強みとしては Linux で開発されたシステムのほとんどを使うことができるというものがあります。これは残念ながら Windows では WSL(Windows 上で Linux を使えるようにしたアプリケーション) などを介さないとできないです。

シェルは基本的に開発を行うのに不可欠なものなので、この先どういうものを作りたくても触ることになると思います。もちろん授業でも触ることになると思います。

最初は文字を打つ画面だけを提示されることになるので戸惑うこともあるかと思いますが、慣れていくと、それが当たり前になっていきます。

どのようなコマンドを打てばよいかは ChatGPT などに聞くと早く解決できて良いです。

また、シェルのようなアプリケーションに触れることで、その仕組みに興味を持つきっかけになったりもすると思います。

ちなみにシェルにもたくさんの種類があって、一番基本的な Bash、その拡張版とも言われる Zsh、python が使える Xonsh などがあります。

3.1 シェルにプラグインを入れる

デフォルトのシェルは無機質な感じがするのですが、このシェルをカッコよく装飾することもできます。ここにこだわる人がたくさんいるのですが、長く使っている人で、最低限の装飾だけを施している人もまた多い印象です。このような装飾はシェルにプラグインを入れる形で導入することができ、Zsh ならプラグインの導入を支援してくれる Oh My Zsh が有名です。シェルのプラグインもたくさんあって、打ったコマンドの履歴を参照してコマンドを補完してくれたり、実行できるコマンドなら緑に、できないコマンドは赤に表示

してくれるものがあったりします。

3.2 シェルをカッコつける

```
bash-5.2$
```

たとえばバニラの bash でのターミナルでの表示はこんな感じです。

しかし、設定を触るとこんな感じの表示ができたりします。以下は今記事を書いているディレクトリでのターミナルでの表示です。

```
iwai2025 on   articles/kickstart [?] took 3m57s
❯
```

このような設定をすることで今いるディレクトリや、Git を使っている場合はそのブランチ名を表示することができます。`took 3m57s` と表示されているのは、一つ前に実行されたコマンドがどれくらいの時間実行されていたかを示しています。Bash を例示するために実行していた時間が 3m57s だったようです。他にも現在使っている PC のユーザや現在時刻、PC のホスト名などいろいろなものを表示することができます。

このような設定ができるシェルのプラグインはたくさんあり、私は Starship を使っています。また、組版の関係上文字化けしてしまうかもしれないのですが、articles の左に書いてある文字は Nerd Fonts と呼ばれる記号がたくさん収録されているフォントを使用しており、この Nerd Fonts もたくさんの種類があってどの Nerd Fonts を使うかもオタク心がくすぐられる要素の一つです。

4 おわり！

今回はこれで終わりです。大学の授業が始まる前にできる準備としてこれだけやっておけば、その先の開発ライフであまり困ることはないでしょう！

実際に Linux をやってみたいが色々不安な点が多く触るのが大変そうだと感じている人は、私が相談に乗るので、X(@whatacotton) や Discord(@whatacotton) からでも連絡を取ってみてください。

電工二種を取る

文　編集部　北野尚樹 (@puripuri2100)

1　はじめに

新入生の皆様、御入学おめでとうございます。春から大学生になり、新しいことを始めようと思っている方も多いと思います。そこでおすすめするのが「**第二種電気工事士**」という資格の取得です（通称「**電工二種**」）。

情報科学類で電子機器を扱う中で自然と発生してくるのが、電気工事の需要です。サーバールームの中の配線の取り回しを良くしたくなったり、部屋に簡単なスイッチや電源類を増設したくなったりしてきます。このときに電気工事士の資格があれば必要な知識や技能を持った状態で臨むことができ、適切な選択を選ぶことができます。

電気工事に関連する資格はいくつかありますが、その中でも最も取得しやすいのが一般用電気工作物と小規模事業用電気工作物の 2 種類のみを工事範囲の対象とする第二種電気工事士の資格です。第一種電気工事士や電気主任技術者などの資格もありますが、これらはより法令の知識や技能を身につける必要があり、ハードルが高いです。

それでは早速電工二種を取りに行ってみましょう。

2　申し込む

電工二種の試験は 2 段階行われます。まず、法令の知識や基本となる電磁気の知識、電気工事を行うための配線などの知識が問われる学科試験があります。これを突破すると、ケーブルを実際に切断・加工してコンセントやスイッチ同士をつなぐ課題をこなす技能試験を受けることになります。技能試験を突破すると晴れて合格となり、免状を発行することができるようになります。

学科試験は特定の日時に試験場に集められてマークシートに回答する筆記方式と、自分の都合がつく日に全国に設置されている試験会場に行ってパソコンで回答を行う CBT 方式の 2 種類があります。

技能試験は特定の日時に試験場に行き、実際に手を動かして工作物を作る必要があります。茨城県の会場は筑波大学か水戸の 2 択らしいです。筑波大学を引くことができればギリギリまで寝ることができます。祈りましょう。博打が嫌いな人は東京など他の都道府県の会場に申し込むのもありだと思います。

電工二種は年に 2 回試験が行われます。電気技術者試験センターの当該ページ[*1]に日程の情報が記載されています。今年の上期日程はすでに締め切られていますが、例年 4 月頭に申し込みが締め切られ、学科試験が 5 月頃にあり、技能試験が 7 月中旬のようです。来

[*1] https://www.shiken.or.jp/construction/second/

年以降受けるときの参考にしてください。今年の下期試験は 8 月 18 日から 9 月 14 日までの間に申し込むようです。技能試験が秋 AB 期末試験の一週間前にありますが、たぶん大丈夫です。息抜きとして工作物を作りましょう。楽しいです。

申込みは、電気技術者試験センター用のアカウントを作るところから始まります。このアカウントは合否結果を早めに知ったり受験開催地や受験番号を思い出したりするときにも使います。アカウントを作成することができたら、必要な情報を入力して申込みをし、支払いを行います。支払いを済ませて申込みが完了すると、筆記試験のを CBT 方式にするかどうかが選べます。もし筆記方式の受験日程の都合が悪い場合はこちらから CBT 方式の日時と場所を選択して申し込みましょう[*2]。

3　学科試験対策

学科試験では次の 4 項目について 4 択マークシートの問題が計 50 問が出題され、全体を合算して 30 問正解すれば合格する、というのが例年の流れです。概ね 4 項目のうち 1 項目半くらいは落とせます。苦手な分野は切り捨ててほかを完璧にする戦略もありです[*3]。

- 機器・材料・工具・施工方法
- 検査方法・法令
- 電気の基礎理論
- 配線図

このうち、機器・材料・工具・施工方法・検査方法・法令については単純な暗記であり、問題についても多くが過去問の使いまわしですので、一問一答系のアプリを回しまくることで対策することができます。定数値を答えさせる問題や、与えられた条件に合致する機器の選定、写真を見て機器名を選択する問題が多いです。

電気の基礎理論は高校の理系物理をやっていて参考書を読んで思い出すことができれば概ね正答できるはずです。理系物理を完全に忘れている場合は簡単な回路の問題だけを解いてあきらめましょう。

配線図とは、問題用紙に描かれた配線図の中で指定された箇所の芯線数や使用するコネクタの種類と数、使用する施工工具やパイプなどを答えるものです。配線図を描く練習を積み重ねることで芯線数やコネクタの種類などが分かるようになってきます。この技能は技能試験でも使うのでやっておいて損はありません。また、適切な施工工具やパイプなどを答える問題では記号の読み方と施工方法と材料の組み合わせを覚えておけば、ある程度正解できるようになってきます。

電気の基礎理論と配線図の問題は過去問と問題集を活用し、何度も演習するべきでしょう。問題集は世の中に数多く出版されていますが、結局は数をこなすことが大事なので適

[*2] 自分はライブの翌日に日程を入れてしまい、ライブ会場で泣きながら対策本をやる羽目になりました。日時は前後に余裕のある日を選びましょう。

[*3] 実際に自分は電磁気の基礎理論が無理だったので捨てています。

当にレビューを見ながら選ぶと良いと思います[*4]。重要なのはやることです。

4　技能試験対策

技能試験は事前に提示されている 13 個の課題の中から 1 つが出題されます[*5]。課題は単線図と呼ばれる簡略化された図面と必要な長さのケーブルと機器から、図 1 のような**工作物**を作り出すものになります。

図 1　工作物の例

最もスタンダードな対策法は工具と練習のセットを一式買い[*6]、事前提示課題を見ながら複線図を書いて解説動画を見ながら工作物を作る[*7]のを繰り返すことです。

実際の作業方法や細かい解説については解説動画が詳しいですが、自分なりにやってみてわかったことについて共有します。

- 手袋をつけないと作業中に指の肉を切ってしまうので、手の大きさにフィットした作業しやすい手袋はしたほうが良い
- ランプレセプタクルなどのよく作る部分については端材を使って繰り返し作ることで慣れておくと良い
- 腰袋などの道具をまとめられるものを買っておくとすぐに道具を取り出せて便利
- 手の大きさなど、巻き尺を使うことなく大まかな長さを測れる手段を用意しておくと良い
- あると一発アウトになる「**欠陥事由**」の基準は練習のときから厳密に確認しておくと本番後に無駄に不安にならなくて済む
- VVR の被覆を剥くのは結構難易度が高いので練習すると良い

工具と練習セットの一式が高すぎるのでケチりたいと考えている方には、ケーブルと圧

[*4] 自分はこれを使いました：早川義晴. 電気教科書 第二種電気工事士［学科試験］はじめての人でも受かる！ テキスト＆問題集 2024 年版. 翔泳社, 2024.
[*5] この事前に提示されている課題は概ね毎年共通ですが、たまに変更されます。
[*6] https://www.hozan.co.jp/corp/denko2/pg/1tool/
[*7] https://www.hozan.co.jp/corp/denko2/pc/02020/

着スリーブと必要な工具のみを買い、器具は既に受験済みの人から借りるという方法があります[*8]。工具は次の物（図 2,3）があれば最低限何とかなります。その中でも VVF ストリッパーは図 2 のようにストリップ以外にもスケール・カッター・ペンチなども一本で兼ねられる便利なものがあるので、これを使いましょう。

- VVF ストリッパー
- 圧着工具
- プラスドライバー
- マイナスドライバー
- ウォーターポンププライヤー

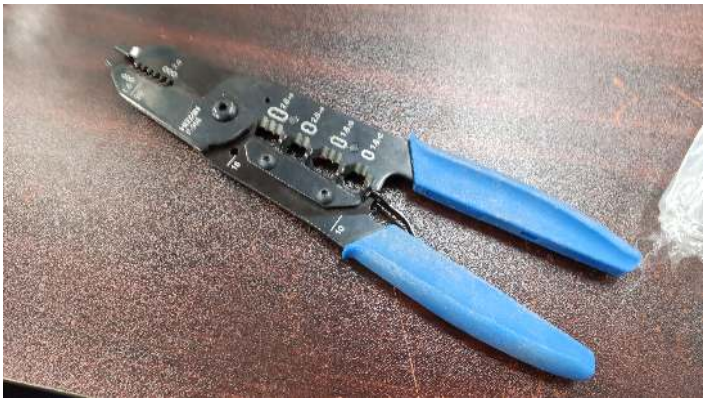


図 2　VVF ストリッパー

図 3　圧着工具・ドライバー・ウォーターポンププライヤー

[*8]自分から借りたい方は puripuri2100@gmail.com や https://x.com/puripuri2100 にメッセージを送ってください。

5　免状発行

試験に合格しても免状が無ければ意味が無いので発行していきましょう。免状の発行は住民登録されている都道府県が窓口になります。例えば茨城県民であれば**茨城県 防災・危機管理部 消防安全課 産業保安室**という所に必要なものを郵送して待っていれば免状が届くことになります。

茨城県で受験して茨城県に免状発行の申請を行う場合はとても簡単です。まず、試験場で渡される**電気工事士免状交付申請書**という書類に必要事項を書きます[*9]。そして警察署や市役所にある**収入証紙売り捌き所**で収入証紙を指定の金額分を**現金**で購入します[*10]。そして身分を証明できるもののコピーと合格通知書[*11]と一緒に封筒に入れて前述の窓口宛てに郵送するだけです。

他の都道府県で受験して茨城県に申請する場合は試験の時に渡される申請用紙が茨城県の物ではないので、茨城県のホームページから適切なものをダウンロードする必要があります。それ以降の手続きは上記の物と共通です。

他の都道府県に申請する場合は必要な書類や窓口が色々とあるようです。頑張って調べてください。

6　おわりに

できることなんてあればあるほど良い。

[*9]無くしても茨城県のホームページからダウンロードできます。https://www.pref.ibaraki.jp/seikatsukankyo/shobo/sangyo/info/sangyohoan/denkimenjyo/mokuji.html

[*10]手数料など様々な事情があるようで現金オンリーです。

[*11]無くすと試験センターに再発行申込書を郵送する必要があるらしいです。https://www.shiken.or.jp/shiken/issue/

お絵描きしよう！

文　編集部　mikio

1　お絵描きへの誘い

情報科学類というのは他学類の方々に比べオタク[*1]が多い[*2]と感じております。そのような方々におかれましては、「自分の好きな○○の絵、描いてみたいな～」とか思ったことも 1 度はあるでしょう[*3]。

しかし、絵を描くというのはなかなかにハードルが高いものです。絵心がないな～とか、昔一瞬描いて絶望して諦めてしまったとか、そもそも何を用意して何から始めればいいのか、など悩みは尽きません。そのため、この記事をもっと気軽にお絵描きを始めるための足掛かりとしてもらうことを願っております。

2　環境構築

何事においても一番最初に立ちはだかる壁であり、最もつまづきやすい場所であるところの環境構築から説明していきたいと思います。

お絵描きの環境というのは大まかにアナログ・デジタルの 2 種類に分けられ、まずはそれぞれのメリット・デメリットを解説します。

2.1　アナログで描く

アナログで描くというのは従来の絵の描き方、皆さんも美術の授業で必ず行ったであろう、そう、つまり物理的な紙に直接描いていくものです。

アナログの利点として、次の 4 点が挙げられます。

1. （おそらく）デジタルよりも手軽で、紙とペンさえあればどこでも描ける。
2. 簡単なセットで始めるのであればかなり安価で始められる。
3. 物理的な実体があるため、完成後の幸福感、所有感が大きい。
4. 「アナログ」な良さがある（紙質、インクの掠れ、ムラなど）。

1 番目の手軽さというのはアナログの大きな特権でしょう。デジタルでは最低でも iPad と Apple Pencil などを持っている必要がありますが、アナログであればその場にある紙とボールペン、シャーペンさえあればそこがアトリエです。このことは 2 番目にも関わってきます。

[*1] パソコンオタクに限らず

[*2] 諸説

[*3] 2 回目の諸説、ショセ・ツー

しかし、アナログにも問題点があります。主に 3 点、

1. 完璧に揃えようとするとかなり高価。
2. セットを持ち歩くとかさばる。
3. 絵の操作という面でかなり制約が大きい。

1 番目、2 番目に関しては先ほどのメリットとは真逆で、最小要件こそかなり低いものの推奨要件を満たそうとすると遥か高いものが求められるという点です。

例えば、コピック[*4]を揃えようとすると 1 本 200〜300 円、それを 30 本以上集める必要があるため余裕で 1 万円を超えてしまいます。

これは絵の具＋筆でも一緒で、*Pay to Win*[*5]なところがあります。色鉛筆であればかなり抑えることができますが、相当な手練れでない限り色鉛筆で目的の塗りをすることは難しいでしょう。

3 番目の制約についてですが、デジタルでは線の消去・キャンバスの回転・パーツの移動などがかなり手軽にできる一方、アナログではそのようなことが一切できません。

線を消すのであれば消しゴム・修正液を使い、回転は紙自体を回し、パーツの移動に至っては不可能です（描き直しをしない想定）。消しゴムで消したら紙がグチャグチャになってしまった経験は誰しもあるでしょう。ペンを持ち直すのもかなりの手間になります。

しかし、これらのデメリットを打ち消すほどにアナログの「始めやすさ」というのは大きいです。この記事を読み終わって絵を描きたくなったら何かペイントソフト[*6]をインストールする前に紙とペンと手に取り、何か描いてみましょう。

2.2 デジタルで描く

先ほどアナログについて色々述べましたが、実は筆者はアナログで描いた経験がほとんど無く、根っからのデジタル絵描きでした。

今でこそネット上で見られる絵、デザイン、漫画などのほとんどの絵はデジタル中心で描かれているものが多いですが、それは何故でしょうか。

「アナログで描く」と同じように、デジタルのメリット・メリットをまとめてみたいと思います。

デジタルのメリット：

1. 細かな操作が楽にできる。

[*4] 色塗り用のマーカーのようなもの
[*5] お金を注ぎ込めば注ぎ込むほど良い絵が描きやすい
[*6] アイビスとか、クリスタとか

2. データの共有が容易。
3. ツール類が集約されている。
4. 初心者でもある程度自分の理想に近づけやすい。

細かな操作については先ほども述べましたが、絵に対する修正力というのはアナログとデジタルで雲泥の差があります。3 番目もほぼ同じなのですが、細かな調整がしやすい、狙った色を簡単に置ける、線が綺麗に引けるなど簡単に「それっぽく」見せることができます[*7]。

