プログラミング及び実習2

ガイダンス

佐野 彰

本日の内容

1. **科目の概要**（15 分）
    - 到達目標、実施方法、成績評価、質問方法
2. **前期科目との違い**（10 分）
    - タートルグラフィックスの独自環境 → 各 OS 標準の C 言語環境へ
3. **演習課題の進め方**（10 分）
    - aiJudge（オンラインジャッジ）、paiza ラーニング、WebTerm Learn

1. 科目の概要

この科目の到達目標

C 言語（を書く）

- C 言語を使った**基本的なプログラミング**ができる
  - 変数・制御構文・関数・配列・ポインタの基礎
- 前期の独自環境ではなく、**各 OS 標準の開発環境**（コンパイラ・ターミナル）を使いこなす

開発環境・ツール（を使う）

- ターミナル（CUI）でのファイル操作、コンパイル・実行ができる
- エディタ（Visual Studio Code）を使った開発に慣れる
- 「動くコードを自分で書いて確かめる」習慣をつける

> 前期「プログラミング及び実習1」の続きです。C 言語の文法はすでに一度学んでいる前提で、
> **2 年次以降の科目でも使う標準的な開発環境**に慣れることに重点を置きます。

科目の位置づけ

レベル	内容	対応する科目
1. 基盤	計算機の仕組み（ハードウェア、CPU、OS）	情報処理システムⅠ・Ⅱ
2. コーディング	言語の構文、変数・関数・制御構造	プログラミングⅠ・Ⅱ・Ⅲ
3. 計算的思考	アルゴリズム、データ構造	アルゴリズム・データ構造
4. ソフトウェア設計	**部品の組み合わせ、環境構築、開発ツール**	ネットワーク、オブジェクト指向、グラフィックス
5. ユーザー・社会	UI/UX、倫理的・持続的な意義	—

- **AI がレベル 2 を代替**できるようになり、ソフトウェア開発における
  人間の役割は**レベル 4 以上**へシフトしてきている
- ただし技術開発は、**そこだけ分かっていればできる**ものではない。
  自分のメインターゲットの**上下の階層**を理解していないと成り立たない
- 教育もまったく同じ構造です（**教員志望の人は実感できるはず**）
  - 高校数学を教えるには、その上（大学数学）と下（中学数学）が見えていないといけない

科目の実施方法

毎回の流れ（木曜 4・5 講時）

- **4 講時**（約 1 時間）: 講義（Teams でリアルタイム配信・録画あり）
- **5 講時**: 実習（TA が教室で対応）

課題の種類

- aiJudge（オンラインジャッジ）でプログラミング課題を提出・自動採点
- paiza ラーニング・WebTerm Learn の講座受講
- 画像・PDF などプログラム以外の提出物も、すべて aiJudge で受け付ける

提出期限

- 講義日を 1 日目とした **1 週間**

受講方法と質問方法

- ４・５ 講時とも実習室を対面で利用できる
- 講義パートは Teams 会議で配信、録画も閲覧可
- **演習時間中に対面参加している場合** → 実習教室の TA に**直接、対面で**質問
- **それ以外**（オンライン参加時・時間外） → Teams「★ 質問用チャネル」に投稿
  - 時間外の質問も歓迎。返事は気長にお待ちください

成績評価

評価方法

- 演習課題（ex01〜ex14）＋ プログラミング試験 2 回（exam08, exam15）
- 詳しい配点はシラバスを参照

生成 AI の利用

- 演習・デバッグでの利用は**推奨**（コードの解説、エラーの修正、24 時間対応）
- ただし **プログラミング試験での利用は禁止**（Web 検索は可）
  - 試験は教室で実施
- 「自分で簡単なコードを書ける」ことがこの科目の単位取得の条件

本日の目標

- 各 OS 標準の C 言語プログラミング環境を構築する
- ターミナル（CUI）でファイル操作・コンパイル・実行ができる
- paiza ラーニング・WebTerm Learn を使えるようになる
- **aiJudge** にログインして課題を提出できるようになる
- 前期との違い（独自環境 → 標準環境）を理解する

おまけ

生成 AI 時代に、なぜプログラミングを学ぶのか

先にまとめ

1. **AI は本当にコードを書けます。**それを認めた上で話をします
2. 電卓を使えても、**足し算の理解**は要る。同じく AI を使えても、**プログラムの理解**は要る
   —— 別の技能だからではなく、**その道具をちゃんと使うためにこそ**要る
3. **「分解して、組み立てる」**という型は、ソフトウェアに限らず一生使う
4. 実証されていること: AI に丸投げすると**理解が残らない。**
   基礎がある人だけが AI で加速でき、差はむしろ**広がる**
5. 大学の 4 年間は**「専門的な脳みそ」**を作る期間。
   基礎 ＋ プラス α で、**自分だけの個性**を育ててください

ライブデモ: AI はここまでできる

- 目の前で生成 AI（Claude など）に C 言語のプログラムを書かせてみます
- 仕様を渡せば、**テストも自分で書き、デバッグし、動くまでループを回す**
- 誇張でも脅しでもなく、これが 2026 年の現実です

この科目でも生成 AI の利用は**推奨**します（p.8）。
その前提の上で、「では、なぜ自分で学ぶのか」を考えます。

"AI が書けるなら、

学ぶ意味はないのでは？"

まったく正当な疑問です

電卓があるのに、なぜ足し算を学ぶのか

- 電卓は 50 年前から、足し算を人間より速く・正確にこなす
- **計算機を使えること**は、いまや必須の技能 —— それはその通りです
- では、それを学べば**足し算の理解はもう要らない**のか？ 誰もそうは言いません

