実践講習会 #01

環境構築

実践講習会って何やるの？

- ICPC の問題を解けるくらいの考察力を身につける
- コンテストの形式にある程度慣れる
- 上記 2 つは昨年度の目的ですが、
  今年度もこのように行うかは未定です

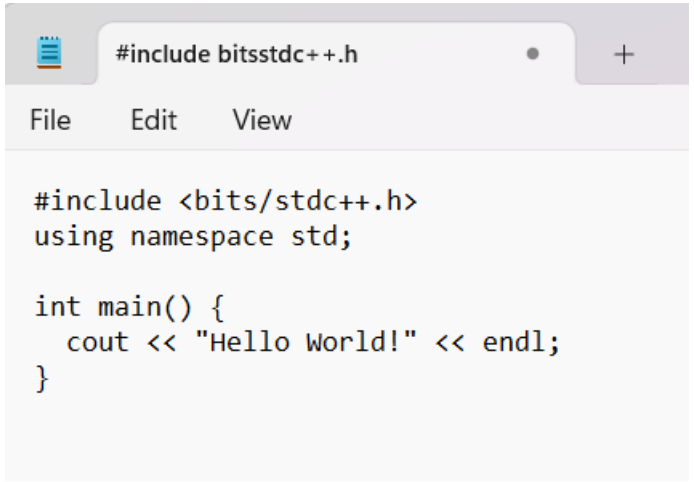
環境構築

環境構築って？

- 作業に適した「環境」を整えること
  - 環境: 何がインストールされているか？設定は？などの構成
- 今回は C++ でプログラミングするための環境を整えます
  - Python などほかの言語を用いるのならそれでも良いですが、
    環境構築や文法などは特に説明はしないので各自でお願いします

なんで環境構築が必要なの？ [1/5]

メモ帳



Visual Studio Code

なんで環境構築が必要なの？ [2/5]

- メモ帳で書くのは読みづらいし書きづらい
  - 例えば、括弧の対応関係が分かりづらい
  - 不要な部分を折りたためない
- テキストエディタを使おう
  - Web 研で VSCode はインストール済み？

なんで環境構築が必要なの？ [3/5]

- AtCoder のコードテスト
  - 512 KiB を超えたら？
  - 標準入力はコピペ？

なんで環境構築が必要なの？ [4/5]

- AtCoder のコードテスト
  - 出力を全部見たい

なんで環境構築が必要なの？ [5/5]

- これらの問題を解決できます
  - 大きいコードになってしまっても実行できます
  - 出力を全部見ることができます
  - (やる気があれば) テストケースを自動実行できます
  - (メモリ・時間の許す限り) 時間をかけて実行できます
  - デバッグが容易になります

環境構築の前に – WSL

- WSL: Windows Subsystem for Linux
- Windows 上で Linux を動かすためのシステム
- Windows だと動かない or 設定が面倒

  → Linux を Windows 上で動かそう！

環境構築の前に – g++

- プログラムを書いても、そのまま実行できるわけではない
- 例えるならば (ほとんどの日本人の場合)
  - 「ياخشىمۇسىز ، ئەڭ چوڭ دۇنيا!» نى بېسىڭ.」してください←？？？？
  - 翻訳「『Hello, Maximum World!』と出力」してください」←わかる
- C++ の場合、この “翻訳機” が g++
  - 他にもありますが広く用いられている g++ を使います

WSL のインストール

1. 管理者権限で PowerShell か コマンドプロンプト を開く
2. `wsl --install` を打って実行
3. 再起動を求められるかも

- 注意事項
  - Windows 11 もしくは Windows 10 Version 2004 以上か確認 `winver`
  - WSL のインストール | Microsoft Learn

g++ のインストール

1. WSL のユーザー設定をする
2. `sudo apt update`
   - 「[sudo] password for ○○」と出たらさっき設定したパスワードを打つ
   - 打ってるのに何も表示されないのは正しい挙動です
3. `sudo apt upgrade`
4. `sudo apt install g++`

macOS でのインストール

1. ターミナルを開く
2. Homebrew をインストール

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
```

3. GCC をインストール

```
brew install gcc
```

4. この記事に従う: macOSでGCCのg++を使用する。 - Qiita

※ macユーザーじゃないので間違ってるかもです

g++ を実行してみよう

- 「g++ --version」と打ってみましょう
- 以下のようなメッセージが出れば OK です

```
asa@Asa-IdeaPadS340:/mnt/c/Users/a01sa$ g++ --version
g++ (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

asa@Asa-IdeaPadS340:/mnt/c/Users/a01sa$
```

コンパイル

- ソースコードをコンピュータが実行できるようにする

  “翻訳” の作業を「コンパイル」といいます

1. `cd <ファイルのあるフォルダ>` で移動します
2. `g++ <ファイル名>.cpp` で「a.out」が生成されます
3. `./a.out` で実行できます

コンパイル例

1. `cd /mnt/c/users/me/programming/maximum/kyopro/01`

   /mnt/.../01 に移動する → C:¥...¥01 と同じ

2. `g++ a.cpp`

   /mnt/.../01/a.cpp をコンパイルする

3. `./a.out`

   /mnt/.../01/a.out を実行する

コンパイルオプション

- コンパイルする際に、設定をしたいときがある
  - 出力ファイルの名前を変えたい！警告出して！など
- オプションを指定するとよい
- `g++ <ファイル>.cpp` につなげて
  - `-o <xxx>`：出力ファイルを <xxx> とする
  - `-Wall`：ある程度警告表示して！

(参考) AtCoder でのコンパイルオプション

- 「ルール」ページにコンパイルオプションが掲載されている
  - APG4b の例: https://atcoder.jp/contests/apg4b/rules

言語	コンパイル・インタプリタ	実行コマンド
C (GCC 9.2.1)	gcc -std=gnu11 -O2 -DONLINE_JUDGE -o ./a.out ./Main.c -lm	./a.out
C (Clang 10.0.0)	clang -std=c11 -O2 -DONLINE_JUDGE -o ./a.out ./Main.c -lm	./a.out
C++ (GCC 9.2.1)	g++ -std=gnu++17 -Wall -Wextra -O2 -DONLINE_JUDGE -I/opt/boost/gcc/include -L/opt/boost/gcc/lib -I/opt/ac-library -o ./a.out ./Main.cpp	./a.out
C++ (Clang 10.0.0)	clang++ -std=c++17 -stdlib=libc++ -Wall -O2 -DNDEBUG -DONLINE_JUDGE -I/opt/boost/clang/include -L/opt/boost/clang/lib -I/opt/ac-library -o ./a.out ./Main.cpp	./a.out
Java (OpenJDK 11.0.6)	/usr/lib/jvm/java-11-openjdk-amd64/bin/javac ./Main.java	/usr/lib/jvm/java-11-openjdk-amd64/bin/java -Xss{stack_size:mb}M Main

おわり

余裕のある人は入門講習会や APG4B の内容を進めましょう！！

次回は 5/11 の予定です