初期投資さえしてしまえば全てのツールが使い放題なので、場所も圧迫せず試したい描き方がすぐに使えることも大きいです。

そして、2 番目のデータの共有です。アナログでは紙の直接手渡し・スキャンして取り込むなどしなければデジタルデータとして残すことは不可能ですが、デジタルイラストはそれ自体がデジタルデータなので共有・保存が楽です。絵が劣化することもないのはデジタルの最大の特権とも言えるでしょう。

次に、デメリットです。

1. 初期投資が高い。
2. アナログで描いてきた人は慣れない可能性がある。
3. 未だにデジタル反発派がいる。

デジタルの最大のデメリットは、やはりその初期投資の大きさです。

PC をすでに持っている人であればある程度抑えられますが、それでもペンタブ[*8]やタッチペンを揃えるとかなりの額になります。機材に加え、ソフトも必要になることも忘れてはいけません[*9]。

アナログ→デジタル移行勢はデジタル特有の画面の滑らかさというのも気になるかもしれません。長時間描いていると目が疲れてくることもあります。

3 番目はただの愚痴でデジタルのデメリットでもなんでもないので気にしないでください。

さて、ここまで両者の特徴について長々とまとめてきましたが、筆者としてはデジタルで始めてみるのをお勧めします。情報科学類であれば PC は基本持っているでしょうし、iPad も持っている人も多くいるでしょう。一からアナログ機材を揃える必要もありませんし、イ

[*7]逆に言えば、それっぽくとも初心者とプロの差が顕著に出るとも言えます。
[*8]ペンで描けるタッチパネルのようなもの。あとで紹介します。
[*9]有料・無料のものがあります。これも後述します。

ラストを描いたら Discord[*10]や Twitter[*11]で自慢することもできます。

次の章からはデジタルイラストに焦点を当てて、ソフトの選び方や実際の描き方について解説していきたいと思います。

3　機材の選び方

環境構築でもちょこっと触れましたが、デジタルイラストを描く方法としては主に 2 通りあり、

- PC + ペンタブ、液タブ
- （iPad or スマホ）+ Apple Pencil

が主流です[*12]。

3.1　PC

1 番目の構成ではまずパソコンがないと始まりません。情報科学類であればある程度は大丈夫ですが、メモリは多めに積んでおくといいでしょう[*13]。OS は Windows, Mac どちらでも良いですが Mac の方がメモリ効率が良いので動かしやすい…かもしれません。

Linux ユーザーはペイントソフトを Wine 経由で動かす[*14]か、Krita を使用する[*15]といいと思います。

そして、肝心のタブレットについては、

- 液タブ（液晶タブレット）
- 板タブ（板タブレット、ペンタブ）

の 2 種類に分けられます。

液タブはタッチペン対応のサブモニターのようなもので、手元の絵を見ながらアナログさながら描けるため PC で描くとなるとこちらが主流となります。しかし、次の板タブに比べると値段が跳ね上がるため予算と相談になります。

板タブはその名の通りモニター機能のないただの板で、板の上でペンを動かすとそのペン先に追従してメインモニターのカーソルが動くものになります。手元には何も写っていないため、最初は慣れないかもしれませんがいつかは慣れます。

[*10]筑波大生御用達のチャットツール。サークルなどの情報共有は基本 Discord で行われます。
[*11]新 X。
[*12]たまに指で描いたりマウスを使っている猛者もいますが、少数かつ自分が未経験のためここでは省きます。
[*13]Windows 11, RAM 16GB だと大きめのキャンバスでカクつくことがあります。
[*14]`https://forum.mattkc.com/viewtopic.php?t=336`
[*15]WORD54 号「WORD 編集部は除草剤を撒いていません号」、”Linux で絵を描いてみよう”を参照

さらに、液タブと違い下を向く必要がないため、手元でペンを動かしながら正面のモニターを見ることで姿勢良くお絵描きできます。ちなみにですが、ポケカのイラストなどを担当されているさいとうなおき先生は板タブ派だそうです。

メーカーは液タブ・板タブどちらも Wacom, XPPen, Huion あたりのものを買うと安心です。Wacom は日本メーカーで信頼性がありますが、かなり高いです。筆者のデジタル歴は One by Wacom（板タブ）, XPPen Artist 13.3 Pro（液タブ）, XPPen Artist 24（液タブ）と XPPen 激推しですが、Wacom に比べてかなり安くかつ性能も申し分ないのでおすすめです[*16]。

液タブや普通のモニタによっては色味にクセがあることもあるので、本気でプロを目指したい || コミケなどで印刷したいのであれば色校正用のモニタがあると安心ですが、話すと長くなってしまうのでここでは割愛します[*17]。左手デバイスもあるとキーボードショートカットを覚える必要がなく制作スピードもさらに早くすることができるのですが、これも割愛します[*18]。（気になったら TABMATE とかで検索してみてください。）

3.2 iPad, スマホ

どちらかを持っていればタッチペンさえあれば始められるため、気軽に始めるならこの構成もおすすめです。

PC に比べ持ち運びがしやすいため、アナログに近い感覚で移動が可能です。

4 ソフトの選び方

この章ではお絵描きに欠かせないソフトの種類、選び方について解説していきます。

1. CLIP STUDIO PAINT（クリスタ）
   おそらくデジタル絵描きの中でも最もメジャーなペイントソフトで、他のソフトでできることはほぼ全てクリスタでできると言っても過言ではないほど機能が豊富です。その反面、最初は機能の多さに迷うかもしれませんが色々試していくうちに覚えていくと思うので迷ったらこれを買いましょう。Pro と EX の 2 種類がありますが、漫画を書くつもりなないのであれば Pro の年額プランを買うことをおすすめします。3000 円/年です。

2. Photoshop
   Adobe 製品でのペイントソフトといえば Illustator を思い浮かべる方も多いかもしれませんが、Illustrator はロゴやデザインがメインのソフトとなっており、「イラストを描く」のであれば Photoshop を使用することになります。プロの現場でもよく使

[*16] PR ではない
[*17] 筆者は ASUS ProArt PA279CV-J というモニタを使用しています。
[*18] 筆者は Razer Tartarus v2 を使用しています。

用されており、クリスタに負けず劣らずの優秀さとなっています。Photoshop でイラスト制作→ Illustrator で誌面、デザイン編集という流れもよく見かけますね。デメリットとしては、異常に値段が高い点です。CC を学校から付与されているのならば別ですが、一学生にとってはかなりの負担となります[*19]。

3. SAI
   今でこそあまり見なくなってしまいましたが、少し前はクリスタとタメを張っていたペイントツールになります。動作の軽さが魅力で、ツールに関しても SAI の水彩ペンをクリスタ用に自作する人がいるほどには優秀でした。現在は買い切りとなっており、1 ライセンスあたり 5500 円になります。

4. MediBang Paint
   基本無料で使えるペイントソフトで、基本的に一通りのツール・機能が揃っているため最初から PC で始める方に気軽に試してもらうという観点でおすすめです。

5. ibisPaint
   こちらも基本無料のペイントツールになっており、スマホや iPad で描く方に人気のソフトです。現在もアップデートが頻繁に行われており、クリスタに近づいてきているという噂もあります。広告が邪魔らしいです。（自分は使ったことがないので、よくわかりません…）

やはりおすすめはクリスタです。プロと同じ制作環境が年 3000 円で手に入ることはとても大きく、利用者が多いため、tips や素材の配布が盛んでとても使用感が良いです。

クリスタの詳細な使用方法についても書きたいのですがが、それだけで 1 記事が書けてしまうのでまた今後にしたいと思います。

5　実際に描いてみる

ここまできたらもう後はタブレットにペン先を添えるだけです。

イラスト制作において一番大切なことは観察です。何も参考にせず、数描けば上手くなるだろうというのは単なる幻想に過ぎません。

自分が良いと思ったイラストを常に横に表示しておき、線画のサイズ、強弱の付け方、肌色・影色の使い方、構図、その絵イラスト全てを参考にして描き進めていきましょう。実際の風景、自分自身の写真、3D モデルなどを参考にするのも良いでしょう。

6　終わりに

それではここまで読んでくださり本当にありがとうございます。液タブの選び方について気になった、クリスタの使い方がわからないなどあれば@mikio815(Twitter) に DM でも

[*19]以前は筑波の学生は CC を利用できたらしい。復活してほしいですね。

送ってくだされればなんでも答えます。げんしけん[*20]のイラスト・漫画班の班長もやっているので、ぜひ遊びに来てください。

この記事を読んだ方のうち 1 人でもイラストを描き、その魅力を知って下されば嬉しい限りです。

[*20]現代視覚文化研究会。筑波の古参サークル。

Go でシステムプログラミングするな！

文　編集部　n4mlz

1　はじめに

この記事は非常に強い思想が含まれる可能性があります。注意してご覧ください。

2　Go の悪口

近年、再び Go が注目されている。例えば今年の 3 月 12 日には、Microsoft が TypeScript コンパイラを Go で書き直すという「Project Corsa」[*1]が発表されたことも記憶に新しい。しかしそんな中、私はあえて Go について否定的な意見を述べたい。

2.1　Go に苦労した実体験

私は以前、Go でコンテナランタイムを自作したことがある。コンテナランタイムは、Docker や Podman などのコンテナを実行するためのソフトウェアである。コンテナに関するプロジェクトは大抵 Go で書かれている。例えば Docker や buildx、compose、また低レベルランタイムである containerd、runc なども全て Go で書かれている。私はこれらのプロジェクトを参考にしながら、「tiny-runc」[*2] という名前で低レベルコンテナランタイムを Go で書いていくことにした。tiny-runc は、非特権環境で動作する OCI Runtime Specification [*3] 準拠の低レベルコンテナランタイムを目指して開発している。tiny-runc という名前の通り、「runc」[*4] という OSS を参考に書いている。runc は、Docker や Podman の中でデフォルトで動いている低レベルコンテナランタイムである。

tiny-runc はコードをめちゃくちゃ丁寧に書いて、人々のコンテナランタイム自作の実装のサンプルにして欲しい！ みたいな気持ちがあって書いていたが、だいぶ失敗した。まず、Go は **`fork` と `exec` を分離して実行することが出来ない**。この問題のせいで、例えば `fork` したあとの処理を親と子で 1 つずつ書けばいいところを、わざわざ子プロセス用の処理を自身のサブコマンドとして分けて親から自分自身を呼び出す、みたいな遠回りなことをしないといけない。

例えば、コンテナランタイムの実装では以下のような処理が頻出する。疑似コードなので実際には動かないことに注意。

```
1 struct ContainerBuilder {
2     ...
```

[*1] https://devblogs.microsoft.com/typescript/typescript-native-port/
[*2] https://github.com/n4mlz/tiny-runc
[*3] https://github.com/opencontainers/runtime-spec
[*4] https://github.com/opencontainers/runc

```
}

func (c *ContainerBuilder) mainProcess() {
  pid := syscall.fork()
  if pid == 0 {
    // 子プロセス用の処理
    c.childProcess()
  } else {
    // 親プロセス用の処理
    c.parentProcess()
  }
}

func (c *ContainerBuilder) childProcess() {
  // c のフィールドを使うような処理
}

func (c *ContainerBuilder) parentProcess() {
  // c のフィールドを使うような処理
}
```

しかし実際には fork と exec を分離して実行することが出来ないので、以下のようなコードを書くことになる。

```
struct ContainerBuilder {
    containerID string
    ...
}

func mainProcess() {
  container_builder, err := json.Marshal(&ContainerInfo{
      containerID: "...", ...})

  cmd := exec.Command(os.Args[0], "child", string(
      container_builder))
  cmd.Stdout = os.Stdout
  cmd.Stderr = os.Stderr
  err = cmd.Run()
```

```
13    parentProcess(container_builder)
14  }
15
16  // サブコマンドが child である場合に呼ばれる関数
17  func childProcess() {
18    c := os.Args[1]
19
20    // c のフィールドを使うような処理
21  }
22
23  func parentProcess(container_builder string) {
24    c := json.Unmarshal(container_builder)
25
26    // c のフィールドを使うような処理
27  }
```

このようにすると明確にプロセスが分かれてしまうので、親で利用していた構造体を子で使うみたいなことができない。例えばせっかくオブジェクト指向的抽象化を行っているのに、プロセスを fork するたびに exec を行ってしまうと、もう一度同じ構造体を作らないといけない、みたいなことになる。これはまさしくカスである。

ちなみに runc ではこの問題を、「namespace の分離の部分は C 言語で記述する」という方法を取って解決している [*5] 。ただこの方法は「シンプルで理解しやすいコードで簡潔に記述する」という、tiny-runc で満たしたかった当初の方針と反してしまうので、tiny-runc ではこれを行っていない。結果、fork するためにわざわざ子プロセス用の処理を自身のサブコマンドとして分けて、親から自分自身を呼び出すみたいなカスみたいなコードになってしまったのだ。

またこれはシステムプログラムには全く関係がないが、Go ではパスとパッケージが同じ扱いなので、循環インポートを避けるために同一パッケージにしたい、あるいはドメインモデリング的に同一パッケージの方が自然だが同じパスにファイルが増えすぎてしまい、ある程度のまとまりごとに整理したい、みたいな時に非常に困る。これは Go という言語自身の思想に起因する問題であり、ソフトウェアアーキテクチャを考える上でも Go は非常に不便である。

それまでは「Go って雑に書けていいね」みたいに思っていたのだが、この件のせいで一気に Go が好きではなくなった。まぁ runc でも名前空間を分けたり fork したりする部分だけは C で書かれているくらいなので、「Go でシステムプログラムをするな」(より厳密に言えば「Go でプロセスを扱うようなシステムプログラムをするな」) というのが私個人の

[*5] https://github.com/opencontainers/runc/blob/main/libcontainer/nsenter/nsexec.c

感想である。

何も考えず、脳死で Go を選択したのが間違いだった。Go はシステムプログラムに向いていない。私がこのように主張する理由は、Go が goroutine を実現するために M:N スレッドスケジューリングモデルを採用していることにある。

そもそもスレッドスケジューラというのは、fork(2) と非常に相性が悪い。fork(2) は呼び出し元のスレッドだけをコピーし、それ以外のスレッドはコピーしない。これにより、fork(2) 単体で行うと、子プロセスから見れば Go のランタイムが管理する他の (OS レベルの) スレッドが消失することになる。こうなると、Go のランタイムは子プロセスで正常に動作しなくなり、破滅する。このようなランタイムの破壊を防ぐために、Go は fork した直後に exec を実行するような、syscall.forkExec() という関数を提供している。つまり、Go は fork と exec を分離して実行することが出来ない。ちなみに Go は、この syscall.forkExec() を更に抽象化した、os/exec パッケージを使うことを推奨している。

2.2 余談: それでも無理やり fork(2) を呼ぶ方法

以下の方法は、邪道である。あまりおすすめしない。

syscall.RawSyscall を使う

Go にはシステムコールをラップした syscall パッケージが用意されているが、当然システムコール番号を直接指定して呼び出すこともできる。

```
1  func main() {
2      pid, _, errno := syscall.RawSyscall(syscall.SYS_FORK, 0,
           0, 0)
3      if errno != 0 {
4          fmt.Println("fork␣failed:", errno)
5          return
6      }
7
8      if pid == 0 {
9          // 子プロセス側
10         fmt.Println("Hello␣from␣child␣process")
11     } else {
12         // 親プロセス側
13         fmt.Printf("Forked␣child␣process␣with␣PID␣%d\n", pid)
14     }
15 }
```

cgo を使う

Go は C 言語との相互運用をサポートしている。cgo を使って C 言語の関数を呼び出すことで、fork(2) を呼び出すこともできる。

```
/*
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

void forkProcess() {
    pid_t pid = fork();
    if (pid == 0) {
        printf("Hello from child process\n");
        exit(0);
    } else {
        printf("Forked child process with PID %d\n", pid);
    }
}
*/
import "C"

func main() {
    C.forkProcess()
}
```

2.3 (ちなみに) 自作コンテナランタイムはその後どうなったか

Go で散々な目にあったので、tiny-runc はその後完全に放棄して、新たに Rust で「tiny-youki」[*6] という名前でコンテナランタイムを書き直した。自分でもかなり綺麗に書けていると思っていて、「私が本当に書きたかったコードはこういうものだ！！」と思っている。ノンストレスである。

