

プログラミング発展

統合開発環境の外側, 最終課題への布石

2023年度2Q 火曜日5~7時限(14:20~16:50)
金曜日5~7時限(14:20~16:50)

工学院 情報通信系

尾形わかは, 松本隆太郎,

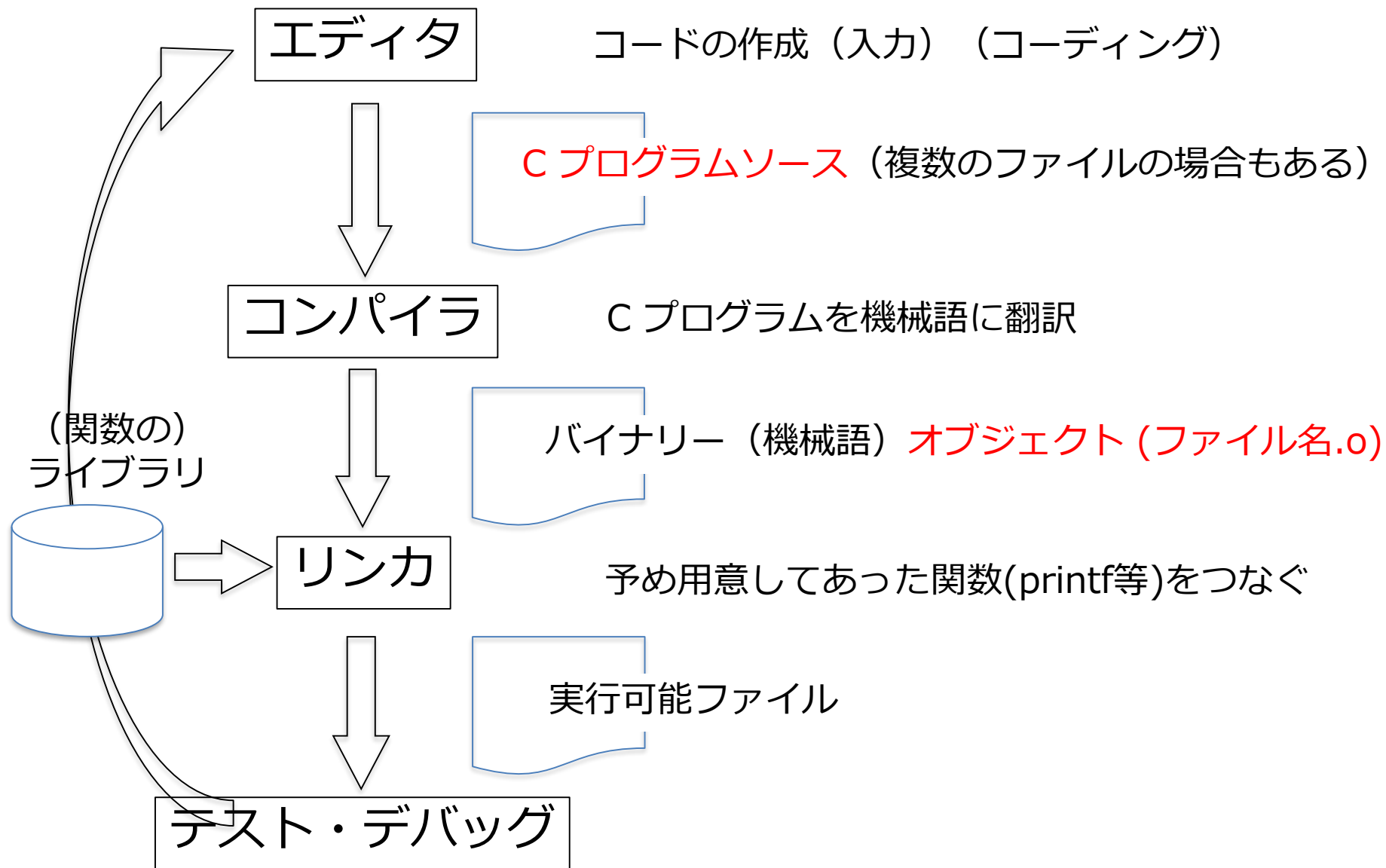
Chu Van Thiem, Saetia Supat

TA:東海林郷志, 千脇彰悟

(最終更新 : 7月17日 12:10)

