プログラミング基礎演習 2：開発環境構築

柴田祐樹

2020 年 5 月 12 日

概要

本書では Windows あるいは MacOS における Bash シェルを用いた C++ 開発環境の構築例を説明する．URL などは [∗] の様な形式で本書最後にまとめて記載してあるので，参照の際は注意していただきたい．

1　手順概要と背景

OS を操作するユーザインタフェースを Shell と呼ぶ [9]．このうち，マウスクリックなど，幾何学的な操作を用いる Shell を Graphical User Interface (GUI), テキストで記述される命令文でやり取りする Shell を Command-line interface (CLI) と呼ぶ．例えば，ファイルを探すための Windows Explorer [10] や Finder [11] といったアプリケーションは GUI で実装された Shell である．それぞれの特徴としては以下のとおりである．

- GUI の利点
  1. ボタンの操作などキーボード入力に依存しないという操作の一般性がある
  2. 即応性がある（マルティメディア編集と相性が良い）
- CLI の利点
  1. OS を明確に操作できる（使い方の説明が一意に書ける）
  2. 開発コストが低い

- GUI の欠点
  1. 目で見て探すのに時間がかかる（視覚能力に依存する）
  2. 他人に使い方を説明するのが容易ではない
  3. 開発コストが高い
- CLI の欠点
  1. 図や動画などマルティメディア編集が困難

とくに，GUI の欠点，他人に使い方を説明するのが容易ではない，について，教育の時間が少ない大学教育で指導要綱を作りづらいというのは大きな制約である．航空機などの操縦，工場の制御盤など，即座の判断が求められる場面では（ゲームにおけるキャラクタの操縦を含めても良いだろう） GUI が必須であると言えるが，ここでは簡潔な説明のために，Shell としては CLI を基本とすることを推奨し，紹介させていただく．また，CLI のうちでも，2020 年現在 Windows, Mac, Linux 全てで共通して利用可能な Bash [12] を紹介する．

C++ ではコンパイラが必要であるが，ここでは g++ を用いる．これはオープンソースで大変広く普及している C++ 用コンパイラであり，無料で使え，多くの用途に適している．エディタとしては Visual Studio Code [13] が Mac, Linux, Windows で使え，C++ 以外の言語にも対応するなど利便性が高く，現在流行しており Web 上の資料も豊富なためこれを紹介する．Visual Studio Code は GUI である．CLI のテキストエディタも有るが，流石に初学者には厳しいので勧めない．

CLI を利用した C++ の開発環境構築のために，本書で紹介する大まかな手順は，Windows の場合次のとおりである．

1. Windows を Windwos 10 の 1903 までアップデートする

図 1　システム情報表示の例

2. Windows subsystem for Linux を有効化する
3. g++ をインストールする
4. Visual Studio Code をインストールする
5. 試験コードをコンパイルする

MacOS の場合次のとおりである．

1. Mac OS を Catalina 等できるだけ最新版に更新する
2. Xcode をインストールする
3. Finder で home フォルダが見れるように設定する
4. Visual Studio Code をインストールする
5. 試験コードをコンパイルする

Linux の場合はディストリビューションにもよるが，おおよそ以下のとおりである．

1. g++ をインストールする
2. Visual Studio Code をインストールする
3. 試験コードをコンパイルする

各詳細は以降の節で説明する．Windows，Mac それぞれ自分の環境に適したものだけ読んでいただければ良い．第 4 節は全環境に当てはまる内容である．不慣れなうちは面倒で大変な作業であるが，こういったことは今後必須の技能であり，また一度やればあとは開発に集中するだけなので，慣れていってもらいたい．

2　各環境における実際の手順

2.1　Windows

2.1.1　OS の更新

まずは Windows 10 を 1903 以上の版へ更新する．現在の版は，画面左下 Windows マークを右クリックし，システムを選び表示される内容に記載されている．図 1 に表示例を示す．更新自体は Windows update を繰り返し適応しているだけで基本は問題がなかったと記憶している．しかし，それでうまく行かない場合，[6] よりインストール用プログラム（DVD などに焼ける ISO 形式）を取得し，OS の再インストールなどを試してみてもらいたい．