3 おわりに

Rust 書け。

[*6] https://github.com/n4mlz/tiny-youki

入学した学類が合わないかも？ と思ったら

文　編集部　おかし

1　はじめに

新 1 年生の皆さん、ご入学おめでとうございます。入学を心より歓迎します。知識情報・図書館学類 3 年のおかし (@oka4shi) です。

さて、今後の大学生活の中で、もしかしたら自分の入った学類よりも他の学類の方が自分に合っていると感じたり、興味が出てきたりすることもあるかもしれません。実際、入った学類について後悔した経験のある人は少なくないと言われています。入学して早々でこのような話にはまだ実感が湧かないかもしれませんが、今後そういった機会がもしかしたら来るかもしれません。本記事では、社会工学類の前期入試で入学し、3 年次に知識情報・図書館学類に転学類をした私の経験も交えながら、そんなときに一体どうしたら良いのか記していきたいと思います。

2　年度内にできること

今の学類が自分に一番合ったものではないように感じたときにできることには、大きく分けて年度内で比較的すぐにできることと、中長期的に考えていく必要のあることがあります。例えば、実際に別の学類に移ったりすることはすぐに可能なことではなく、ある程度の期間と準備が必要です。

まずは、前者の比較的すぐできることを紹介していきます。

2.1　他学類の授業を履修する

まず、最も基本的にするべきことは実際に興味のある他学類の授業を履修してみることになります。筑波大学では、他学類の単位も一定の単位数を自学類の卒業要件の単位としてカウントすることができます。この枠を活用し、実際に他学類の授業がどんな様子なのかを履修してみて確かめると良いでしょう。学類によって卒業要件に算入できる単位数は異なりますが、情報科学類の場合は理系科目なら 4 単位まで、文系科目なら 10 単位まで（両方合わせて 10 単位まで）となっています。こちらに関する詳細は履修要覧の履修細則部分で確認してみてください。

履修する他学類の単位はもちろん基礎科目でも問題ありません。ただ、基礎科目と専門科目では雰囲気が結構異なることもあります。そのためシラバスを見て履修要件を満たしているようであれば、あまり標準履修年次を気にしすぎずに取ってしまうのも一つの手です。標準履修年次はあくまでも「標準」なので、必ずその学年で履修しなければいけないというわけではありません。たまに履修が認められないこともありますが、基本的には問題ない場合がほとんどです。

後述する転学類の成功率を上げるためにもこれは必要だと考えられるので、積極的に行うと良いと思います。

2.2 学類の中で自分の興味のある分野が本当にないか調べる

次の選択肢として、自分の学類の中で自分の興味がある分野が本当にないかどうか調べてみるというのも有効です。特に 1 年生の間は基礎的な科目が多く、そこまで興味がないような分野を多く学ぶことになりがちです。しかし 2 年次以降では段々と専門科目が増え、中にはかなり興味関心に近いことを扱っているものも少なくないかもしれません。そのような授業を履修してみたり、同じ学類の先輩がいれば聞いてみたりすると良いでしょう。また、学類公式のウェブページをぜひ隅々まで読んでみることをおすすめします。参考になる情報があるケースも結構あります。

3 長期的に考えること

これまでに紹介したような方法を試してみて、やはり他の学類等に移りたいと決心した場合には主に転学類、再受験、編入学、他分野の大学院を選択といった方法が考えられます。

3.1 再受験

他大学や筑波大学を再受験するということが考えつくかと思います。しかし、取得した単位を持ち越すのが難しいこと、1 年からやり直しになること、受験勉強と不合格だった際のための単位修得を両立するのが用意ではないことを始め、様々な障害があります。そのためリスクが高く、本当に最終手段にするべきです。

3.2 編入学

主に他大学に移る手段として、編入学という方法があります。各大学が編入学試験を実施しており、合格すれば 2 年次または 3 年次から入学することができます。また、ある程度の筑波大学で履修した単位を単位変換によって他大学で卒業要件を満たすための単位として使えることもあります。実際に 4 年制の大学から他の大学に編入学したという事例も耳にします。他大学に移籍したい際には有力な選択肢となるでしょう。

3.3 転学類

筑波大学には転学類という制度が存在します。これは、筑波大学学群学則第 20 条（学群又は学類間の移籍）で定められている、筑波大学内の他の学類に移籍することのできる制度です。筑波大学に興味がある分野の学類が存在する場合には非常に有力な選択肢となります。メリットとして、対象の学類の進級条件を満たしていれば留年の必要なく他分野に移ることができるということがあります。それ以外にも同大学であることによって勝手がある程度分かることなども含め、再受験や編入学よりも比較的大掛かりにならず負担が小さいかと思います。入学金や授業料などの金銭的な負担を比較的小さくするできることもメリットの一つとなるかもしれません。デメリットとしては、選考が存在しその選考に通るか分からないということがあります。しかし、通らなくても自分の学類に所属し続ける

ことは当然できるので、狭き門ではありますが筑波大学で移籍したい場合はチャレンジしてみる価値があるでしょう。

転学類の流れ

筑波大学には転学類という制度が存在します。これは、筑波大学学群学則第 20 条（学群又は学類間の移籍）で定められている、筑波大学内の他の学類に移籍することのできる制度です。筑波大学に興味がある分野の学類が存在する場合には非常に有力な選択肢となります。メリットとして、対象の学類の進級条件を満たしていれば留年の必要なく他分野に移ることができるということがあります。それ以外にも同大学であることによって勝手がある程度分かることなども含め、再受験や編入学よりも比較的大掛かりにならず負担が小さいかと思います。入学金や授業料などの金銭的な負担を比較的小さくするできることもメリットの一つとなるかもしれません。デメリットとしては、選考が存在しその選考に通るか分からないということがあります。しかし、通らなくても自分の学類に所属し続けることは当然できるので、狭き門ではありますが筑波大学で移籍したい場合はチャレンジしてみる価値があるでしょう。

転学類選考について

転学類が可能なのは、1 年から 2 年に上がる段階と 2 年から 3 年に上がる段階です。一度転学類選考を受けた学類に対して、もう一度転学類を志望することはできないため、よく考える必要があります。

選考の基準は正直よく分かりません。様々な噂はありますが、そもそも転学類を志望する人数があまり多くないため十分なデータがありません。学類によっても細かい基準は異なると予想されます。

願書において「私は、転学群・転学類にかかる選考において、入学出願時に提出した調査書および入学試験成績を使用することを、承諾します」という欄に署名を求められることから、入学試験の成績が参照されるという噂があります。また、入試難易度の高い学類から低い学類に移るのは比較的容易だが逆は難しい、といったような噂もありますが真偽は不明です。個人的にはこれらの噂に関してあまり気にする必要はないと思います。

それよりも入学後にどんな単位を履修しているかという点が重要になってきます。履修している科目・成績は選考過程で参照されます。これは選考にかなり影響していると考えられます。ここからは私の見解も混じりますが、恐らく学類は転学類後に卒業がどれくらい現実的かという点をかなり気にしているはずです。面接の内容からもそれが示唆されていたように思います。

そのために、まずは転学類先の履修細則で進級要件及び卒業要件を確認しておきましょう。そして、それをもとにあらかじめ転学類先の単位を一定程度履修しておくということが必要になります。特に進級要件は満たしていないと落とされる可能性が高いと予想できます。また、転学類後の履修を仮組みしておくのが望ましいです。これによって、標準修業年限で十分卒業ができると示す事ができます。それが厳しそうなら転学類の望みは薄いの

で、そうならないよう単位を修得しておく必要があります。

他にも「その学類である必然性」は重要であると思います。対象の学類の授業を複数履修して興味のある分野を見つけ、具体的に何をしたいのかを志望理由書や面接で伝えられるようにしておく必要がありそうです。このためにも、とにかくまずは興味のある学類の授業を履修してみることを強くおすすめします。

ちなみに、学類にもよるとは思いますが成績（GPA）はあまり考慮されないようです。私の GPA はそれほど高くなく、むしろ低い方でしたが選考を通過することができました。成績よりも志望の理由やきちんと卒業ができそうかという点が重視されるようです。

3.4 他分野の大学院を選択

長期的なことにはなりますが、大学院に進学しようと思っている場合には大学院の選択を別分野にするという手もあります。特に大学での移籍などが上手くいかなかったとしても、大学院の入学試験を受験しそれに合格すれば他分野に移ることができます。

4 おわりに

仮に自分の学類の扱う内容に興味があまり持てなくても心配する必要はありません。筑波大学には幅広い分野が存在し、他の分野の授業を履修することも容易にできます。それに、自分の学類で満足できない場合にもいわゆる仮面浪人や再受験をしなくとも十分に他分野に移れる余地があります。

また、自分の学類にも案外面白い分野や側面があるかもしれません。もし移籍を志望してそれに失敗してしまったときに絶望しないためにも、それを探してみるのも良いでしょう。特に分野が近い場合は、移籍志望先と元の学類にはかなり似ている側面や近い部分があることも多いです。

ここまで読んでいただきありがとうございます。最後にはなりましたが、皆さんの充実した大学生活をお祈りしつつこの記事を締めたいと思います。

函館の五稜郭はなぜ五角形なのか

文　編集部　appare45

WORD 編集部の appare45 です。新入生・編入生の皆様、ご入学おめでとうございます。

ところで、私は先日 jsys24 [*1] で北海道に旅行に行きました。そこで函館にある五稜郭がなぜ五角形なのかを調べたのでご紹介します。

なお、私は特に歴史や北海道に詳しくないので、誤りなどを見つけましたら教えていただけると幸いです。

1　五稜郭とはなにか

「五稜郭（ごりょうかく）」とは北海道函館市にある星型の城です。国内の城では割と新しく、江戸時代末期の 1864 年築城となっています。

図 1　五稜郭の航空写真

2　函館に行く

私達は、もともと札幌に旅行に行く予定でした。そのためホテルなども札幌にとってありました。しかし、私と arata_nvm は過去に札幌を訪れたことがあったので、二人で札幌

[*1] 2024 年度筑波大学学園祭実行委員会情報メディアシステム局のこと。詳しくはこちらの記事をご覧ください https://zenn.dev/sohosai/articles/fab04eefa12115

以外の場所に行こうという話になりました。

ところで、札幌から函館までは鉄道でおよそ 300 キロ（直線距離で東京から名古屋までくらい)、特急 [*2] で片道 4 時間弱 [*3] かかります。

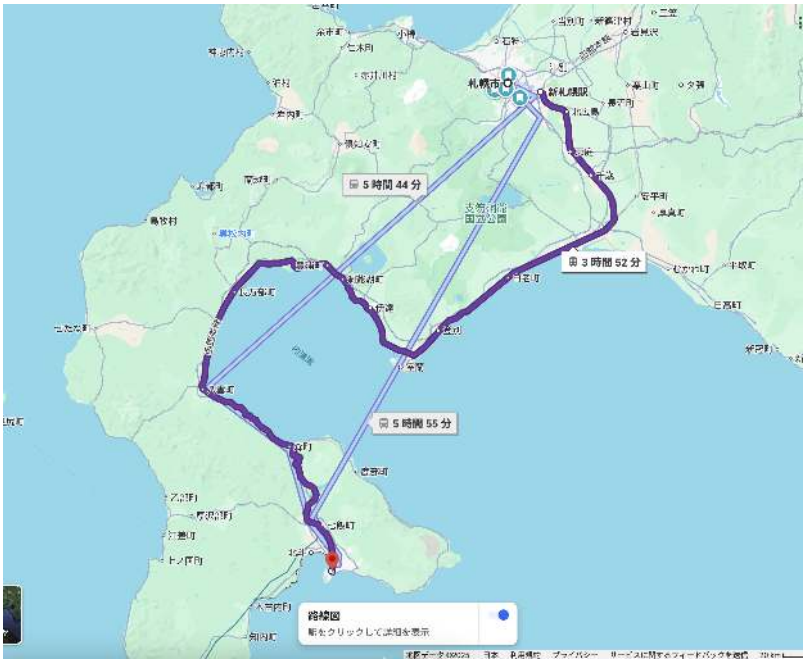


図 2　Google マップでの札幌から函館までの経路検索結果

もし、五稜郭を訪れてみたいという方は、函館にホテルなどを確保することをおすすめします。

3　いざ五稜郭へ

五稜郭には五稜郭タワーというタワーがあります。ここに登ると五稜郭全体を見渡せるほか、その歴史についても学ぶことができます。

[*2]特急北斗に乗りました。片道約 1 万円！

[*3]暇すぎてつくばエクスプレスの駅名を列挙する遊びをしていたのですが私はみどりのと三郷中央を忘れていました

図 3　五稜郭タワーから見た五稜郭の様子

図 4　コンピュータ手相占い

館内にはなぜか「コンピュータ手相占い」なるものがありました。

4　五稜郭誕生の経緯

五稜郭がなぜ五角形なのかを知るためにまず次の 2 つの書籍を読みました。どちらも筑波大学図書館に所蔵されており、大学の図書館ってすごいのだなぁと思いました。[*4]

函館市史 通説編 第 1 巻

[*4]私は昔の文章を読むのが得意ではないのですが ChatGPT に頼むと読みやすくて良かったです

函館市の歴史についてまとめられている資料です。2007 年出版ということで日本語がかなり読みやすい印象を受けました。オンラインでも読めます。https://adeac.jp/hakodate-city/catalog/mp000010-100020

新撰北海道史 (復刻版) 第 2 巻

北海道全体の歴史がまとめられている資料です。原著が 1936 年出版ということで日本語が読みづらい一方で、図などが豊富でした。

これらの資料からわかったのは次のことです。

まず、函館はもともと松前藩の管轄でしたが、江戸時代末期が近づくにつれて外国船の接近が見られるようになり 1802 年に蝦夷奉行が設置され江戸幕府の直轄地となりました。その後色々あり、松前藩による統括となったりもしますが、1854 年に箱館奉行が設置され、函館は再び幕府の直轄地となります。

この背景には幕府が 1854 年 3 月 3 日に米国と結んだ日米和親条約の結果函館を開港することになったことがあります。

当時の箱館奉行所は旧松前藩の役所を使っていましたが、立地的に警備が難しく防衛などの観点からも移動が必要だとされました。その結果奉行所は内陸の亀田（現在の五稜郭の位置）という場所に移されることが決められました。

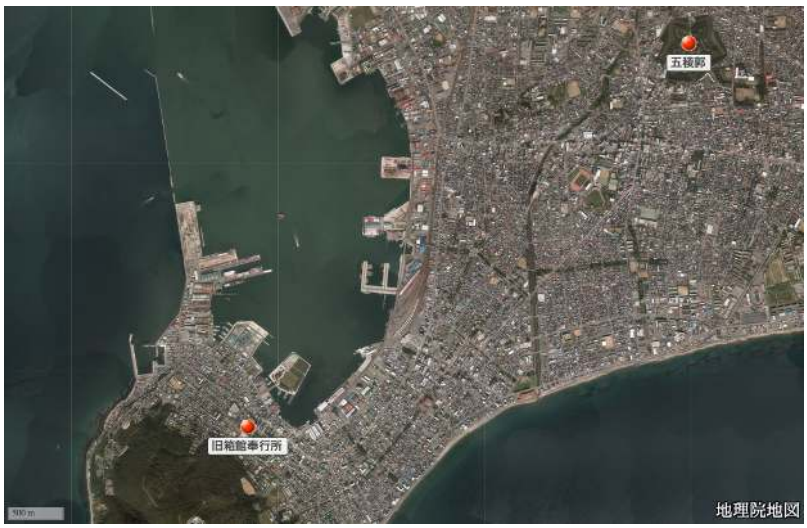


図 5　旧箱館奉行所と五稜郭の位置関係

その後具体的な予算を決めるに当たり、亀田役所の具体的な計画を進める事になり箱館奉行は西洋の陣法術式を採用することを 1855 年 12 月、幕府に対して上申しました。この際に予算書にはじめて五稜郭という記述が登場しました。この計画はその後承認され、工事が始まることになります。

このときの計画では五稜郭は軍事的機能を持ち外国からの襲来に備えられるよう設計されていたようでした。

工事にあたっては役所機能の完成が最優先事項となっていました。そのため、堀[*5]や石垣をすべて完成する前に役所を移転しその後石垣を工事する計画となっていました。しかし、地盤の問題からこの工法は難しいことが途中でわかり、役所の完成よりも前に石垣を完成させることになりました。

また、当初の西洋式石垣を実現しようとすると予算不足となることは明白だったため石垣についても見た目を重視するのではなく安価で頑丈な石垣で作るよう設計の変更が図られました。

このような変更が行われた背景には、箱館港の開港後軍事的攻撃を受ける可能性が低いことを認識していたからだと考えられます。一方で対外的に国威を保つため工事は続行していたものと思われます。

結果的に当初の計画[*6]と大幅にグレードダウンした城として五稜郭が誕生したのでした。[*7]

計画段階の五稜郭: `https://hakohaku-archives.c.fun.ac.jp/records/500313`
完成した五稜郭: `https://hakohaku-archives.c.fun.ac.jp/records/S59-0015`

ここまでで五稜郭がなぜ生まれたのかはなんとなく理解することができました。なお、結果的に誕生した五稜郭は旧幕府軍と新政府軍の戦いの間で何度も征服されておりあまり強い城ではなさそうだという印象を受けました。