ヘッダーファイルとライブラリ

プログラミング基礎の最初を思い出して下さい

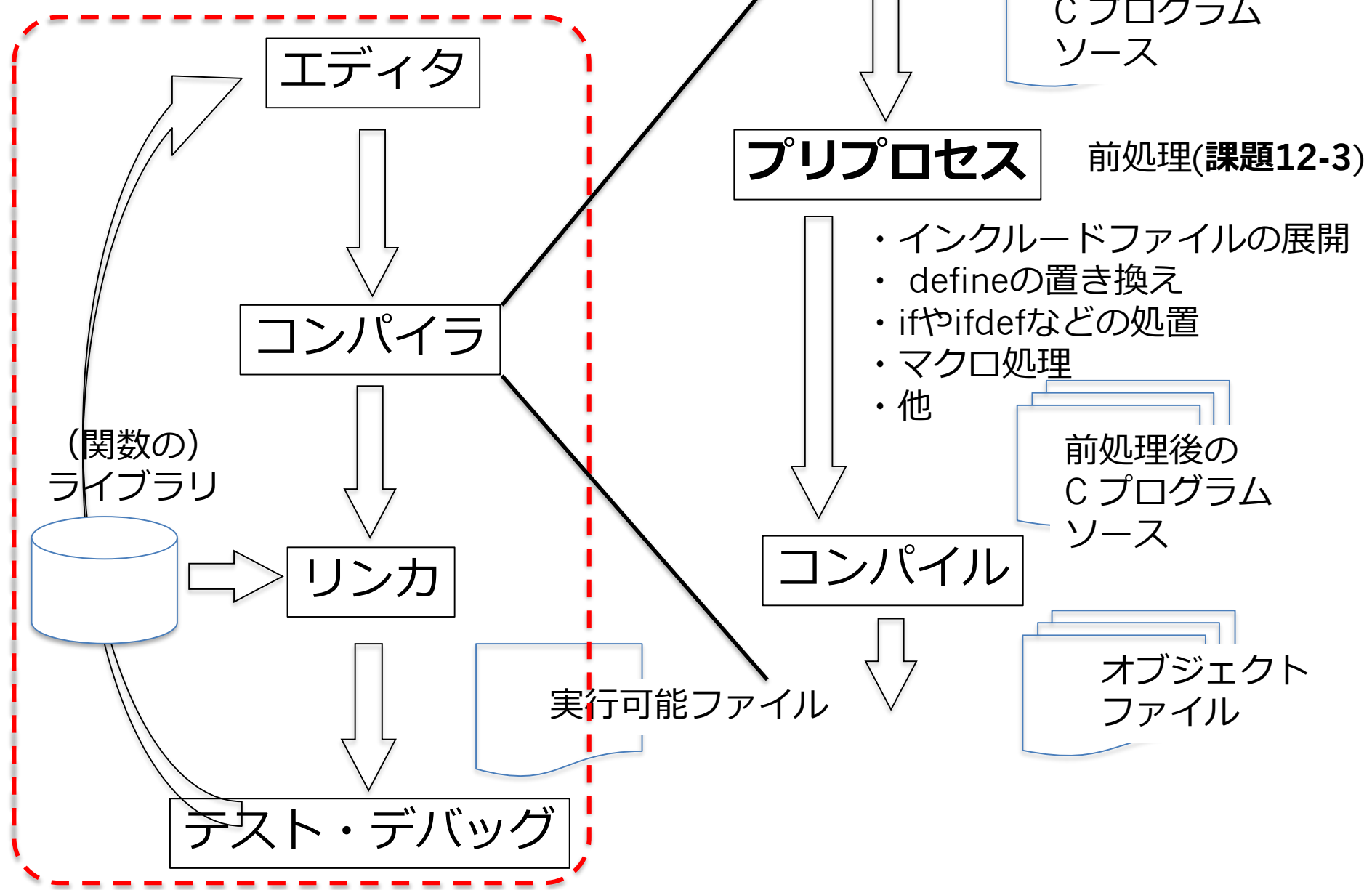


Cプログラムソースとは？

- メモ帳やテキストエディットで正常に表示・編集できるという意味での「テキストファイル」である
- Microsoft Wordで作成したファイルはテキストファイルではない
- Cプログラムソースの中のintやprintfなどは、Ex9-3の解答プログラムを用いると、int, printfなどのように表示される
- Eclipse等で作成した .c ファイルも上記に当てはまる