2.1.2 Windows subsystem for Linux の用意

つぎに，[5] を参考に，Windows subsystem for Linux (WSL) を有効化する．WSL とは，Windows 10 以降搭載されている Linux の仮想環境である．研究機関では従来より Unix (Linux は Unix 互換) が使われており，Windows 上で Unix 環境を再現するため Cygwin [7] や [8] を用いるなど様々な努力がなされてきたが，2018 年以降の近年，この WSL を用いることでこのような苦労がある程度なくなってきている．確認した所，本演習で扱う分には問題ないと思われたため，WSL を使うことを推奨する．なお，[5] に示される作業の途中で選ぶ Linux のディストリビューションであるが，特にこだわりがなければ Ubuntu をインストールしていただきたい．

WSL の起動はコマンドプロンプトから bash と入力する方法と，Windows のアプリケーションリストから起動する方法がある．前者は root 権限でログインするためのものなので，通常の利用では後者の方法を勧める．後者の起動方法は，デスクトップ左下の Windows マークを選び，出てくる Ubuntu などと書かれたアイコンをクリックして起動するだけである．

2.1.3 階層の確認

WSL 上のシステムと，Windows 間でファイルをやり取りする方法を説明する．Windows では，最上位階層は C ドライブ等 HDD などに対応した記憶域となっているが，Linux ではルート（/）と呼ばれる階層から始まる．Linux では全ての記憶域はこのルート以下に配置される．ネットワークドライブさえも，ルート以下に配置されるため，一般的にファイルを扱うことができる．WSL 上からみた Windows のドライブもこのルート以下に配置されており，たとえば C ドライブは　/mnt/c/ 以下に配置されている．試しに次のコマンドを実行して，表示される結果から Windows の C ドライブを利用できることを確認してもらいたい

```
cd /mnt/c/
ls
```

2.1.4 g++ の用意

WSL 上で Linux のインストールが終わったら，起動して次のコマンドを実行し，g++ をインストールする．

```
sudo apt update
sudo apt install g++
```

以下のコマンドを実行し，正常にインストールが完了しているか確認する

```
g++ −−version
```

版は違うかも知れないが，以下のように表示されるはずである．

```
g++ (Ubuntu 7.5.0−3ubuntu1˜18.04) 7.5.0
Copyright (C) 2017 Free Software Foundation, Inc.
```

2.1.5 Visual Studio Code の用意とコンパイル試験

サイト [13] から Windows 64 bit 用の Visual Studio Code (VScode) を取得し，インストールする．インストール完了後に起動して，File - New file を選び，新規ファイルを作成状態とする．付録中図 4 の内容を記載した状態で，Ctl + S 等によりファイルを保存するとファイル名と保存場所の決定を促されるため，C:/CSB/test.cpp などとして保存する．Windows の操作に不慣れな人は，わかりやすいように C ドライブ直下に新しい階層を作っておき，そこへ保存すると良い．この例では CSB という階層を作っている．

WSL の Bash から，/mnt/c/CSB へ移動し，コンパイルを行い，実行する．一連のコマンドは次のとおりである．

```
cd /mnt/c/CSB
```

```
ls
g++ test.cpp −O3 −o test
./test
```

ここで，いくらか説明を加えておく．二番目 ls は階層の状態を確認するためのもので必須ではないが，このように確認する癖を付けておいたほうが良い．三番目のコマンドがコンパイルコマンドである．-O3 は最適化オプションで，-o test は出力ファイル名を test とすることを指示している．最後四行目はプログラムを実行するコマンドである．Unix 環境では階層と共に実行ファイルを入力することで実行を指示したことになる．拡張子により動作を識別する Windows とはこの点で異なる．

2.2 Mac での手順

2.2.1 OS の更新

Apple の説明ページ [1] 等が参考になる．注意しておいたほうが良い点は，10.8 以前の版から 10.15 などへは直接アップデートできないため，一度 EL Capitan へ更新してから，最新版へ更新する必要がある．

2.2.2 Xcode の導入

Xcode [2] は Mac における統合開発環境である．Mac 上で動くアプリケーションはもちろん，iPhone 用のアプリケーションなども開発可能である．また，インストールしたときにはエディタとコンパイラ，各種ライブラリがインストールされる．本演習では Xcode を直接使う課題は出さないが，この時インストールされるコンパイラが開発に必要である．インストール方法は [3] などを参考にしていただきたい．数時間など，結構時間がかかるはずである．