しかし、なぜ五稜郭が五角形になったのか、なぜ突然西洋式の城を突然作ることになったかについては良くわかりませんでした。この背景について考えることにしました。

5　五稜郭を設計した武田斐三郎

五稜郭を設計したのは江戸時代の蘭学者「武田斐三郎（たけだあやさぶろう）」とされています。武田斐三郎はオランダを通じて西洋式の築城手法に関する知見があったと考えられます。1857 年に武田斐三郎は前年に函館へ来航したフランス軍艦の船員の助言を得て「ヴォーバン式」を採用したと語っています[*8]。

たしかに 図 6 を見るとその形は五稜郭の当初の設計図と酷似しているように思います。

ではなぜヴォーバン式の城は星型なのでしょうか？ これは、古くから欧州の塔の形状として使われていた円柱形には死角が生まれてしまうという課題がありました。そこでこの円形を尖らせた結果星型の角の部分が誕生したのでした。

しかし、ヴォーバン式の城がすべて五角形なわけではありません。実際国内では函館市内にはなんとびっくり四稜郭が存在しています。また、海外では六角形のもの[*9]や三角形[*10]のものまで存在しています。

[*5] 堀の水は冬になると凍ったらしく、これを氷として販売していた時期もあったそうです
[*6] 五稜郭の設計図は複数あるがどれも同じような見た目をしている
[*7] 本当は図を貼りたかったのですが許諾を取るのが大変そうだったのでリンクを貼っておきます
[*8] この原本を読みたかったのですが、見つけることができませんでした
[*9] ノヴェー・ザームキ（スロバキア）の計画
[*10] チェコ オロモウツの稜堡と要塞計画

図 6　フランスリールの航空写真（Google Earth より）

図 7　四稜郭の航空写真

しかし、どういう経緯で武田斐三郎がヴォーバン式を採用し五角形が選択されたのかについて明らかにすることはできませんでした。[*11]

もしあなたが五稜郭を訪れる機会があれば、ぜひ五角形の真実を探ってほしいと思います。

[*11]「箱館五稜郭築城史　五稜郭フランス式築城論並に築城考証年表」という本に書かれているようなのですが北海道立図書館にしか配架がなさそうでした

続・グルノーブルだより

文　編集部　Azumabashi

1　あらまし

大学院情報理工学位プログラム[*1]と，フランスのグルノーブル・アルプ大学 (Université Grenoble Alpes; UGA) は，2024 年度より修士ダブルディグリープログラム (double degree program; DDP) を開始しました．WORD は本来情報科学類誌ではありますが，WORD 56 号[*2]に引き続き，少しだけ現地の様子をお伝えしようと思います．

入学した途端に学部卒業後なんて遠い未来に起こり得る（しかも「必ず起こる」ではなく「起こり得る」でしかない）ことを話さないでほしいなぁ，と思ってしまったそこのあなた！ 実はそうとも限らんぞ，ということがわかるかもしれません．

2　UGA での授業

2.1　授業があるのは半年だけ

ダブルディグリープログラムは，修士の学位を筑波大と UGA の双方から取得することを目指すプログラムです．したがって，UGA でも「それ相応」のことに取り組む必要があります．ここでの「それ相応」のこととは，授業の受講と研究です．UGA では，秋から冬（9 月から 1 月末まで）にかけての第 9 セメスター (S9) は授業だけを行い，その後の第 10 セメスター (S10)[*3]は研究だけを行う，という方式でプログラムが構成されています．筑波大をはじめとする日本の大学院のように，講義の空き時間に研究をする，というようなシステムにはなっていません[*4]．

なお，セメスター番号が 9 と 10 になっていますが，これは学部 1 年生から番号が通算されていること，フランスの学士課程は 3 年間で構成されていることによります．以上のような事情で，本稿執筆段階（3 月末）では S9 は既に終了し（つまり UGA での授業はもう受けなくてよい），S10 に入っている，ということになります．

2.2　よくある授業の構成

さて，当然「UGA での授業ってどんな感じ？」というのが気になる方もいらっしゃるでしょう．授業は授業なので，基本的にはおおよそ想像通りの授業です．ただし，授業時間は筑波大のもの（75 分）よりも長いです．具体的には，1 コマ 90 分が基本で，ほとんどの場

[*1] 情報科学類生が進学する大学院のプログラムのうち最も有力なもののひとつ．

[*2] https://www.word-ac.net/post/2024/1123-word56/

[*3] 2 月から 8 月と思われますが，修論審査が 6 月と 9 月の 2 回あるため，正式な終わりがいつなのかは微妙なところです．

[*4] 色々な事情で S9 のうちから研究を始めておくことをおすすめしますが，そういう細かいことはおいおい…….

合は（5 分程度の休憩時間を含んで）180 分連続で授業が進みます．もっとも，これに関しては筑波大の授業時間が日本の大学における標準的な授業時間（90 分）よりも短いせいなのですが……．

授業の進め方も，ほとんど変わりません．ほとんどの場合はスライドを使って説明が進んでいきます．ただし，1 つの講義で最初から最後まで同じことをやる，ということはあまりありません．多くの場合では，1 つの講義は 2〜3 個のパートに分けられており，しばしばそれぞれのパートごとに課題や試験が課されます（課題はない場合もあります）．1 つのパートは 1 人（たまにオムニバス形式的に複数人で担当することもある）の先生が担当し，別々のパートは別々の先生が担当するので，UGA での 1 つの講義は（筑波大の感覚でいうところの）複数の講義を連結したもの，とも言えるでしょう．

授業で出される課題は，主にプログラミングの課題ですが，数学的な要素が絡んだり，プログラムの振舞いに対する考察を書いたりすることもあります．どのようなフォーマットで出題されても，**情報科学類の 4 年間で習得するべきことがきちんと身についていれば，あまり心配することはない**でしょう．言い換えれば，それだけこれからの 4 年間で学習することは，大学院以降の基盤となる内容であり，従ってそれだけ重要であるということです．課題は，完全に宿題として出されたり，授業の時間の一部 (lab session) を使って行ったりします．後者の場合は，学内のコンピュータ室で授業が行われます．ただし，コンピュータ室のコンピュータを使うことは（コンピュータはきちんと動くのですが）DDP を通して留学する人にはあまり推奨しません．手持ちのコンピュータでやったほうが便利，というのもありますが，それ以前の理由があります．なぜかは行けばわかります……．

2.3 期末試験

先程「試験が課される」と述べましたが，UGA では（少なくとも情報理工学位プログラムとの DDP を通して入学するプログラムでは）よほどのことがない限りレポート 100% で成績がつくということはありません．最低でも成績の 50%（しばしば 70% 程度）は，期末試験の点数が反映されます．期末試験は，多くの場合 2 時間から 3 時間ほどの時間で，与えられた問題を解いていくという形式です．多肢選択式ではなく，すべて記述問題です（「〜を証明せよ」「〜を求めよ」「〜を説明せよ」という形式の問題が並びます）．記述問題ですから，当然答えだけでなく，その答えを導くための導出も書く必要があります．当然期末試験ですから，「ちゃんと授業内容をわかっていれば点が取れるが，わかっていないと取れない」というような問題が出されます[*5]．このボリュームの試験が一週間に一気に来るので，試験直前は試験勉強にかなりの時間を割くことになります．その分，試験が終わった後の開放感は際立ちます．私も，試験が終わったその流れで，学内の student bar[*6]に立ち寄って友達とビールを飲みつつ，試験の終了を祝った記憶があります．

ちなみに，試験の過去問は担当の先生のページで公開されていることがあります．そうでなくても，LMS（講義資料の配布などに使うシステムで，筑波大での manaba に相当）や

[*5] この点は情報科学類の期末試験でも同じですが．

[*6] `https://www.eve-grenoble.fr/`

メールなどで配布されることが多いです．過去問を見れば，だいたいの雰囲気がわかります（もっとも「だいたいの雰囲気」でしかないのですが）．

2.4 英語力を鍛えよう

授業に関して，ひとつ重要な点に意図的に言及してきませんでした．授業はすべて英語で行われます．当然，課題や期末試験の答案も英語で書くことになります．そのためには英語力のいわゆる「4 技能」のうち，主に reading, listening と writing が必要になります．

reading はどういう用途で使うかわかりやすいでしょう．講義資料や課題の説明，試験の問題は英語で書かれているので，英語が（少なくとも）そこそこ読めないとお話になりません．試験によっては，英語で書かれた論文に関する設問があることがあるので，英語論文を英語のまま読める必要がある，と思ってください．

また，「writing」とはいっても，大学入試で取り組んだかもしれない「日常的な文章[*7]」や「ちょっとしたエッセイ[*8]」に対応するための作文力ではなくて，数学的な文章を書くための，アカデミック寄りの writing 力が主に要求されます．例えば，すべての正の整数 n に対して，

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

が成り立つことはよく知られていますが，この証明が英語で書けるでしょうか？ というお話です．

特に listening は，説明されたことを聞き取れないとお話にならないので，苦手な方も多いかもしれませんが，強化しておくことをおすすめします．スライドがベースで授業が進むことがほとんどとはいえ，定理の証明などは直接黒板に書かれることも多いです．

残りの speaking は，グループ作業があるならまだしも，授業を受けるだけならそこまで必要にはなりません．ただし，グルノーブルで滞在する期間中に研究活動をすることになるので，いずれにせよ必要になることには変わりありません．

悲しいことに，これらの技能は一朝一夕には伸びません．**将来的に留学などを考えているのであれば，今のうちからコツコツと英語力を伸ばすとよいでしょう**．授業だけに限らず，（日本でのものを含めた）研究活動や日常生活（特に同じく UGA に滞在している学生とのやり取りなど）にもきっと役立つでしょう．ちなみに，要求されている英語力の最低ラインは，IELTS Academic 6.0，TOEIC 4-Skills で 700 などです[*9]が，これ以上を目指しておくことをおすすめします．

2.5 最終的な成績

試験を受けると，当然その答案は採点され，予め決められた方法により成績が算出されます．UGA では[*10]，成績は 20 点満点で評価され，20 点中 10 点が取れれば単位取得とな

[*7]「この道をまっすぐ行けばバス停に出られます」のような文章．

[*8]与えられたテーマへの意見を 100 words で書くような設問．

[*9]`https://relint.imag.fr/MainEn/Admission`，2025 年 3 月 28 日閲覧

[*10]実のところフランスの大学はみんなそうらしいのですが，「らしい」程度の情報しか持ち合わせていません．

図 1　雪化粧を纏う山々．この写真は UGA のキャンパス内で撮影したものです．

ります．点数を細かく決める科目では小数第 2 位まで点数がつけられますが，ざっくりな科目では 1 点単位で点数がつきます．筑波大の単位取得基準[*11]と比べると緩いように思えますが，実際のところは，単位取得の難易度（というのはヘンな言い方ですが）そのものはほぼ同等か，UGA のほうがやや厳しいと思われます．1 回の試験だけでは単位取得に問題のあるケースも多いのか，通常の期末試験（以下，「本試験」という）に加えて追試験 (resit exam) が用意されています．本試験で成績が基準に届かなかった場合には，追試験を受けることで成績を挽回できる，というわけです．筑波大ではこういうシステムはないのですが，ではなぜ UGA ではわざわざ追試験が別途用意されているのかを考えると…….

3　ギャラリー

授業ばかりに缶詰になっていても仕方ありません．月に 1 回ぐらいはどこかにお出かけしましょう！ 特にフランスでは，高速バスも高速鉄道もチケット代はダイナミックプライシングに従うので，早め早めから計画を立てておかないと値段が上がってしまいます…….

3.1　グルノーブルの冬

グルノーブルの冬は，クリスマスの前後が一番寒いです．年が明けると徐々に暖かくなってきます．グルノーブルは山に囲まれた街ということもあって，どこにいても山を見ることができますが，この時期になると山も雪化粧を纏います（図 1）．キャンパス内にも雪が降ることもありましたが，積もりはしませんでした…….

[*11] 100 点満点中 60 点取れれば単位取得と言われている．

図 2　クリスマスマーケットの入口に掲げられていた看板.

山があり，雪があり，となれば，スキーに行く人もかなり多いようです．グルノーブルのスポーツ用品店でも，この時期はスキー用品をよく見かけます．ところによっては，グルノーブル市街地から直通のバスでスキー場まで行けるようです．とはいえ，わたしは行きませんでしたが…….

ちょうどそのクリスマスの時期は，グルノーブルでもクリスマスマーケット (Marché de Noël) が開催されています（図 2）．街の中心部の広場全体が 1 か月ほどマーケットとして使われ，食べ物・飲み物・物販など，様々なお店が出ています（図 3）．特にここでいただいたホットワインは今までで一番おいしいものでした．もしワインが飲めればおすすめです．

グルノーブルのクリスマスマーケットは，そこまで大規模なものではありませんが，それでも楽しむには十分な規模があります．比較的混雑はしていますが，混みすぎというほどでもなく，ちょうど良い程度です．中には薪ストーブが設置された小屋もありました（仮設なので，そんなに暖かいものではないでしたが……）．

ちなみに，クリスマスの時期にスーパーを訪れると，「なんでこれが！？」という食べ物が山積みになっています．おそらく多くの人が知っていて，どうやって食べるのかも知っている食品ですが，日本の感覚だとクリスマスとは全く結びつかない食べ物です．それは何かは，行ってみてのお楽しみ…….

3.2　80 ans de la Libération de Grenoble

グルノーブルの市役所 (hôtel de ville) 前のトラムの停留所に，ある旗（？）が掛かっているのを見かけました（図 4）．下部には，‘‘80 ans de la Libération de Grenoble’’ と書かれています．ご存知のように，2025 年は第 2 次世界大戦から 80 年を迎える年であり，またフランスがナチスドイツの支配から解放されてから 80 年を迎える年でもあります．関連して，色々と記念行事が計画されているようです[*12]．

[*12] https://www.grenoble.fr/628-programme-des-80-ans.htm

図 3　クリスマスマーケットの地図.

図 4　‘‘80 ans de la Libération de Grenoble’’

図 5　欧州議会.

3.3　ストラスブール旅行

フランス北東部の都市ストラスブール (Strasbourg) およびこの近郊の都市にも旅行に行きました．ストラスブールは，今でこそフランスの都市ですが，歴史的経緯からドイツの影響を強く受けている街です．現在では欧州議会（図 5）などが存在し，EU 圏内でも重要な都市のひとつになっています.

ストラスブールからライン川（仏 Rhin，独 Rhein）を挟んで反対側は，もうドイツのケール (Kehl) という小さな町です．ストラスブールからケールまでは，直通のトラムで移動できます．国境を跨ぎますが，シェンゲン協定内ということもあり，パスポートチェックなどはありませんでした．ただ，国境近くの橋で高速バスがドイツ警察の検問を受けていたのを目撃したので，パスポートは念の為持ち歩いていたほうが良さそうです．一応ライン川上にフランスとドイツの国境があるはずなのですが，見事に何もありません（図 6）．橋もただの橋で，国境を跨ぐからといって特別な設備は一切ありません．普通に徒歩で国境を跨ぐことができます．本当にただの隣町という感じになっています．ちなみに，フランスでは歩行者用信号は守られないことがデフォルトで，ストラスブールでもそうなのですが，ケールではきちんと守られているので要注意.

ストラスブールの近郊に，コルマール (Colmar) という街があります（図 7）．ここの旧市街は，某アニメの舞台のモデルになった街として有名らしいです．ストラスブールからコルマールまでは，TER（transport express régional，日本で言うところの普通電車）で 30 分ほどです．乗車した TER の行き先はバーゼル（独 Basel）でしたが，バーゼルはスイスの都市です．しれっと国際列車に乗ることになります．この列車は地味に客車列車で，200km/h でぐいぐいと進んでいきます.

