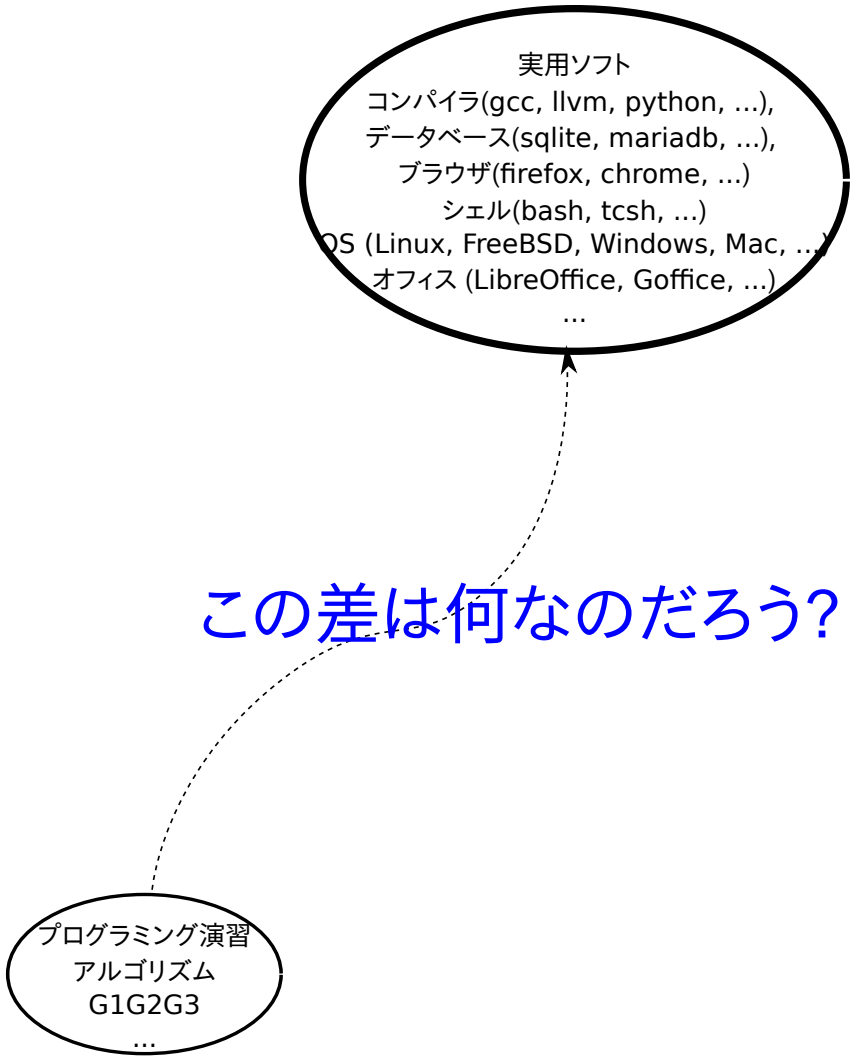
大規模ソフトウェアを手探る

田浦
細川 (田浦研 B4), 佐々木 (田浦研 B4)

この課題の目標

- 「演習レベルの小さなプログラムが作れること」と，「実用規模のプログラムが作れること」のギャップを埋める (ための知識と経験を得る)
- ささやかでも，ソフトをどう改良するかについて，アイデアを出しあう (what と how)



演習レベル vs. 実用ソフト

- 全く違うものか? $\rightarrow$ NO (別に，特別な工場が必要な訳ではない...)
- しかし，演習をこなしていくだけでは，足りないものは確かにある

違い

- 表面的

項目	演習レベル	実用ソフト
ソースファイル	1-数ファイル	多数ファイル/ディレクトリ
ビルド方法	gcc 一回	make その他自動化ツール
依存項目	OS 機能 + 標準ライブラリ関数	OS 機能 + 多数の外部ライブラリ
移植性	不要	要 (ifdef, configure, ...)
拡張性	不要	要
汎用性	そこまで不要	要 (コマンドライン, 設定ファイル, 環境変数, ...)