後に，以下のコマンドを実行し，正常にインストールが完了しているか確認する

```
g++ −−version
```

版は違うかも知れないが，以下のように表示されるはずである．

```
Apple clang version 11.0.3 (clang−1103.0.32.59)
```

2.2.3 Finder の設定

Finder は Windows で言うところの Explorer であり，ファイル検索，操作のための GUI である．日常の多くの操作とちがい，開発に置いては論理的なフォルダの階層構造を意識して使ってもらう．Finder は初期状態でホーム階層を選べる設定になっていないため，これを表示されるよう，[4] を参考に設定していただきたい．ホーム階層とは，CLI でログインしたときの初期階層であり，ここが GUI から見れないとファイルのやり取りに苦労する．

2.2.4 Visual Studio Code の用意とコンパイル試験

Visual Studio Code (VSCode) は Mac OS でも利用できる．ここではこれをプログラムの編集に用いるものとして紹介する．Xcode でも良いが，VSCode は Windows，Linux でも使えるため，こちらをおすすめする．サイト [13] から，Mac 版を取得し，インストールする．

インストール後，VS Code を Launchpad などを利用して起動し，上部メニューから File - New File と選んで，新しいファイルの編集画面へ映る．エディタ内に，付録中図 4 の内容を記載した状態で，command + S によりファイルを保存する．場所はホームフォルダ（ユーザ名の階層）とし，ファイル名は test.cpp などとすれば良い．その後，端末（ターミナル，terminal）を起動し，以下のコマンドを実行してコンパイル，実行する．

```
g++ test.cpp −O3 −o test
./test
```

Windows と比べたら，Mac は最初から Bash shell が使えるため，環境構築に書ける手間が少し少ない．

あとはバックスラッシュ（Back slash）が打てるように [14] を参考に設定しておくと良い．

2.3 Linux の場合

端末（ターミナル，terminal）を開き，パッケージマネージャ（apt など）で g++ をインストールする．例えば apt の場合は次のとおりである．

```
sudo apt install g++
```

その後， [13] より Linux 用の VS code をインストールし，以下のコマンドで起動する．

```
code test.cpp
```

ファイルには付録中図 4 の内容を記載した状態で，Ctl + S により保存し，再び端末を開き，以下の通りコンパイルして実行する．

```
g++ test.cpp −O3 −o test
./test
```

3 Bash shell について

ここでは Bash shell で使うコマンドを幾らか説明する．シェルのコマンドは，コマンド名と引数からなる．例えば，ファイルをコピーするコマンド cp は次のように利用する．

```
cp <path1> <path2>
```

上記例では，cp がコマンド名，⟨path1⟩ が第一引数，⟨path2⟩ は第二引数とよばれるものである．コマンド名と引数の間は半角スペースを入れて区別する．スペースの数は任意である．以下のコマンドは現在階層のファイルと子階層を表示するコマンドである．これは特に引数を指定しなくても実行できる．

```
ls
```

ファイル容量，隠しファイルも確認したい場合，次のように引数 -la を追加する．

```
ls −la
```

このようななくても動作するが，あればコマンドの動作が変わる引数をオプションと呼ぶ．本書では，⟨path1⟩ のように，⟨ ⟩ で囲われたものは必須の引数，[] で囲われたものはオプションであるとする．以下に基本的なコマンドを幾らか示す．

- cd [dir]: 階層を移動．Change directory の略．dir には階層を指定する．「..」とすれば一つ上位の階層，何も指定しなければログイン時の初期階層へ戻る．
- pwd: 現在の階層をルートから表示
- ls [dir]: 階層の内容を表示．何も指定しなければ現在階層の内容を表示．また，「-la」をオプションとして指定すれば隠しファイル，権限を合わせて表示．「ls -la [dir]」 のように用いる．
- cp ⟨path1⟩ ⟨path2⟩ [-r]: ファイルを複製する．オプション -r をつけることで階層ごと複製される．
- rm ⟨path1⟩ [-r]: ファイルを削除する．オプション -r をつけることで階層ごと削除される．
- mkdir ⟨path1⟩ [-p]: 階層を作成する．オプション -p をつけると多重階層を一度に作成する．
- g++ ⟨path1⟩ [-O3] [-o ⟨path2⟩]: path1 のファイルをコンパイルする．オプション -o の後に出力ファイル名を指定することも可能である．-O3 は高速化の最適化を行うことを指示する．本格的な C++ コードになると，このオプションを指定しない場合，10 倍程度遅い実行コードが生成される．C++ は最適化を前提に設計されているためこの点に注意されたい．