4　おわりに

今号では，UGA での授業の話と，旅行に出た話を少しばかりしました．次号では，残った研究の話題などをしようかと思います．発行時期によっては，次回が最終回になるかも

図 6　ドイツ側から見るライン川上のフランス・ドイツ間の国境．川の反対側はフランスです．

図 7　コルマールの旧市街の典型的な風景．プライバシー保護のため，モザイク処理を施しています．

しれません．

おそらく，**大学院生に対して**[*13]，2025 年度出発分の DDP 参加者の募集が近々かかるのではないかと思います．もし興味のある方がいれば，積極的に情報収集するとよいでしょう．運よく筆者を探し出してもらえれば，質問にお答えすることもできます．

本号は一応「入学祝い号」ですので，新入生の皆さんに記憶に留めておいてもらいたいメッセージを最後に載せておきます：

1. **学類の授業は，将来の基礎になるので大事．**
2. **英語力はあるに越したことはない．維持・向上しておこう．**

それでは，よいつくばライフを！

[*13]学類生は 4 年後を待ちましょう！

Coins 新歓パンフを支える仕組み

文　編集部　間瀬 BB, (謎の労働担当：Ryoga.exe)

こんにちは、WORD 編集部の間瀬 BB (@bb_mase) です。

早速ですが、新入生の方は「新歓パンフ」、読みましたか？ ゑ！？ 読んでない？ 今すぐこんな本閉じて新歓パンフを見ましょう。[*1][*2]

図 1　Coins 新歓パンフの表紙

この記事では、新歓パンフがいかようにして作られたのかというのを自慢していきたいと思います。テキトーに Microsoft Word (マイクロソフト ワード) [*3]で書いただけなんじゃないかって？ ここは Coins ですよ？

[*1] ~~この WORD 記事を書いている間も新歓パンフには様々な編集作業が走っているらしい←妙だな……（完成しておらず、手元になかったら本当にすみません。おそらく入学式周辺に渡されます。）~~ ← 引っ越し準備号落としました。なのでここに居ます。泣き

[*2] この本を先に読もうだなんて、救いようがない AC ですね……

[*3] *Microsoft Word* と書くことが非常に大事です。WORD 編集部は Microsoft Word よりも前に WORD を名乗っていたのですが（これは事実）、ビルゲイツに目を付けられて WORD の名を勝手に使われたあげく、そちらの方が有名になってしまったという悲しい歴史的経緯があります。[要出典]

1 技術選定

とにかく、Microsoft Word 以外のソフトウェアを選ぶ必要があります。ここで出てくるのが組版ソフトウェアです。オープンなソフトウェアで言うと、LaTeX（ラテフ）やその拡張である LuaLaTeX（ルアラテフ）などがやはり有名ではないでしょうか。近頃の WORD の冊子も LuaLaTeX をベースにして作られています。[*4]

また、日本発ソフトウェアである SATySFi（サティスファイ）や、Web 技術で組版をする Vivliostyle（ビブリオスタイル）など、様々あると思います。商用ソフトウェアでは Adobe InDesign（アドビ インデザイン）などもあると思いますが、こいつも例のごとく、なんなら Microsoft Word よりたちが悪く[*5]、[redacted][*6]ので今回は除外されました。

先に答えを書いてしまうと、最近ナウでヤングな組版ソフトウェアおよびフォーマットの「Typst（タイプスト）」を使うことにしました。選定というか、周りの先輩方がレポートの提出等でいっぱい使ってて、おもしろそうと思ったからですね。

1.1 Typst についてざっくり

※注：筆者（間瀬 bb）は、LaTeX を情報リテラシー（演習）の授業でテキトーにやったぐらいしかテキストベース組版の経験がありません。そのため、LaTeX と比較して語れる部分があまりありませんのでご了承ください。

さて、ここで初めて本格的に Typst を書いてみるんですが、組版ソフトウェアにしては直感的に書けます。例えば大見出しを書く時には以下のように記述します。

```
= これは大見出しです！
```

結構 Markdown っぽい。ということは、中見出し (H2) は？ 太字は？ とかやっていくとこうなります。

```
== これは中見出し（H2）です
ふつうのテキスト

*強調します*

_Italic_
```

直感的！[*7]

このような感じなので、チュートリアルをしたり、ドキュメントを端から端まで読む必要はなさそうですね。どうやら Typst は

[*4] 昔はジャストシステムズの一太郎を使っていたらしいですね。
[*5] Microsoft Word については、一応規格はオープン（らしい）
[*6] ぼくってもうすぐ「だいがくにねんせい」ってやつになるらしいので、いえないこともあります。
[*7] Markdown を常日頃から書いているので

> Typst is a new markup-based typesetting system that is designed to be as powerful as LaTeX while being much easier to learn and use.

と言っているようです。[*8]

では、組版要素どこやねんとなりますが、それはマークアップに含まれた、# から始まる関数呼び出しによって書いていきます。例えば、下記のコードは先ほど示したコードと同じことを表しています。

```
#heading(level: 2)[これは中見出し（H2）です]
#text[ふつうのテキスト]

#strong[強調します]

#emph[Italic]
```

もちろん、これは上のコードで出力される結果と同様にするために、何もオプション等を付与していません。例えば、`#text[ふつうのテキスト]` の部分では `#text(coler: red)[ふつうのテキスト]` みたいにオプションを付与することも出来ます。Markup では厳しい細かい組版をしたくなったらおもむろにコマンドをゴニョゴニョ書き始める感じですね。

2 RE: 新歓パンフを支える仕組み

では、本題に行きましょう。新歓パンフを支える仕組みは主に 3 つあります。

- GitHub 上でのプロジェクトマネジメント
- GitHub Actions 上での CI/CD
- 組版（見た目の制作）

上から順番に解説していきます。

2.1 GitHub 上でのプロジェクトマネジメント

新歓パンフ 2025 では新歓パンフ 2023 あたりでも行われていたフロー、または WORD のフローを参考にしました。赤入れ等のフローが GitHub の PR 機能を使うとうまくいき、Typst で作るということで GitHub Actions を用いた CI/CD もブンブン回していきたいと考え、基本的に GitHub 上で全てを出来るようにするということになりました。

ただ、新歓パンフを作る人は必ずしも「ローカルで Typst を入れてブランチ切って Makefile を回してね〜」が出来る人とは限りません。そのため、そこについてはかなり意識してやって行きます。

[*8] GitHub の typst/typst レポジトリの README.md (https://github.com/typst/typst/blob/main/README.md) より引用

2.2 GitHub Actions 上での CI/CD

とりあえず、CI/CD をやっていこうとなりました。新歓パンフは先述の通り、README に書いたら環境を用意してくれるような詳しい人がやるとは限りません。

はじめは、「Docker さえ入れれば、あとは `compose.yml` があるので、それだけ実行してもらう」というのも考えましたが、大変不親切です。また、各々の PC に入っているフォントが異なるという問題も面倒です。そのため、WORD 等と同様に、記事を書いた PR を出したら GitHub Actions でビルドし、その結果となる PDF をどこかで見れるようにすると良さそうです。ここではその CI(ビルド)/CD(アップロード) を構築します。

なお、レポジトリはえらいことになっており、とても公開出来ないため、ここではコードの一部分についてちまちま紹介をしていく形を取ります。

基本的には GitHub 上にある、WORD-COINS/article-template [*9] を参考に、CI/CD を組んでいます。WORD の冊子はこれをフォークすることによって出来ています。[*10]

以下の画像は完成したもののディレクトリ構造です。各記事のタイトルや位置は基本的に前年度のものを踏襲するため、確定しています。そのため、ディレクトリ構成は以下のように執筆者に template からコピーしてもらった上で、ルートにドカドカ入れる PR を作成してもらいます。Prefix に番号をつけて、ゴチャゴチャにならないようにもします。

そして、各 PR で CI が回るときに問題になるのが、ここでは全体のプレビューではなく執筆者が書いた記事のプレビューが必要になるので、どこのディレクトリに入ってビルドすれば良いのかが CI にはわからないというものです。

そのため、執筆者には main ブランチからフォークした `articles/${ディレクトリ名}`のブランチで出来た PR を出してもらいます。それを CI で切り出して (`# [1]`)、そのディレクトリに cd して (`# [2]`) ビルドをします。

ちなみに PAT を作って checkout に食わせている (`# [3]`) のは、新歓パンフの中に「筑波大学辞典」が含まれている関係です。それは外にあるプライベートレポジトリに CSV が切り出されており、それを submodule として突っ込んでいる関係でコケてしまうためですね。

その後ファイル名を適当な時刻にして、R2 に突っ込みます。そしたらリンクが生えるので、PR にコメントします。(`# [4]`)

```
1 name: Articles Branch CI
2
3 on:
4   pull_request:
5     branches:
6       - "main"
7
8 jobs:
```

[*9] https://github.com/WORD-COINS/article-template/tree/main

[*10] フォーク先は残念ながら Private なので見れないですが……

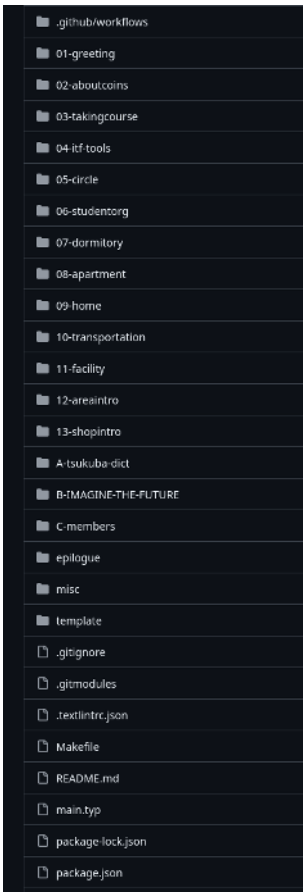


図 2　Coins 新歓パンフのディレクトリ構成

```
build_typst_documents:
  runs-on: ubuntu-latest
  if: contains(github.head_ref, 'articles/')
  steps:
    - name: Create PAT Token
      uses: actions/create-github-app-token@v1
      id: app-token
      with:
        app-id: ${{ secrets.HELLOCOINS_ACTIONS_BOT_APP_ID }}
        private-key: ${{ secrets.
            HELLOCOINS_ACTIONS_BOT_PRIVATE_KEY }}
        owner: ${{ github.repository_owner }}

    # [3]
```

```
22        - name: checkout
23          uses: actions/checkout@v4
24          with:
25            submodules: recursive
26            token: ${{ steps.app-token.outputs.token }}
27
28        - name: Debug GITHUB_REF
29          run: |
30            echo "GITHUB_REF: ${GITHUB_REF}"
31            echo "Branch: $(echo ${GITHUB_REF#refs/heads/} | cut
                  -d\/ -f 2)"
32            echo "branch=$(echo ${GITHUB_REF#refs/heads/} | cut
                 -d\/ -f 2)" >> $GITHUB_OUTPUT
33        # [1]
34        - name: extract branch name
35          shell: bash
36          run: echo "branch=$( echo ${GITHUB_HEAD_REF#refs/heads
               /} | cut -d\/ -f 2 )" >> $GITHUB_OUTPUT
37          id: extract_branch
38
39        - name: Download Typst
40          run: |
41            wget https://github.com/typst/typst/releases/
                 download/v0.13.0/typst-x86_64-unknown-linux-musl.
                 tar.xz
42            tar -xvf typst-x86_64-unknown-linux-musl.tar.xz
43            mv typst-x86_64-unknown-linux-musl/typst /usr/local/
                 bin/typst
44            chmod +x /usr/local/bin/typst
45        # [2]
46        - name: Build Typst document
47          run: |
48            cd ${{ steps.extract_branch.outputs.branch }}
49            sudo apt update && sudo apt install -y make
                 msttcorefonts
50            make compile
51        # [4]
```

```
52        - name: Get current date and time
53          env:
54            TZ: "Asia/Tokyo"
55          id: date
56          run: echo "date=$(date +'%Y-%m-%d-%H-%M')" >>
                $GITHUB_OUTPUT
57        # [4]
58        - uses: ryand56/r2-upload-action@latest
59          id: upload
60          with:
61            r2-account-id: ${{ secrets.R2_ACCOUNT_ID }}
62            r2-access-key-id: ${{ secrets.R2_ACCESS_KEY_ID }}
63            r2-secret-access-key: ${{ secrets.
                  R2_SECRET_ACCESS_KEY }}
64            r2-bucket: ${{ secrets.R2_BUCKET }}
65            source-dir: ${{ steps.extract_branch.outputs.branch
                  }}/main.pdf
66            destination-dir: ./${{ steps.extract_branch.outputs.
                  branch }}-${{ steps.date.outputs.date }}
67        # [4]
68        - name: Comment PR
69          uses: thollander/actions-comment-pull-request@v3
70          with:
71            message: |
72              ビルドが完了しました。以下のリンクからPDFをダウン
                    ロードできます。(7日間有効)
73
74              ${{secrets.R2_BUCKET_PUBLIC_URL}}/${{ steps.
                    extract_branch.outputs.branch }}-${{ steps.date
                    .outputs.date }}/main.pdf
```

本質的な部分である Typst のビルドについては make をしているのでアレですが、それについて Makefile の一部を以下に貼ります。1 行で済んでおり、それ以外の所に時間を割けて良かったです。

```
1 TYPST = typst
2
3 compile:
```

```
4     $(TYPST) compile --root ../ --input PREVIEW=true main.typ
          main.pdf
```

その後、Cloudflare R2 にビルドした PDF を入れています。そこについて、最初は GitHub Actions Artifacts に突っ込もうとしたのですが、Free でやっていくのは到底無理そうです。[*11]

また、結果のファイルをダウンロードするところが微妙に奥まった所にあったり、じゃあ PR のリンクを取得してコメントすればと思いました。が、リンクを取得するのが大変そうだったため、大人しく Cloudflare R2 に入れることにしました。Cloudflare R2 は 10GB までなら無料なのですが、PDF って普通にデカいのでビビって 7 日で消えるような設定を入れました。

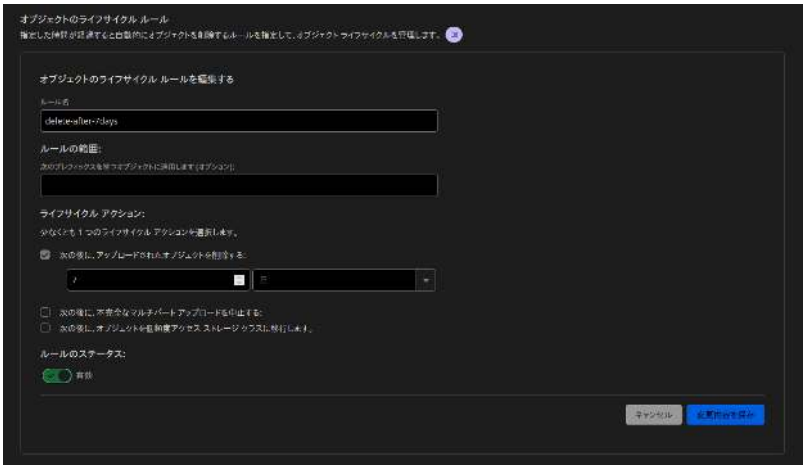


図 3　ライフサイクルルールの設定

`main` ブランチにプッシュしたら、自動的に全記事版をコンパイルするようにもしてほしいです。しかし、commit には何とコメントすることが出来ません。

そのため、R2 を使わず main に Release し、Attach File として PDF を入れるという方式にしました。

また、textlint という自然言語にかける Lint みたいなのも、最低限のルールを追加して、指摘事項をコメントにするという形で入れてみましたが、やかましいだけでロクに役に立たなかったのでやめておいたほうがいいです。

```
1  name: Run textlint and comment
2
3  on:
4    pull_request:
5      branches:
```

[*11] About billing for GitHub Actions - GitHub Docs (https://docs.github.com/en/billing/managing-billing-for-your-products/managing-billing-for-github-actions/about-billing-for-github-actions#included-storage-and-minutes)

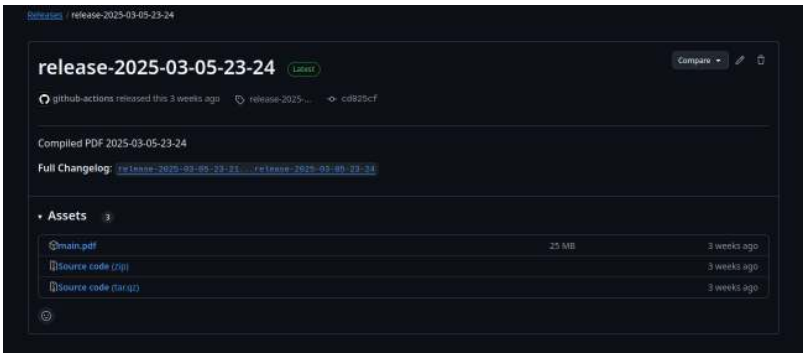


図 4　リリースの様子、このように PDF が配信される

```
      - "main"

jobs:
  run_textlint:
    runs-on: ubuntu-latest
    if: contains(github.head_ref, 'articles/')
    steps:
      - name: checkout
        uses: actions/checkout@v4

      - name: setup node
        uses: actions/setup-node@v4
        with:
          node-version: "22"
          cache: "npm"

      - name: setup project
        run: npm install

      - name: extract branch name
        shell: bash
        run: echo "branch=$( echo ${GITHUB_HEAD_REF#refs/heads
            /} | cut -d\/ -f 2 )" >> $GITHUB_OUTPUT
        id: extract_branch

      - name: run textlint
        id: run_textlint
```

```
32          continue-on-error: true
33          run: |
34            npx textlint ${{ steps.extract_branch.outputs.branch
                  }}/main.typ -o .textlint_output
35            echo 'result<<EOF' >> $GITHUB_OUTPUT
36            cat .textlint_output >> $GITHUB_OUTPUT
37            echo 'EOF' >> $GITHUB_OUTPUT
38
39        - name: Comment PR
40          uses: thollander/actions-comment-pull-request@v3
41          if: ${{ steps.run_textlint.outputs.result != '' }}
42          with:
43            message: |
44              日本語がおかしいところ一覧
45
46              ${{ steps.run_textlint.outputs.result }}
47
48              静的解析ツールによる指摘ですので、参考程度です。気
                    になる方は修正を検討してください。
49        - name: Comment PR
50          uses: thollander/actions-comment-pull-request@v3
51          if: ${{ steps.run_textlint.outputs.result == '' }}
52          with:
53            message: |
54              日本語OK?
55
56              最低限のチェックでしかないので、ご自身でも日本語を
                    今一度確認してください。
```