- 質的: ある程度以上の規模のソフトは，「全てを隅々まで把握して」一行一行を書くわけではない

鍵となる「スキル」

- 全容がよくわからないソフトウェアの概要を把握し，拡張・変更できるようになる

鍵となる「スキル」

- 全容がよくわからないソフトウェアの概要を把握し，拡張・変更できるようになる
- ← そのために，ソースファイル中で修正が必要な場所を突き止める

鍵となる「スキル」

- 全容がよくわからないソフトウェアの概要を把握し，拡張・変更できるようになる
- ← そのために，ソースファイル中で修正が必要な場所を突き止める
- ← そのためのツール
  - 初級: ソース検索 (grep とか) (← これ「だけ」ってのは卒業したい...)
  - 中級: クロスリファレンスツール (gloals とか)
  - 中級: デバッガ (gdb, lldb)
  - 中級: プロファイラ, トレーサ (uftrace)

付随する「基礎知識」

- 多くのソフトをソースからビルドする，決まりきったやり方 (make, configure, その他の，ビルドの自動化ツール)

付随する「基礎知識」

- 多くのソフトをソースからビルドする，決まりきったやり方 (make, configure, その他の，ビルドの自動化ツール)
- 多数のファイルからなるソフトはどう書くのか (C 言語，分割コンパイルの基本)

付随する「基礎知識」

- 多くのソフトをソースからビルドする，決まりきったやり方 (make, configure, その他の，ビルドの自動化ツール)
- 多数のファイルからなるソフトはどう書くのか (C 言語，分割コンパイルの基本)
- 実用ソフトなら常識的なソースの書きかたあれこれ

付随する「基礎知識」

- 多くのソフトをソースからビルドする，決まりきったやり方 (make, configure, その他の，ビルドの自動化ツール)
- 多数のファイルからなるソフトはどう書くのか (C 言語，分割コンパイルの基本)
- 実用ソフトなら常識的なソースの書きかたあれこれ
  - コマンドライン引数，設定ファイルなど，プログラムを汎用的にするための書き方

付随する「基礎知識」

- 多くのソフトをソースからビルドする，決まりきったやり方 (make, configure, その他の，ビルドの自動化ツール)
- 多数のファイルからなるソフトはどう書くのか (C 言語，分割コンパイルの基本)
- 実用ソフトなら常識的なソースの書きかたあれこれ
  - コマンドライン引数，設定ファイルなど，プログラムを汎用的にするための書き方
  - `#ifdef` など，単一ソースを何通りにもコンパイルできるようにするための C 言語の頻出処理

付随する「基礎知識」

- 多くのソフトをソースからビルドする，決まりきったやり方 (make, configure, その他の，ビルドの自動化ツール)
- 多数のファイルからなるソフトはどう書くのか (C 言語，分割コンパイルの基本)
- 実用ソフトなら常識的なソースの書きかたあれこれ
  - コマンドライン引数，設定ファイルなど，プログラムを汎用的にするための書き方
  - `#ifdef` など，単一ソースを何通りにもコンパイルできるようにするための C 言語の頻出処理
  - その他，後々の修正がしやすいように，大きなプログラムがよくやっている技法

これを身につけることの意義

- ソフトウェアの研究では，成果をソフトウェアとして発信することが重要
- しかし大きなソフトウェアを一から作ることは，困難な場合が多い
- → 既存のソフトウェアの拡張として作ることが多い
- 一から作る場合でも, 成果を実用的なソフトウェアとして発信する際，常識的なお作法を守っておくことは重要

課題全体のロードマップ

1. デバッガでのソフトウェアの動作追跡手法を練習 (本日)
2. チームを作る // 手探るソフトを決める
3. 役にたたなくてもいいから変更する案を決める (ミニ発表，議論)
4. 案に従って変更してみる (適宜，進捗発表)
5. なるべくやる気の起きる (≈ 役に立つ) 変更目標を議論して決める
6. 目標に向かって実行
7. 最終発表

オススメの心構え

- この課題では，きっと，各班ごとにずいぶん違うことがチャレンジになる

オススメの心構え

- この課題では，きっと，各班ごとにずいぶん違うことがチャレンジになる
- 大事なのは「知らなくてもナントカなる」という気合

オススメの心構え

- この課題では，きっと，各班ごとにずいぶん違うことがチャレンジになる
- 大事なのは「知らなくてもナントカなる」という気合
- 全てを事前に知っておくなどということは不可能

オススメの心構え

- この課題では，きっと，各班ごとにずいぶん違うことがチャレンジになる
- 大事なのは「知らなくてもナントカなる」という気合
- 全てを事前に知っておくなどということは不可能
- ましてや，「教えておく」のは無理．わからないことの答えが教科書に書いてあるというのは無理だし無意味

オススメの心構え

- この課題では，きっと，各班ごとにずいぶん違うことがチャレンジになる
- 大事なのは「知らなくてもナントカなる」という気合
- 全てを事前に知っておくなどということは不可能
- ましてや，「教えておく」のは無理．わからないことの答えが教科書に書いてあるというのは無理だし無意味
- 作業を効率的にするために，後々有効な方法やツールを腰を据えて学ぶ (その場しのぎで終わらせない), 将来のためのレベルアップを目指す姿勢