① **積み上げ**: 足し算の概念が頭にないと、**その上に乗る概念を積めない。**
分数 → 方程式 → 関数 → 微積分 … 各段は、下の段が頭の中にあることを前提にしている。

② **使いこなし**: そもそも**電卓をちゃんと使うため**に、足し算の理解が要る。
入力を打ち間違えても、桁が一つずれていても、**それに気づけない。**

AI とプログラムも、まったく同じ構造

	必須の「使う技能」	代行されない「理解」
これまで	電卓・計算機を使う	足し算そのもの
これから	**AI を使う**	**プログラムそのもの**

- AI を使えることは、これから必須の技能です。**それは大いに学んでください**
- でも**「別の技能だから」**という以上に —— **AI をちゃんと使うためにこそ、理解が要る**
- プログラムを理解する = **計算機の基本的な動作単位を理解する**こと
  - 値を覚える（変数）／条件で分かれる（分岐）／繰り返す（ループ）／まとめて名前をつける（関数）
- この感覚がないまま、高度なゲームやアプリを**設計**できるか？ AI に**的確に指示**できるか？
  （p.5 の **レベル 2 が頭にないと、レベル 4（設計）は積めない**）

「分解して、組み立てる」は一生使う型

ソフトウェア設計に限らず、世の中のあらゆる問題解決は同じ形をしています。

1. 問題を、扱える**小さな部分に分解**する
2. 個々を**処理**する
3. **組み合わせて**、問題全体を解決する

- プログラミングと数学（解法・証明）は、この型を**脳に刻み込むための練習**
- 練習として質が良い理由: **できたかどうかが、ごまかせない形で即座に返ってくる**
  - 作文のように「なんとなく書けた気がする」で済ませられない
- C 言語を一生使わなくても、**この型は残る**（数理・情報科学課程で両方やる意味）

「書けた」と「身についた」は別（実証）

Anthropic の無作為化比較試験（52 名、主に若手エンジニア）

- AI 支援ありのグループは、直後の概念テストが **17% 低い**（約 2 段階の成績差）
- 一方、**作業時間の短縮は約 2 分で、統計的に有意ではなかった**
  - プロンプトを書き、返答を読む手間が、浮いた時間を食ってしまう
- 例外: **「なぜこうなるのか」を AI に聞き返した人**は、理解度が保たれていた

AI を作っている会社自身による研究です。

AI は差を「縮める」のではなく「広げる」

- Gerlich (2025), 666 名: AI 利用が多いほど批判的思考のスコアが低い
  （認知的オフロードがそれを媒介）
  - **若い世代ほど** AI 依存が高く、スコアが低い
- ただし、同じ研究群が示すもう一つの結果:
  - **基礎と自己制御がある人**は、AI で学習を**加速**させる
  - それを持たない人が、最も害を受ける

問いは「AI があるから学ばなくていいか」ではなく、
「AI に加速される側に立つか、AI に食われる側に立つか」。
この科目でやるのは、加速される側に立つための基礎づくりです。

大学の 4 年間は「専門的な脳みそ」を作る期間

- **高校まで**: 全員が同じ内容を学び、**基本的な脳の構造**を作る
- **大学**: 一つの分野の**考え方そのもの**を脳みそに刻む ← 今ここ
- **その先**: その専門性の上に、**自分だけの個性**を育てる

> 足し算と同じで、刻み込むには**自分の頭を通す**しかありません。
> これだけまとまった時間を一つの分野に使える期間は、この先そう多くありません。

ただし、基礎だけでは AI に負けます

- この科目は **1 年生向けの基礎**です。足し算と同じで、ここは外せない
- でも、**「大学の授業でプログラミングを学びました」で止まる人は AI に負けます**
  - 教科書的な、よくある課題 — そこは AI が最も得意な領域そのもの
- 大学のカリキュラムは**「全員に必要な最低限」**を並べたもの
  - 得意技は、カリキュラムの**外側**にしかありません

> IT 業界で生きていくつもりなら、**卒業までの 3 年半**を使って、
> カリキュラムを超えたスキルと脳みそを、**自分で意識して**育ててください。

「プラスαの得意技」を在学中に作る

作る・人に使わせる

- 自分が欲しいものを実際に作る
  （アプリ、ゲーム、Web サービス、bot、自動化スクリプト）
- 作って終わりにせず、**人に使わせて直す**
- GitHub に公開して積み上げる／OSS に PR を出す

動かす・運ぶ

- クラウド（AWS / GCP / Azure）で**デプロイまで**やる
- Linux サーバ、Docker、CI/CD を自分で回す
- データベースを設計して使う

人と組む

- チーム開発（ブランチ運用、レビュー、issue 管理）
- **ハッカソン**、競技プログラミング（AtCoder / ICPC）
- インターン、勉強会、技術ブログ・登壇

武器を足す

- 別の言語を自分で覚える（Python / TypeScript / Rust …）
- **数学を武器にする**（機械学習、最適化、暗号）← この課程の強み
- AI を「使われる側」ではなく**使いこなす・作る側**に回る

参考になりそうな資料

- Anthropic, "How AI assistance impacts the formation of coding skills" (2026) https://www.anthropic.com/research/AI-assistance-coding-skills
- M. Gerlich, "AI Tools in Society: Impacts on Cognitive Offloading and the Future of Critical Thinking", *Societies* **15**(1), 6 (2025) doi:10.3390/soc15010006
- gihyo.jp「AI がコードを生成できる時代の『プログラミング教育』の意味とは？」(2026)
- Qiita「生成 AI 時代のプログラミング学習とコンピュテーショナル・シンキング」