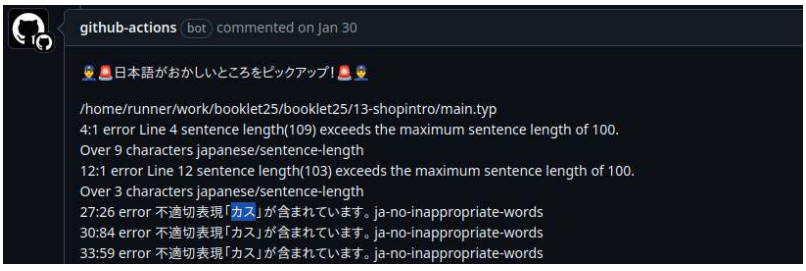


図 5　カスミの「カス」の部分に過剰反応する Textlint

2.3 組版（見た目の制作）

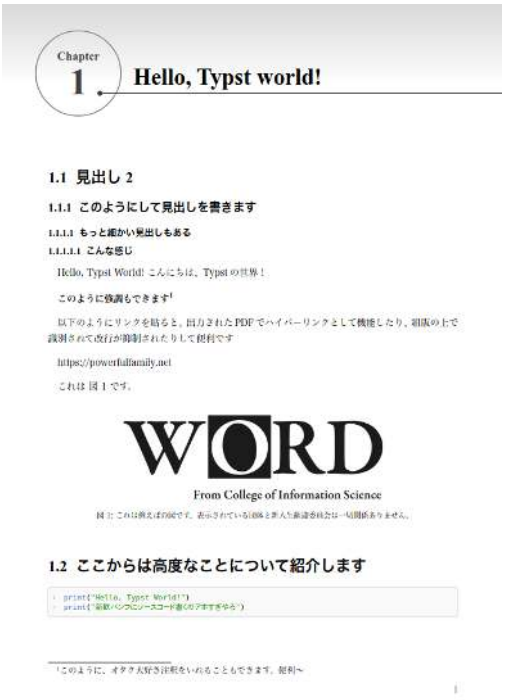


図 6　見た目

まず、大見出しの見た目を決めないといけません。これについては、WORD 編集部に落ちていた「CPU の作り方〜初歩のデジタル回路動作の基本原理と制作〜」を非常に参考にして、Typst で再現をしました。お持ちの方は、ぜひ見ていただければ分かると思います。以下にコードを書きました、h1 だけ異常な量があります。

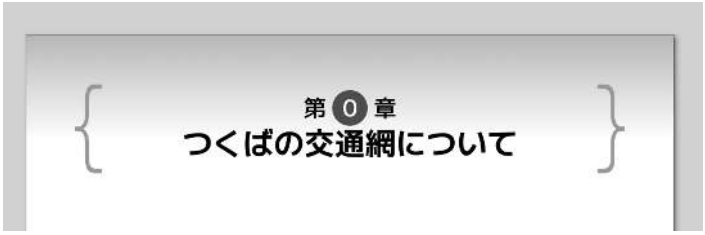


図 7　ボツになった案

```
1  show heading: set text(font: fonts.heading, weight: "bold",
       lang: "ja")
2
3  // h1 をタイトルに
4  show heading.where(level: 1): it => context {
```

```
    let is-appendix = repr(counter(heading).display()).starts-
        with("\"付録")
    pagebreak(weak: true)
    v(10pt)
    place(top, dx: -25%, dy: -14%, box(width: 10000pt, height:
         125pt, fill: gradient.linear(luma(80%), white, angle:
        90deg)))
    place(top + left, dx: -3%, dy: -5%, circle(radius: 50pt,
        stroke: 1.3pt + luma(50%))[
      #align(center + horizon)[
        #stack(
          dir: ttb,
          spacing: 0.75em,
          text(fill: luma(25%), size: 14pt)[
            #if is-appendix [Appendix] else [Chapter]
          ],
          text(fill: luma(25%), size: 43pt)[
            #numbering(if is-appendix {"A"} else {"1"},
                counter(heading).get().at(0))
          ],
        )
      ]
    ])
    place(top + left, dx: 13%, dy: 5.5%, line(length: 100000pt
        , stroke: luma(25%)))
    place(top + left, dx: 12%, dy: 5.1%, circle(radius: 3pt,
        fill: luma(25%)))
    box(width: 100%)[
      #h(100pt) #text(27pt, baseline: -17pt)[*#it.body*]
    ]
    v(50pt)
  }
```

また、以下のような Appendix、付録についても組版に置いて力を入れました。

また、新歓委員メンバーリストなどもあり、その付録について特に凝った組版をしました。

図 8　筑波大学辞典

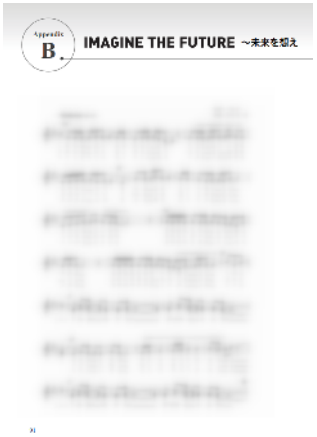


図 9　楽譜

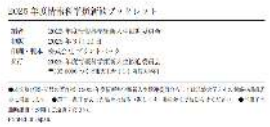


図 10　奥付

3　新歓パンフを支える人達

新歓パンフの執筆と上記にあるシステムの構築は同時進行で行われました。そのため、とりあえず執筆者には Markdown 等の任意の形式で書いてもらい、実は Typst にも対応していた Pandoc で後でいい感じにしようということで進めてもらいました。

新歓パンフレット担当割り

項目名	
1. 挨拶	
2. coinsとは	
3. 展示について	
4. 筑波大学生向けツール各種	
5. サークル等	
6. 学生雑種	
7. 寮舎	
8. アパート	
9. 自宅	
10. 筑波の交通網	
11. 施設紹介	
12. 地区紹介	
13. お店紹介	

※1人1つ以上を担当・思い書換も

執筆方法は問いません。過去版、去年の冊子通りになっていれば問題ありません。
各項目の後ろにコラムとして、自由な執筆を行っていただいて構いません。

図 11　新歓パンフレットの担当割りの様子

組版らへんは最初の方は間瀬がやろうとしていたのですが、やはり深い知識がないと理解できない部分も多く、Ryoga.exe に色々助けを求めていたら、何故か新歓パンフの中の人ではない Typst に詳しい Ryoga.exe が主な担当となって、間瀬と共に行われました。が、最終的には締切直前にも Template や Appendix の編集が行われていました。

また、WORD では出来る Markdown での執筆についても CI の条件が増えるのが面倒で

やりませんでした。普通に詳しい人がファイルをもらい、手元で Pandoc を回し変換していました。

が、その Pandoc も Typst からの変換はポンコツなようで、結局最終的には Typst に手動で変換する、ということが多かったです。

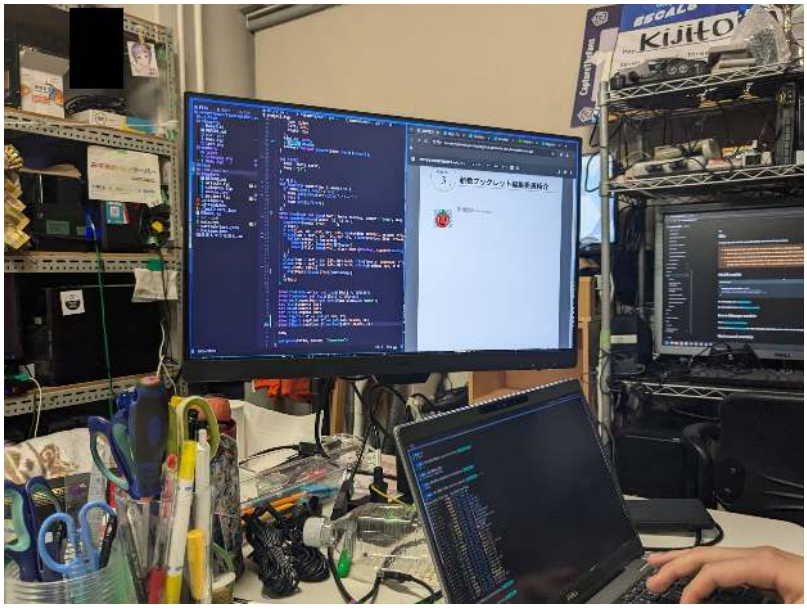


図 12　なぜか新歓パンフの組版をさせられる Ryoga.exe

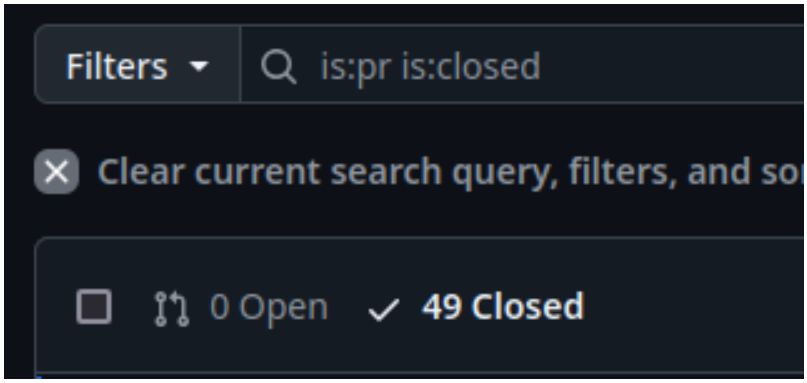


図 13　新歓パンフに出た PR たち、約 50 にものぼった

最後に、もう 30 分ぐらいしか締め切りまでありません、冒頭に出した表紙を間瀬がチャチャッと作ります。

そして、トラブル防止の為、適当に Adobe CC を持っている友人に Illustrator でアウトライン化してもらいます。一応アウトライン化したものとアウトライン化していないものを一緒に ZIP に突っ込み、納品します。何故か 25MB の PDF が 2.5MB になっていますが、時間がありません。

そして、納品日……

画像がガビガビになっていたのが納品されました……[*12]

4　おわりに

こうして 2025 年度の新歓パンフは作られ、新入生のもとへ届けられました。

今回、非常に**計画性**とタスク管理が重要だと痛感しました。特にギリギリで何かをする締め切り駆動というのは危険です！ やめましょう。[*13]

最後まで詰めきれず、悔しいようなところが多くあります。いざ製本された実物を見ると、より「ここをこうしたら、もっと読みやすかったかなぁ」などと感じてしまう今日この頃です。

最後に、ここには紙面の都合上、さらに細かい組版やこだわった点、そしてその具体的な Typst での実装方法は載せていませんが、詳しく知りたいという方はぜひ Ryoga.exe (@Ryoga_exe) に聞いてみてください。授業のある日はラウンジか WORD か AC 部屋にいることが多いです。

[*12] 大変申し訳ありませんでした。

[*13] この記事の赤入れを反映していると同時に、某間瀬 BB は AC 部屋の成果報告書の締め切りに追われているようですが

自転車ことはじめ

文　編集部　Takayuki Ueno

はじめまして。coins22 ／ 3 年の Takayuki Ueno (kichi2004) です。情報科学系の趣味としては Web 開発、競技プログラミングなどで、ほかに音楽ゲーム・FPS ゲームや旅行などが好きです。1 年前の入学祝い号 2024 [*1] に執筆した「クレジットカードのすすめ」という記事も新入生の方には参考になるかと思いますので、興味がございましたらぜひお読みください。

さて、今回はつくばにおける自転車の乗り方について解説していきます。筑波大学において、自転車は「人権」とも言われる重要なアイテムです。ぜひ自転車の乗り方をマスターして、安全で快適な大学生活を送ってください。

また、既に自転車に乗っている方も、後半の交通ルールを改めておさらいしていただき、ぜひ安全運転に努めてください。

1　自転車の購入・整備

1.1　自転車の種類

生活の中で、自転車を使ってコンビニやスーパー等に買い物に行く機会は多いです。そのため、筆者としては、ママチャリやシティサイクルといった、かご付きの自転車を推奨します。

一方で、自転車とは別で乗用自動車や二輪（バイク）を使う場合や家のすぐ近くにスーパーがある場合など、自転車を買い物に使う機会が少ないのであれば、クロスバイク等を購入しても困ることは少ないかと思います。

1.2　自転車の購入場所

筑波大学周辺の店で自転車を購入する場合、次の店舗が候補に上がるでしょう。

井上サイクル　天久保 3 丁目 10-5
Cycle Chic **つくば店**　桜 2 丁目 15-5
サイクルベースあさひ イーアスつくば店　研究学園 5 丁目 19
サイクルベースあさひ つくば店　松代 2 丁目 12-3

また、インターネット通販で購入するという方法もあります。

購入店舗によっては、無料や格安で点検・修理を提供しています。購入店舗を選ぶ際には、立地や品揃えのほかに、保証や修理についてもあわせて検討するようにしましょう。自

[*1] https://www.word-ac.net/post/2024/0409-iwai2024/

転車が故障した際には、徒歩で自転車を持っていくことになる可能性があります。

実家から自転車を持っていく場合／自転車を譲り受ける場合の注意

家族や友人などから自転車を譲り受ける場合や、茨城県外から自転車を持っていく場合には、防犯登録の抹消・登録が必要です。あらかじめ従前の登録の都道府県において、自転車販売店などの防犯登録所で抹消手続きを行った上で、茨城県内の自転車防犯登録所にて新規登録を行いましょう。同じ自転車での新規登録の際には、抹消登録の控えが必要です。

茨城県内で引っ越しする場合にも、防犯登録所での更新手続きが必要になります。

1.3　自転車の整備

茨城県内には、公共施設や民間の店舗などにおいて、空気入れや工具の貸し出しなどを行っている「サイクルサポートステーション」があります。サービス内容は施設・店舗によって異なりますが、自宅や通学経路の近くにある施設を見つけておくと、空気入れ等で便利になるでしょう。

大学の付近では、次の施設・店舗がサイクルサポートステーションとなっているようです[*2]。（2025 年 3 月現在）

ファミリーマート つくば春日四丁目店　春日 4 丁目 15-10
Cafe4　つくば市春日 4 丁目 16-2-1F
つくば駅 関東鉄道学園サービスセンター　吾妻 1 丁目 8-10

そのほか、BiVi つくば 1 階のつくば駅前観光案内所に空気入れがあるほか、大学の 1A 棟 2 階にある学生生活課学生支援でも空気入れを貸し出しているようです。また、「カスミ筑波大学店」（通称：平砂カスミ、平カス）にも電動空気入れが設置されていましたが、こちらは 2024 年夏頃から修理中のため一時的に撤去されているようです。

2　自転車の登録

筑波大学では、学内に駐輪する自転車を登録しなければなりません[*3]。登録の際には、支援室に申請書を提出し、2,000 円を払い込むことにより、自転車に貼り付ける IC タグの交付が受けられます。自転車を買い換えた際にも、再度登録が必要です。

なお、入学手続きの際に申請している場合には、オリエンテーション等で配布されますので、重複申請をしないように注意してください。

3　学内での走行

学内のペデストリアンデッキ等では、特に交通ルールが制定されているわけではありません。合流地点や下り坂では速度を落とし、歩行者や他の自転車にも十分な注意を払いましょう。また、科目間の時間や 6 限後などは自転車の交通量が非常に多くなります。混雑

[*2] https://cycling.pref.ibaraki.jp/ringring/station/?city=tsukuba#station-list
[*3] https://ssc.sec.tsukuba.ac.jp/office-of-student-welfare/bike

時は特に歩行者に気をつけて走行し、自転車同士でも譲り合って通行しましょう。

4　公道での走行

この節では、公道での通行ルールについて説明します。大学構内でも、路線バスや一般の自動車が通る道路については公道になります。

自転車は歩行者ではなく、車両（軽車両）の 1 つです。公道の通行にあたっては、道路交通法等の法令が適用されます。違反した場合には、罰金など刑事罰の対象になる場合もあります。また、自転車に対しての反則金制度（青切符）が 2026 年 5 月までに導入されます。今後、自転車の取り締まりがさらに厳しくなる可能性もあるでしょう。

筑波大学の周辺では、逆走や無灯火、一時停止無視などの交通違反をしている自転車が多く、筆者も遭遇するたびに怖い思いをしています。自転車は、ルールを守って安全に利用しましょう。

ここでは、自転車を運転するにあたっての、道路交通法などの法令を解説します。なお、この記事では、一般的な大学生以外が運転する場合や、筑波大学周辺以外の地域で行われている規制等については書いていない場合があるため、注意してください。