オススメの心構え

- この課題では，きっと，各班ごとにずいぶん違うことがチャレンジになる
- 大事なのは「知らなくてもナントカなる」という気合
- 全てを事前に知っておくなどということは不可能
- ましてや，「教えておく」のは無理．わからないことの答えが教科書に書いてあるというのは無理だし無意味
- 作業を効率的にするために，後々有効な方法やツールを腰を据えて学ぶ (その場しのぎで終わらせない), 将来のためのレベルアップを目指す姿勢
- やる気のわく面白い変更・拡張のアイデアをチームで議論し，クラスでも議論する．電車の中や寝ながらも何をしたいか考える

オススメの心構え

- この課題では，きっと，各班ごとにずいぶん違うことがチャレンジになる
- 大事なのは「知らなくてもナントカなる」という気合
- 全てを事前に知っておくなどということは不可能
- ましてや，「教えておく」のは無理．わからないことの答えが教科書に書いてあるというのは無理だし無意味
- 作業を効率的にするために，後々有効な方法やツールを腰を据えて学ぶ (その場しのぎで終わらせない), 将来のためのレベルアップを目指す姿勢
- やる気のわく面白い変更・拡張のアイデアをチームで議論し，クラスでも議論する．電車の中や寝ながらも何をしたいか考える
- だが時間的な制約もあるのである程度で妥協&決心も必要 (そこで萎えない)

オススメの心構え (2)

オレ (私) はそんなに強くないし... という人へ

オススメの心構え (2)

オレ (私) はそんなに強くないし... という人へ

- いわゆる「強い」という印象 ≈ プログラムが書ける, バイトで稼いだり何やら本格的な開発が出来る, ...

オススメの心構え (2)

オレ (私) はそんなに強くないし... という人へ

- いわゆる「強い」という印象 ≈ プログラムが書ける, バイトで稼いだり何やら本格的な開発が出来る, ...
- 正しいプログラムが書けるかどうか, そのためのきちんとした論理的・数学的説明が出来るか, という以外の部分は, それに比べれば「表面的・経験すればどうにかなる話」

オススメの心構え (2)

オレ (私) はそんなに強くないし... という人へ

- いわゆる「強い」という印象 ≈ プログラムが書ける, バイトで稼いだり何やら本格的な開発が出来る, ...
- 正しいプログラムが書けるかどうか, そのためのきちんとした論理的・数学的説明が出来るか, という以外の部分は, それに比べれば「表面的・経験すればどうにかなる話」
- バイト代は稼げなくても, オープンソースソフトの開発に加わる, その手前の経験をこの演習でやる (その過程でツールや, スキルを身につける) ことで, 身につく話

オススメの心構え (2)

オレ (私) はそんなに強くないし... という人へ

- いわゆる「強い」という印象 ≈ プログラムが書ける, バイトで稼いだり何やら本格的な開発が出来る, ...
- 正しいプログラムが書けるかどうか, そのためのきちんとした論理的・数学的説明が出来るか, という以外の部分は, それに比べれば「表面的・経験すればどうにかなる話」
- バイト代は稼げなくても, オープンソースソフトの開発に加わる, その手前の経験をこの演習でやる (その過程でツールや, スキルを身につける) ことで, 身につく話
- 個々の場面で行き当たりばったり, 答えを知れば良しとせず, 今後使えるスキルや道具を手に入れよう, という心構え

今日の残り時間の課題

- gnuplot を以下のように変更する
- キーワード “plot” を省略可能にする．例:

```
gnuplot> sin(x)
```

だけで

```
gnuplot> plot sin(x)
```

の動作をするようにする

代案

- gnuplot を以下のように変更する
- 巨大データを間引く機能

```
gnuplot> plot "hoge.dat" max_samples 10000
```

と書くと，”hoge.dat” に 1 億個の点が含まれていても，10000 だけを等確率で選び出し，表示する (メモリと時間の節約).

そのために練習して欲しい過程

- gnuplot のソースファイルをダウンロード
- gnuplot を，デバッグシンボル有り (-g)，最適化ゼロ (-O0) でビルド (configure, make)
- gdb で動作追跡
- 気が向いたら `htags --suggest` でソース一覧
- ここまでは教科書に懇切丁寧なガイドあり
- (Linux) uftrace を使ってトレースもしてみて (ホームページ → 拙著: ブログ記事 参照)
- その他教科書の付録には
  - 分割コンパイルのための規則
  - makeってなにさ
  - configure について少し
- あとは臨機応変で!

- その他独自案を考えてくれてもいいですが，今日の目的はあくまでソースからビルド，gdb で追いかける練習，ということなので，ハマる可能性のあることに挑戦しなくてもいいです
- ソフト・環境ごとに異なる地雷があるので，まずは穏便に gnuplot (こちらでお手軽なことを確認済み) をオススメ
- メインディッシュは次回以降，自分で決める