C言語によるプログラミング～プリプロセスなど～

統合開発環境に含まれる範囲



統合開発環境で防がれる不適切操作

某大学の画像処理の大学院講義で、C++の見本を示し少し改変して実行するように指示したところ

- Word等で作成したファイルをコンパイルしようとしてできない
- コマンドプロンプト（Windowsならcmd、Macならターミナル等）にC++を1行1行入力し毎行エラーを出しながら最後まで入力する

という学生がいて一部の学生が画像処理どころではなかった…という話を聞きました

統合開発環境のメリットデメリット

- 前ページのような不適切な操作をかなりの程度統合開発環境は防止することができる
- 前項とだぶるが、自分が何をしているのか理解していない人にもある程度作業させることができる
- 直観的でわかりやすいユーザーインターフェースを提供できる
(Eclipseはこれに失敗していると思う)
- 自動化が困難である (採点スクリプトやcheck1.shをEclipse, Xcodeで作ることは不可能に近い)
- メモリの無駄遣い (コンパイルして実行するだけなら640MBのメモリでお釣りがくる)

統合開発環境の外側を説明する狙い

- 「情報」と名前に付く学科・系で「自分が何をやっているかわからないけど取り合えず出来ます」という学生を輩出したくないため。理解して使うことが期待されています。
- GUIや統合開発環境が用意されない環境での開発が、みなさんが将来所属する研究・開発現場ではよくあるため（例えば新しい計算デバイスの開発環境を自分が開発する場合など）

統合開発環境の外側を説明する狙い2

- 統合開発環境のようなGUI（グラフィカルユーザーインターフェース）の開発は、WindowsパワースhellやMacのターミナルのようなインターフェース（CUI）と比べて、多大な手間とコストがかかる
- そのため、先進的環境やニッチな環境（Amazon AWS・Google GCEなどのクラウド計算環境やRaspberry Piなどのシングルボードコンピュータ）は、CUI（≒ターミナルのようなもの）を使いこなせないと操作不能である

統合開発環境の外側を説明する狙い3

- また、我々が製作し利用している自動採点システムや check1.sh も CUI を使うことでしか製作・保守できない
- CUI を避けて通ると、他人が商用で提供する出来あいのものの組み合わせ以上のことを計算機にやらせることは不可能に近い
- CUI (cmd のようなもの) が大嫌いなら、出来あいのものの組み合わせを超えることを計算機にやらせずに済むキャリアプランを考えるのがよいと思います

コンパイラを直接起動する

- Eclipse, Xcode などはエディターとコンパイラとデバッガーなどを統一したユーザーインターフェースで用いることができるが、このようなシステムを統合開発環境と呼ぶ
- 今回は、分割コンパイルなどを説明するが、その準備として、エディタ（テキストエディット・メモ帳など）でC言語ソースファイルを直接編集し、直接コンパイラを起動する方法を説明する。

実行可能ファイル

- 計算機（CPU）が直接実行できるデータを格納したファイルは、「実行可能ファイル」と呼ばれる。
- 実行可能ファイルは「ダブルクリック」で実行できる。

コマンドプロンプトやターミナル

- ウィンドウズのコマンドプロンプト（やPowerShellやWindows Terminal）、ならびにMacのターミナルでは、「実行可能ファイル」の名前をキーボードから入力することは、実行可能ファイルのアイコンをダブルクリックするのと同じである
- 直観的に操作できることを面倒くさくしている印象を受けるのは否めない
- 「ダブルクリック」では後で述べる「リダイレクト」や `argv[1]` への代入をできないが、ターミナルではできる。これらはXcode, Eclipse等の統合開発環境でできなくはないが（Eclipseの場合は）面倒くさい