見出しに「★」がある小節については、詳細な解説をした図が記事の最後にあるので、あわせてご覧ください。なお、WORD の Web サイト[*4]上で公開される PDF ファイルでは、フルカラーで掲載されています。

凡例
［法第〇条］：根拠となる法令と条文
〔用語〕：法令上の用語
文字列：筆者が特に重要だと考えている部分
挿入されている図では、印刷の都合により、道路を白色、白線を黒色で表現しています。

4.1　通行方法 ★

自転車は、歩道または路側帯と車道が分かれている道路では、車道を通行しなければなりません。［道路交通法（以下「法」といいます。）第 17 条第 1 項］ただし、歩行者優先で、路側帯（歩道がない道路または道路の歩道がない側の、外側の白線より外側）も通行できます。［法第 17 条の 3 第 1 項・2 項］

自転車は、片側で複数車線がある道路〔車両通行帯の設けられた道路〕ではもっとも左の車線を、それ以外の道路では道路（車道）の左側端に寄って通行しなければなりません。［法第 20 条第 1 項、第 18 条第 1 項］

ただし、自転車レーン〔自転車専用通行帯〕があるときは、その通行帯を通行しなければなりません。［法第 19 条］

自転車は、他の軽車両と横並びで通行〔並進〕してはいけません。［法第 19 条］

4.2　信号 ★

道路を通行する際には、信号機の信号や警察官の手信号に従わなければなりません。［法第 7 条］

[*4] https://www.word-ac.net/

歩道の通行中（後述）または歩行者用信号に「歩行者・自転車専用」と表示されている場合には、歩行者の信号に従う必要があります。［道路交通法施行令第 2 条第 4 項・5 項］

上記の場合を除く、車道の通行中には、車の信号（青信号・直進／左折矢印・黄信号・赤信号）に従う必要があります。［道路交通法施行令第 2 条第 1 項］

4.3　左折・右折 ★

自転車は、左折する際は、道路の左側端に沿って、右折する際には、道路の左側端に寄り、交差点の側端に沿って（二段階右折）、いずれも徐行しなければなりません。［法第 34 条第 1 項・第 3 項］（左折専用・右折専用等の車線〔指定通行区分〕があっても、自転車には適用されません。）

二段階右折の方法の詳細

右折するときは、対面する従うべき信号（青信号または直進矢印）に従って道路を横断し、進路を右向きに変えて停止し、向きを変えたあとに対面する信号に従って進行しなければなりません。（図を参照）

丁字路で向きを変えたあとに道路を横断しない場合（T の下方向から右方向への右折）であっても、信号を無視して進行することはできません。信号のない交差点であっても、停止する必要はありませんが、左端に寄って右折（二段階右折）をしなければなりません。

ただし、丁字路において、T の左方向から下方向への右折をするような場合、二段階右折用の信号機が存在しない場合も多いです。そのような場合にはたいてい、二段階右折時に待機する場所も車道上となり危険です。自転車を降りて歩道を通行するのが安全でしょう。

4.4　一時停止 ★

自転車で通行中に、信号がない横断歩道があり、横断しようとしている歩行者がいる場合には、一時停止して譲らなければなりません。［第 38 条第 1 項・第 2 項］

自転車は、信号のない交差点において、一時停止の標識がある場合には、停止線の手前で一時停止し、交差する道路を通る車両の通行を妨げてはなりません。停止線の直前で停止したあと、交差する交通が見える位置で再度停止し、安全を確認してから交差点に進入しましょう。

4.5　灯火

自転車は、夜間（日没から日出まで）やトンネル内、濃霧がかかっている場所等では、都道府県の公安委員会が定めるライト〔灯火〕をつけなければなりません。［法第 52 条第 1 項、道路交通法施行令第 18 条第 1 項第 5 号、同第 19 条］

茨城県においては、「白色又は淡黄色で，夜間前方 10 メートルの距離にある交通上の障害物を確認することができる光度を有する前照灯」および「橙色又は赤色で，夜間後方 100 メートルの距離から点灯を確認することができる光度を有する尾灯」または基準に適合する反射器材をつけなければならないとされています。［茨城県道路交通法施行細則第 10 条］

4.6 歩道通行の特例 ★

自転車は、次の場合には、歩道を通行することができます。［法第 63 条の 4 第 1 項］

- 「普通自転車等及び歩行者等専用」の標識があるとき
- 車道や交通の状況に照らして、自転車が通行する安全を確保するために、歩道を通行することがやむを得ないとき

歩道の通行が認められる場合には、道路の右側の歩道も通行できます。

歩道が通行できる場合でも、自転車は、歩道の車道寄りの部分を徐行し、歩行者の通行を妨げる場合には、一時停止して譲らなければなりません。また、歩道上に自転車が通行する部分として指定されたところ〔普通自転車指定通行部分〕があるときは、その部分を徐行しなければなりません。ただし、普通自転車指定通行部分では、その部分を通行する歩行者がいないときには、歩道の状況に応じた安全な速度と方法で進行することができます。［法第 63 条の 4 第 2 項］

車道から歩道に入る場合や歩道から車道に出る場合には、前方・後方を目視でしっかり確認し、歩行者や他の車両に十分注意して進路を変更しましょう。

4.7 ヘルメットの着用

自転車を運転する際には、ヘルメットを着用するよう努めなければなりません。［法第 63 条の 11］

4.8 安全運転義務

自転車を運転する際は、ハンドルやブレーキ等を確実に操作し、道路や交通、自転車の状況に応じ、他人に危害を及ぼさないような速度と方法で運転しなければなりません。［法第 70 条］

手放し運転や蛇行運転、傘を差しながらの運転などは危険なのでやめましょう。イヤホンやヘッドホンを装着しながらの運転も、周囲の音が聞こえなくなり危険です。各都道府県の公安委員会による規則でも傘を差しながらの運転やイヤホン等の使用を禁止しています。［法第 71 条第 6 号］

茨城県では、「交通頻繁な道路において，傘を差して自転車を運転しないこと。」［茨城県道路交通法施行細則第 13 条第 2 号］「イヤホン又はヘッドホン（略）を使用して音楽等を聴くなど安全な運転に必要な音又は声が聞こえないような状態で（略）自転車を運転しないこと。」［同条第 15 号］と定められています。

さらに、スマートフォン等を手で保持して、通話のために使用したり、画面を注視したりしてはならないことも定められています。［法第 71 条第 5 号の 5］

全都道府県の道路交通法施行細則を調べたところ、傘差し運転とイヤホン等の使用については次のようになっていました。

傘差し運転　茨城県・栃木県・熊本県では、交通の頻繁な道路では禁止。それ以外の 44 都道府県では、すべての道路で禁止。ただし、京都府では、極めて閑散な道路ではこの限りではない。

イヤホン使用　すべての都道府県で、安全運転に必要な音声が聞こえないような状態での運転禁止。千葉県・兵庫

県・山口県・佐賀県以外の 43 都道府県では、同条文内でイヤホンまたはヘッドホンを例示している。

4.9 飲酒運転の禁止

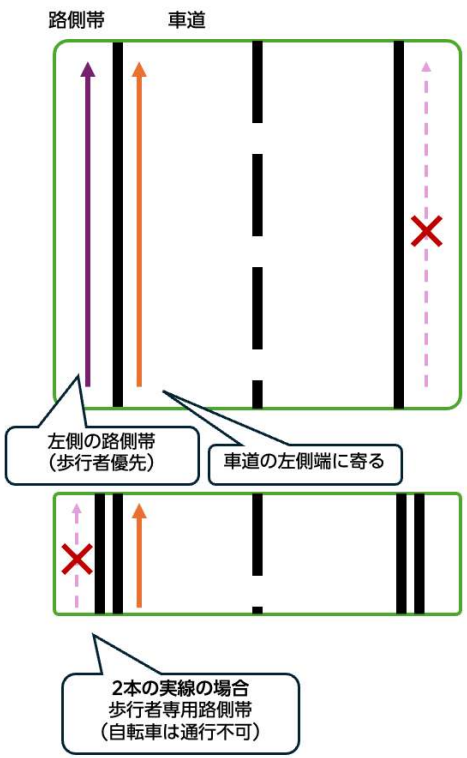
酒気を帯びて自転車を運転してはなりません。［法第 65 条］

4.10 参考となる資料

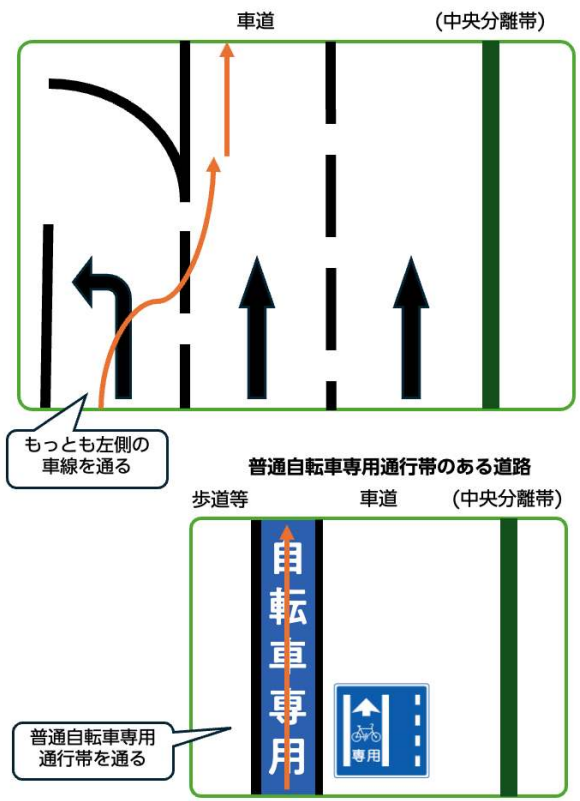
- 自転車は車のなかま～自転車はルールを守って安全運転～（警察庁）：https://www.npa.go.jp/bureau/traffic/bicycle/info.html
- 自転車の安全利用指導マニュアル（三重県警察本部）：https://www.pref.mie.lg.jp/common/content/001164348.pdf
- 道路交通法：https://laws.e-gov.go.jp/law/335AC0000000105

4.1　通行方法

車両通行帯の設けられていない道路



車両通行帯の設けられた道路



4.2　信号

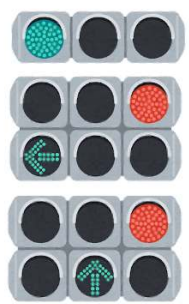
従うべき信号

「歩行者・自転車専用」の信号がある場合

「歩行者・自転車専用」の信号がない場合

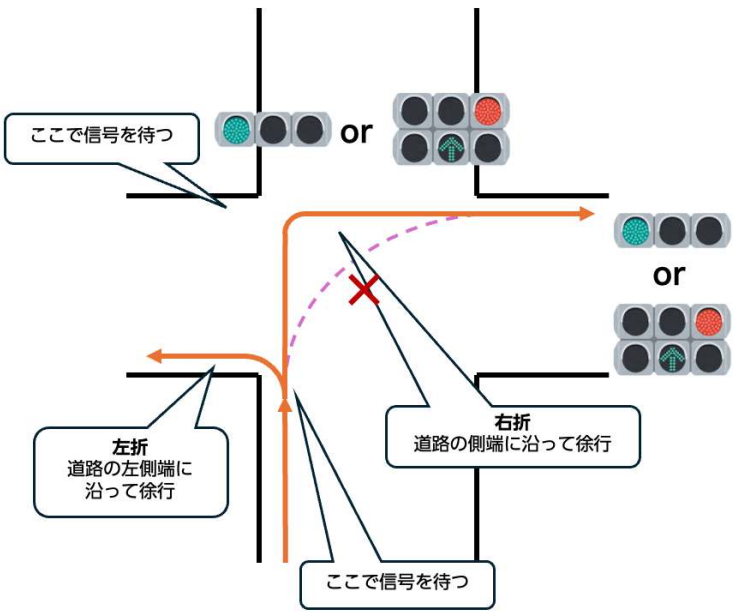
歩道（横断歩道）を
通行するとき

車道を通行するとき



4.3　左折・右折

右左折の方法（原則）

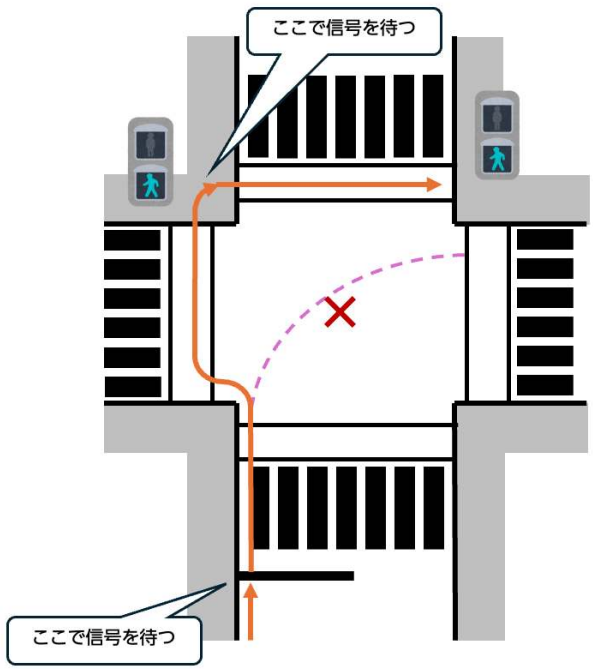


※信号がない場合やでも、実線通りに二段階右折をする

（向きを変えるときの一時停止は不要）

丁字路でも二段階右折をする

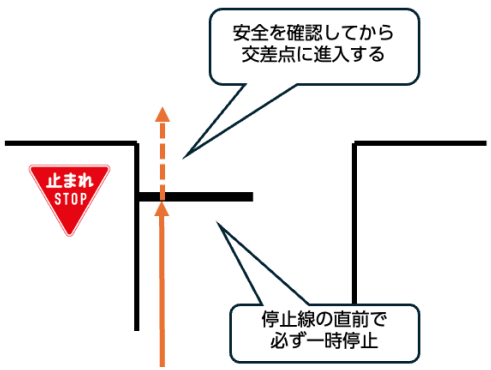
右折の方法（自転車横断帯がある場合）



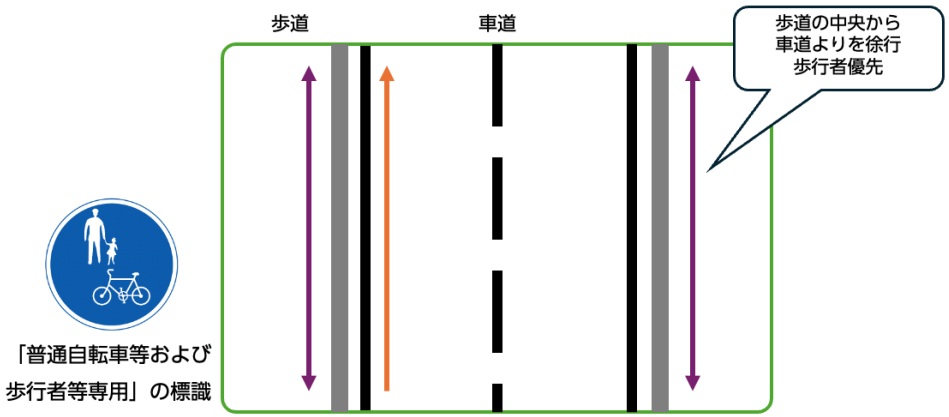
この図通りの右左折が危険だと判断したら、

自転車を降りて、歩道を押して横断しましょう

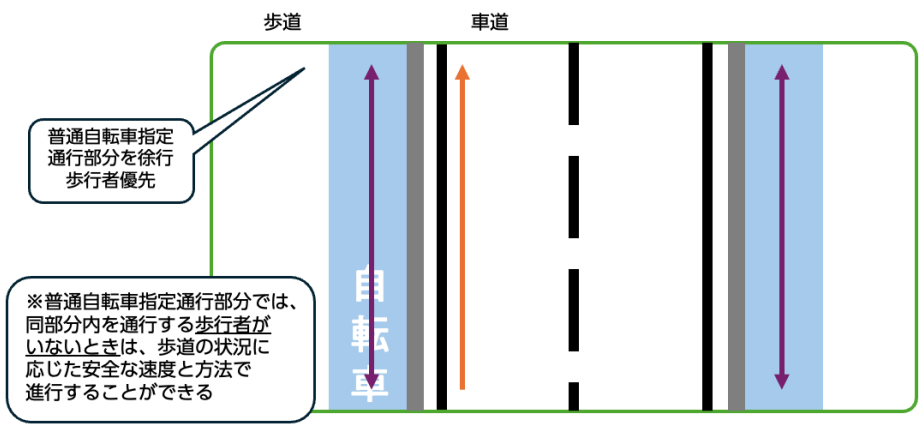
4.4　一時停止



4.6　歩道通行の特例



「普通自転車等および
歩行者等専用」の標識



※普通自転車指定通行部分では、同部分内を通行する歩行者がいないときは、歩道の状況に応じた安全な速度と方法で進行することができる

アニメ初心者におすすめ！ オリジナル TV アニメ作品ガイド

文　編集部　Till0196

1　はじめに

最近、アニメの原作を知っているファンからのネタバレが SNS でよく見られます。これは原作についての話題を盛り上げる部分もありますが、オリジナルアニメには独自の魅力があります。それはストーリーやキャラクターの展開が分からないため、視聴者は毎週新しい驚きを楽しむことができる点です。