また，以下は便利なコマンドである．ぜひ興味を持ってもらいたい．

- ffmpeg: 音声ファイルや動画ファイルの形式を変換したり，画像を動画にしたりできる．
- python3 ⟨path1⟩: 指定された python3 スクリプトを実行する
- bash ⟨path1⟩: コマンドが一行ごとに書かれたファイル中のコマンドを順に実行する．コンパイルから実行までのような繰り返し使う手順をまとめて記述しておくと良い．

他にもいろいろなコマンドがあるが，最近では Google などの Web 検索エンジンで，「Linux ファイル　コピー」などと調べればだいたい出てくる．エラーメッセージなどが出ればそれをそのまま検索エンジンに貼り付けるだけでも結構な結果が出てくる．特に Stack over flow というサイトは回答に対する相互評価が行われているからか，回答の質が高いため，参考にされるとよい．意外とやりたいことをそのまま打ち込めば有用な結果が出てくる．言葉にするのが難しいと感じる場合，教科書で自習する習慣や，大衆向けのもので構わないので，小説を読む習慣を付けられると良い．また，英語でも検索することを勧める．

参考文献

[1] https://www.apple.com/macos/how-to-upgrade/ [accessed in May., 2020]
[2] https://ja.wikipedia.org/wiki/Xcode [accessed in May., 2020]
[3] https://fukatsu.tech/install-xcode
[4] https://dtmmethod.com/mac-finder-home
[5] https://qiita.com/yukio_tokuyoshi/items/042546812c663ceeccf3
[6] https://www.microsoft.com/ja-jp/software-download/windows10ISO
[7] https://en.wikipedia.org/wiki/Cygwin
[8] https://www.msys2.org/
[9] https://en.wikipedia.org/wiki/Shell_(computing)
[10] https://en.wikipedia.org/wiki/File_Explorer
[11] https://en.wikipedia.org/wiki/Finder_(software)
[12] https://en.wikipedia.org/wiki/Bash_(Unix_shell)
[13] https://code.visualstudio.com/download
[14] https://qiita.com/miyohide/items/6cb8967282d4b2db0f61

付録: 試験コード

付録として試験用コードを幾らか記載しておく．環境構築の際に用いてもらいたい．図 5 は実用的にも役立つ例で，

```
./a.out −−check −−double 231 −−input ”/c/c/a/ a”
```

などと実行してもらえればその動作が分かる．ただし，a.out は実行ファイル名である．

```
#include <iostream>
using namespace std;
int main(){
    cout << ”こんにちは \n”;
    return 0;
}
```

図 2　出力試験コード例

```
#include <iostream>
using namespace std;
int main(){
    int s = 0;
    int i = 0;
    while(i<=100){
        s = s + i;
        i = i + 1;
    }
    cout << ”5050になっているはず. sum = ” << s << ”\n”;
    return 0;
}
```

図 3　総和の例

```
#include <iostream>
using namespace std;
int main(){
    double pi=0;
    double sig = 1;
    for(double i=1;i<=1e7;++i){
        pi = pi + sig/(2∗i − 1);
        sig = sig∗(−1);
    }
    pi = pi ∗4;
    cout << ”pi = ” << pi << ”\n”;
    return 0;
}
```

図 4　円周率計算例（Leibniz の公式）

```
#include <iostream>
#include <string>
using namespace std;
int main(int argc, char* argv[])
{
    bool a=false;
    double b=0;
    string path="";
    string state = "none";
    for (int i = 0; i < argc; ++i) {
        if (state == "none") {
            if (string("--double") == argv[i]) {
                state = "double";
            }else if(string("--check") == argv[i]){
                a = true;
            }else if(string("--input") == argv[i]){
                state = "input";
            }
            else if(string("--version") == argv[i]){
                cout << "cmdline test version 1.0.0\nTMU, Computer science, May 13, 2020\n";
                return 0;
            }
        }else if(state == "double"){
            b = stod(argv[i]);
            state = "none";
        }else if(state == "input"){
            path = argv[i];
            state = "none";
        }
    }

    cout << "a = " << a << " b = " << b << " path = " << path << "\n";
    return 0;
}
```

図 5　コマンドライン引数利用の読み取り