コンパイラの実行可能ファイル名

- MacではEclipseの有無に関わらずコンパイラは「clang」である。Macのgccの実態も実はclangでgccは通常無い
- WSL (Linux) ではgccもclangも両方用意されているがどちらも自分でインストールする必要がある
- 最近ではclangのほうが先進的な機能を取り入れていることが多い
- Windows Visual Studio 2022ではcl.exeである

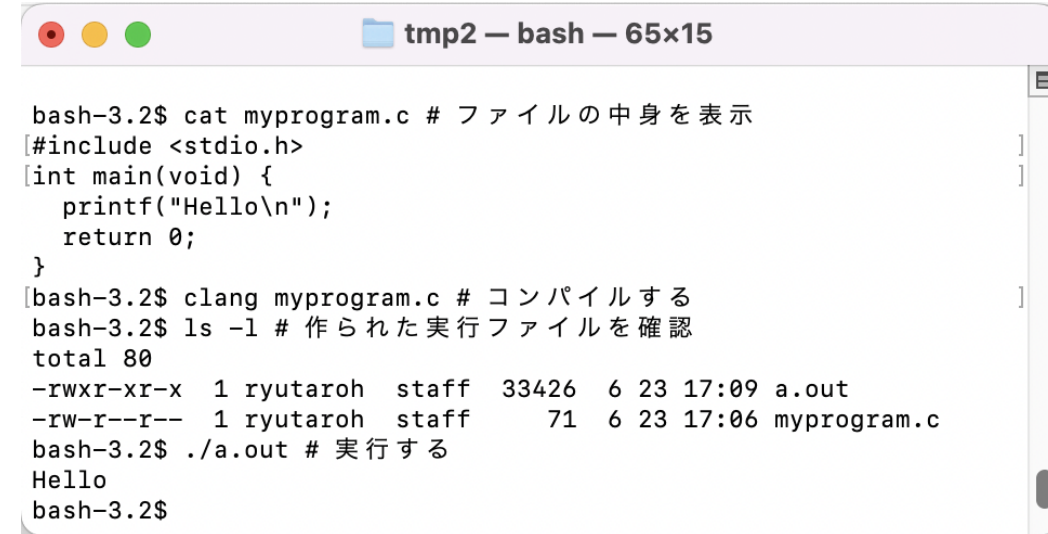
MacOSXでの動作確認（以下はEclipseを使わない手順、Eclipseから起動してもOK）

- 「ターミナル」を起動し「clang -v」とタイプするとAppleが提供するApple Clang（Cコンパイラ）のバージョンが表示される
- 「ターミナル」の起動方法が不明な場合はApple社のサポートページ <https://support.apple.com/ja-jp/guide/terminal/apd5265185d-f365-44cb-8b09-71a064a42125/mac> を見てほしい
- 「gcc -v」とタイプすると「gcc」に偽装したclangがバージョン4.2.1と表示するが、コンパイラ名gccとそのバージョン4.2.1を信用しないこと