オリジナルアニメは、作品の世界を一から作ることができるため、新しいアイデアや意外な展開が生まれやすいです。また、次の展開について SNS などでファン同士で考える楽しみも、オリジナル作品ならではの特徴です。

この記事では、2015 年以降のアニメオリジナル作品を制作会社と放送時期とあらすじとともに紹介しています。

2　青春・学園・恋愛作品

2.1　Just Because!

- **制作会社**：PINE JAM
- **放送開始**：2017 年 10 月
- **あらすじ**：高校 3 年の冬を舞台に、卒業間近の学生たちの恋愛や進路に対する想いが交錯する青春群像劇。
- **主題歌**：
  - OP：「over and over」by やなぎなぎ
  - ED：「behind」by 夏目美緒（CV：礒部花凛）

SNS で定期的に話題になる「月曜日のたわわ」の作者である比村奇石氏がキャラクター原案を担当した作品。入試形態の差などで起こるリアルな人間関係の変化などが描かれている。

2.2　月がきれい

- **制作会社**：feel.
- **放送開始**：2017 年 4 月
- **あらすじ**：中学生の純粋な恋愛を描いた、リアルな心理描写が魅力の作品。
- **主題歌**：
  - OP：「イマココ」by 東山奈央

  - ED：「月がきれい」by 東山奈央

2.3 セイレン

- **制作会社**：Studio 五組
- **放送開始**：2017 年 1 月
- **あらすじ**：各ヒロインごとのオムニバス形式で展開する高校生の恋愛ストーリー。
- **主題歌**：
  - OP：「キミの花」by 奥華子
  - ED：
    - 「瞬間 Happening」（第 1 話 - 第 3 話）by 常木耀（CV: 佐倉綾音）
    - 「アカネ色に隠して」（第 4 話）by 常木耀（CV: 佐倉綾音）
    - 「ムテキの女神」（第 5 話 - 第 7 話）by 宮前透（CV: 下地紫野）
    - 「キミと夜空とフォーチュンと」（第 8 話）by 宮前透（CV: 下地紫野）
    - 「恋のセオリー」（第 9 話 - 第 11 話）by 桃乃今日子（CV: 木村珠莉）
    - 「ふたりふわりら」（第 12 話）by 桃乃今日子（CV: 木村珠莉）

2.4 多田くんは恋をしない

- **制作会社**：動画工房
- **放送開始**：2018 年 4 月
- **あらすじ**：恋をしないと決めた少年と異国の少女のラブストーリー。
- **主題歌**：
  - OP：「オトモダチフィルム」by オーイシマサヨシ
  - ED：「ラブソング」by テレサ・ワーグナー（CV：石見舞菜香）

2.5 つうかあ

- **制作会社**：SILVER LINK.
- **放送開始**：2017 年 10 月
- **あらすじ**：サイドカーレースに挑む少女たちの熱い青春と絆を描く。
- **主題歌**：
  - OP：「Heart to Heart」by スフィア
  - ED：「Angelica Wind」by 宮田ゆり（CV：古賀葵）＆目黒めぐみ（CV：田中あいみ）

2.6 夜のクラゲは泳げない

- **制作会社**：動画工房
- **放送開始**：2024 年 4 月
- **あらすじ**：青春とインターネット文化が交錯する新感覚アニメ。

- **主題歌**：
  - OP：「イロドリ」by カノエラナ
  - ED：「1 日は 25 時間。」by ツルシマアンナ

インターネット世代特有の「何ものにかにならなければならない」という悩みや、各々のクリエイティブへの考え方によりぶつかり合いなどがリアルに描かれた作品。

2.7 ViVid Strike!

- **制作会社**：セブン・アークス・ピクチャーズ
- **放送開始**：2016 年 10 月
- **あらすじ**：格闘技をテーマにした少女たちの熱いバトルと友情を描く。
- **主題歌**：
  - OP：「Future Strike」by 小倉唯
  - ED：「Starry Wish」by 水瀬いのり

2.8 Charlotte

- **制作会社**：P.A.WORKS
- **放送開始**：2015 年 7 月
- **あらすじ**：特殊能力を持つ少年少女たちが織りなす青春と感動の物語。
- **主題歌**：
  - OP：「Bravely You」by Lia
  - ED：「灼け落ちない翼」by 多田葵

2.9 神様になった日

- **制作会社**：P.A.WORKS
- **放送開始**：2020 年 10 月
- **あらすじ**：世界の終わりを予言する少女と少年のひと夏の物語。
- **主題歌**：
  - OP：「君という神話」by Nagi Yanagi
  - ED：「Goodbye Seven Seas」by 麻枝准 × やなぎなぎ

3 音楽・バンド・アイドル

3.1 ガールズバンドクライ

- **制作会社**：東映アニメーション
- **放送開始**：2024 年 4 月
- **あらすじ**：バンドを結成した少女たちが音楽を通じて成長するストーリー。
- **主題歌**：

  - OP：「雑踏、僕らの街」by トゲナシトゲアリ
  - ED：「誰にもなれない私だから」by トゲナシトゲアリ

3.2 バーチャルさんはみている

- **制作会社**：Lide
- **放送開始**：2019 年 1 月
- **あらすじ**：VTuber たちが集まり繰り広げるバラエティ色の強いアニメ。
- **主題歌**：
  - OP：
    - 「AIAIAI（feat. 中田ヤスタカ）」（第 1 話 - 第 6 話）by キズナアイ
    - 「あいがたりない（feat. 中田ヤスタカ）」（第 7 話 - 第 12 話）by バーチャルリアル
  - ED:「ヒトガタ」by HIMEHINA（田中ヒメ + 鈴木ヒナ）

伝説

3.3 ゾンビランドサガ

- **制作会社**：MAPPA
- **放送開始**：2018 年 10 月
- **あらすじ**：ゾンビとなった少女たちがアイドル活動をする異色作。
- **主題歌**：
  - OP：「徒花ネクロマンシー」by フランシュシュ
  - ED：「光へ」by フランシュシュ

3.4 えとたま

- **制作会社**：白組 × エンカレッジフィルムズ
- **放送開始**：2015 年 4 月
- **あらすじ**：干支の擬人化キャラクターたちが繰り広げるコメディバトル。
- **主題歌**：
  - OP：「リトルチャームファング」by にゃ〜たん（CV: 村川梨衣）ほか
  - ED：「blue moment」by ソルラル BOB

4 SF・アクション

4.1 ダーリン・イン・ザ・フランキス

- **制作会社**：TRIGGER × A-1 Pictures
- **放送開始**：2018 年 1 月
- **あらすじ**：巨大ロボット「フランクス」で戦う少年少女のドラマ。

- **主題歌**：
  - OP：「KISS OF DEATH」by 中島美嘉
  - ED：「トリカゴ」by XX:me

4.2 リコリス・リコイル

- **制作会社**：A-1 Pictures
- **放送開始**：2022 年 7 月
- **あらすじ**：特殊エージェント「リコリス」の少女たちが活躍する。
- **主題歌**：
  - OP：「ALIVE」by ClariS
  - ED：「花の塔」by さユり

13 話という話数構成を最大限に活用できている作品。おすすめです。

4.3 Re:CREATORS

- **制作会社**：TROYCA
- **放送開始**：2017 年 4 月
- **あらすじ**：フィクションのキャラクターたちが現実世界に召喚される。
- **主題歌**：
  - OP：「gravityWall」by SawanoHiroyuki[nZk]:Tielle & Gemie
  - ED：「NEWLOOK」by 綾野ましろ

4.4 ラクエンロジック

- **制作会社**：動画工房
- **放送開始**：2016 年 1 月
- **あらすじ**：異世界の侵略者と戦う戦士たちの物語。
- **主題歌**：
  - OP：「STORY」by 小野賢章
  - ED：「盟約の彼方」by 新田恵海

4.5 終末のイゼッタ

- **制作会社**：亜細亜堂
- **放送開始**：2016 年 10 月
- **あらすじ**：魔法の力を持つ少女イゼッタが祖国を守る架空戦記。
- **主題歌**：
  - OP：「cross the line」by AKINO with bless4
  - ED：「光ある場所へ」by May’n

4.6 終末トレインどこへいく？

- **制作会社**：EMT スクエアード
- **放送開始**：2024 年 4 月
- **あらすじ**：終末世界で少女たちが列車で旅する物語。
- **主題歌**：
  - OP：「GA-TAN GO-TON」by 中島怜
  - ED：「ユリイカ」by ロクデナシ

4.7 Lostorage incited WIXOSS

- **制作会社**：J.C.STAFF
- **放送開始**：2016 年 10 月
- **あらすじ**：WIXOSS シリーズの新章、少女たちの運命をかけた戦い。
- **主題歌**：
  - OP：「Lostorage」by 井口裕香
  - ED：「undeletable」by Cyua

4.8 Lostorage conflated WIXOSS

- **制作会社**：J.C.STAFF
- **放送開始**：2018 年 4 月
- **あらすじ**：WIXOSS バトルの最終章、決着の物語。
- **主題歌**：
  - OP：「UNLOCK」by 井口裕香
  - ED：「I」by Cyua

4.9 キズナイーバー

- **制作会社**：TRIGGER
- **放送開始**：2016 年 4 月
- **あらすじ**：痛みを共有する「キズナシステム」に巻き込まれた少年少女たちの青春群像劇。
- **主題歌**：
  - OP：「LAY YOUR HANDS ON ME」by BOOM BOOM SATELLITES
  - ED：「はじまりの速度」by 三月のパンタシア

5 ファンタジー・異世界・バトル

5.1 えんどろ〜！

- **制作会社**：Studio 五組

- **放送開始**：2019 年 1 月
- **あらすじ**：勇者学校に通う少女たちのほのぼのファンタジー。
- **主題歌**：
  - OP：「えんどろ〜る！」by 勇者パーティー
  - ED：「Wonder Caravan！」by 水瀬いのり

5.2 けものフレンズ

- **制作会社**：ヤオヨロズ
- **放送開始**：2017 年 1 月
- **あらすじ**：動物が擬人化した「フレンズ」たちと旅するアドベンチャー。
- **主題歌**：
  - OP：「ようこそジャパリパークへ」by どうぶつビスケッツ ×PPP
  - ED：「ぼくのフレンド」by みゆはん

5.3 装神少女まとい

- **制作会社**：WHITE FOX
- **放送開始**：2016 年 10 月
- **あらすじ**：普通の少女が神の力を得て戦う魔法少女バトル。
- **主題歌**：
  - OP：「蝶結びアミュレット」by Mia REGINA
  - ED：「My Only Place」by スフィア

5.4 宇宙パトロールルル子

- **制作会社**：TRIGGER
- **放送開始**：2016 年 4 月
- **あらすじ**：宇宙を舞台にしたドタバタコメディバトル。
- **主題歌**：
  - OP：「CRY まっくすド平日」by フジロッ久（CV: フジロック）
  - ED：「Pipo Password」by TeddyLoid feat. Bonjour Suzuki

放送当時は「ULTRA SUPER ANIME TIME」枠の中で放送された。個人的には「荻窪」があまりにも印象的なキーワードの作品。

5.5 リトルウィッチアカデミア

- **制作会社**：TRIGGER
- **放送開始**：2017 年 1 月
- **あらすじ**：魔法学校に通う少女たちの成長と冒険を描いたファンタジー。
- **主題歌**：

  - OP：「Shiny Ray」by YURiKA
  - ED：「星を辿れば」by 大原ゆい子

2 クール連続放送で全 25 話構成の作品。

6 日常・コメディ・業界もの

6.1 SHIROBAKO

- **制作会社**：P.A.WORKS
- **放送開始**：2014 年 10 月
- **あらすじ**：アニメ制作現場の舞台裏を描くリアルな業界アニメ。
- **主題歌**：
  - OP：「宝箱-TREASURE BOX-」by 奥井雅美
  - ED：「Animetic Love Letter」by 宮森あおい（CV：木村珠莉）

P.A.WORKS の「お仕事シリーズ」としては第 2 弾の作品。テレビアニメの制作が放送日に間に合わなかった際に SNS で話題となる「万策尽きた」などの流行語を生み出した作品。

6.2 宇宙よりも遠い場所

- **制作会社**：MADHOUSE
- **放送開始**：2018 年 1 月
- **あらすじ**：南極を目指す女子高生たちの冒険と成長を描く感動作。
- **主題歌**：
  - OP：「The Girls Are Alright!」by saya
  - ED：「ここから、ここから」by 玉木マリ（CV：水瀬いのり）ほか

6.3 ひもてはうす

- **制作会社**：バンダイナムコピクチャーズ
- **放送開始**：2018 年 10 月
- **あらすじ**：シェアハウスに住む女子たちの日常をアドリブ満載で描く。
- **主題歌**：
  - OP：「モテたいのー」by ひもてはうすメンバーズ
  - ED：「おうちに帰ろう」by ひもてはうすメンバーズ

6.4 アニメガタリズ

- **制作会社**：ワオワールド
- **放送開始**：2017 年 10 月
- **あらすじ**：アニメ好きの高校生がアニメ研究会で奮闘するコメディ。
- **主題歌**：

  - OP：「アイコトバ」by GARNiDELiA
  - ED：「グッドラック ライラック」by GATALIS

6.5 URAHARA

- **制作会社**：白組 ×EMT スクエアード
- **放送開始**：2017 年 10 月
- **あらすじ**：原宿を舞台に少女たちが街を守るオシャレな日常作品。
- **主題歌**：
  - OP：「アンチテーゼ・エスケイプ」by 春奈るな
  - ED：「KIRAMEKI ☆ライフライン」by 東條詩織（CV: 上坂すみれ）ほか

6.6 石膏ボーイズ

- **制作会社**：ライデンフィルム
- **放送開始**：2016 年 1 月
- **あらすじ**：石膏像がアイドルデビューするシュールギャグ。
- **主題歌**：
  - OP：「星空ランデブー」by 石膏ボーイズ
  - ED：「ワン・オブ・ワンダー」by 石膏ボーイズ

放送当時は「ULTRA SUPER ANIME TIME」枠の中で放送された。

6.7 サクラクエスト

- **制作会社**：P.A.WORKS
- **放送開始**：2017 年 4 月
- **あらすじ**：町おこしのために奮闘する少女たちの物語。
- **主題歌**：
  - OP：「Morning Glory」by (K)NoW_NAME
  - ED：「Freesia」by (K)NoW_NAME

P.A.WORKS の「お仕事シリーズ」としては第 3 弾の作品。

6.8 One Room

- **制作会社**：颱風グラフィックス
- **放送開始**：2017 年 1 月
- **あらすじ**：視聴者視点で展開するショートアニメ。
- **主題歌**：
  - ED：
    - 「春待ちクローバー」by 花坂結衣（CV: M・A・O）
    - 「夏空エール」by 桃原奈月（CV: 村川梨衣）

＊「希望リフレイン」by 青島萌香（CV: 三森すずこ）

キャラクター原案をカントク氏が担当し、1 話 5 分間のアニメ。

6.9 真夜中ぱんチ

- **制作会社**：P.A.WORKS
- **放送開始**：2024 年 4 月
- **あらすじ**：深夜の街を舞台にしたユニークな日常アニメ。
- **主題歌**：
  - OP：「ギミギミ」by りぶ（CV: ファイルーズあい）ほか
  - ED：「編集点」by 真咲（CV: 長谷川育美）

P.A.WORKS の「お仕事シリーズ」としては第 4 弾の作品。YouTuber（作品中では NewTuber）をテーマとしつつ、笑いあり涙あり感動ありの作品。

6.10 ハイスクール・フリート

- **制作会社**：プロダクションアイムズ
- **放送開始**：2016 年 4 月
- **あらすじ**：海洋国家となった日本を舞台に、女子高生たちが艦船で活躍する物語。
- **主題歌**：
  - OP：「High Free Spirits」by TrySail
  - ED：「Ripple Effect」by 春奈るな

第 1 話放送時は「はいふり」という番組名だったため、番組でルール予約をしていた人が 2 話以降の録画に失敗していたことは記憶に新しい作品。定期的に再放送や劇場版が再上映されるなど、放送終了後も根強いファンが多い。

7 おわりに

いかがでしたでしょうか。もし気になるタイトルがあれば、視聴してみてはいかがでしょうか。

情報科学類誌

WORD

From College of Information Science

入学祝い号

発行者　情報科学類長

編集長　川崎晃太朗

制作・編集　筑波大学情報学群
情報科学類 WORD 編集部
（第三エリア C 棟 212 号室）

2025 年 4 月 1 日　初版第 1 刷発行　（128 部）