テキストエディターでの.cファイルの作成から実行まで

統合開発環境を使わずにコンパイルと実行を行うには、

1. まず、MacOSXのテキストエディット等を用いてCプログラム（ソースファイル）を作成する。
2. 作成したCプログラムmyprogram.cをコンパイルするには、コマンドプロンプト（MACならターミナル）で「clang myprogram.c」とタイプする。
3. 生成される実行ファイルはウィンドウズなら a.exe, Macならa.outになる。
「./a.exe」あるいは「./a.out」とタイプすると、作成された実行ファイルが実行される。



```
tmp2 — bash — 65x15

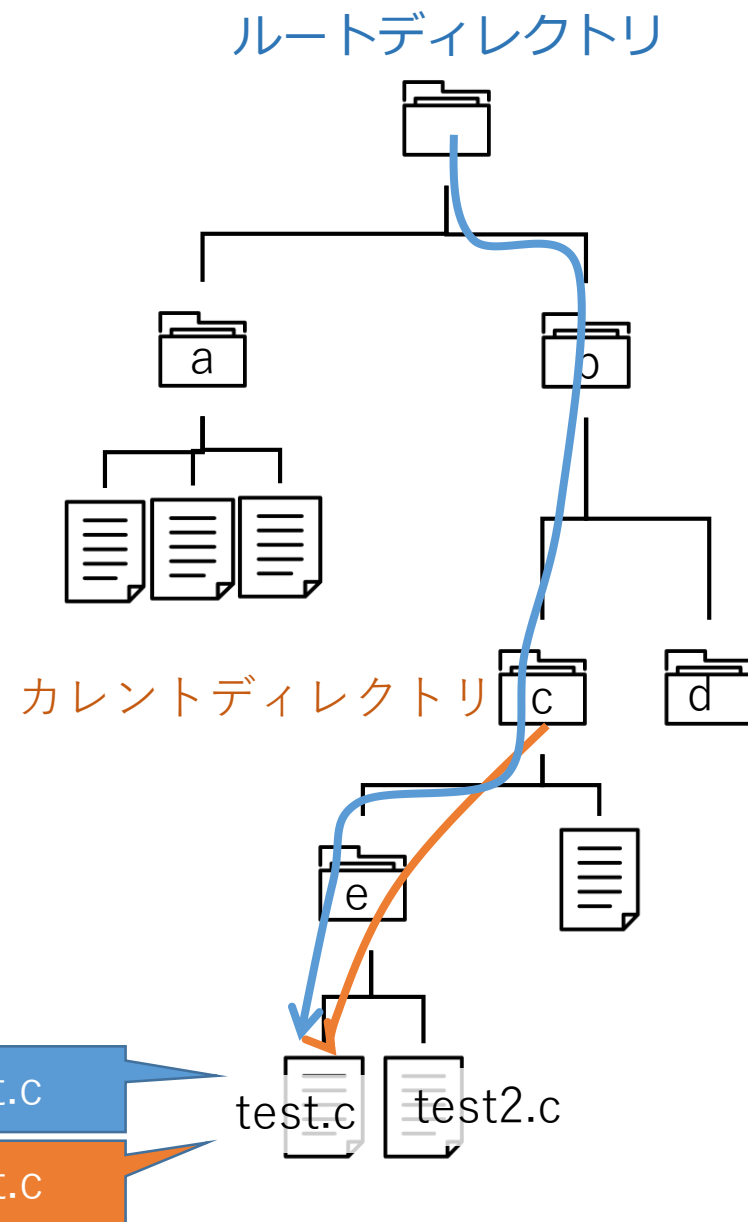
bash-3.2$ cat myprogram.c # ファイルの中身を表示
#include <stdio.h>
int main(void) {
    printf("Hello\n");
    return 0;
}
bash-3.2$ clang myprogram.c # コンパイルする
bash-3.2$ ls -l # 作られた実行ファイルを確認
total 80
-rwxr-xr-x  1 ryutaroh  staff  33426  6 23 17:09 a.out
-rw-r--r--  1 ryutaroh  staff    71   6 23 17:06 myprogram.c
bash-3.2$ ./a.out # 実行する
Hello
bash-3.2$
```

注意事項

- ファイル名は .c で終わる必要がある。Windowsメモ帳ではどうしてもそういう名前のファイルを作成できないため、作成後に手動で名前を変更する必要がある（Macテキストエディットにその不便さは無い）
- コマンドプロンプト（ターミナル）には「カレントディレクトリ」という概念がある。カレントディレクトリは、「cd」コマンドで移動できる。
- ファイルを指定するとき、カレントディレクトリにあるファイルはファイル名のみで指定できるが、それ以外のファイルについては、ディレクトリ名（フォルダー名）から指定する必要がある。
- そこで、ファイルmyprogram.cをコンパイルするときには、「cd」コマンドを用いてカレントディレクトリをmyprogram.cがあるディレクトリまで移動させ、その後「clang myprogram.c」とタイプするとよい
- 詳しくは次ページ以降

ファイルの階層構造とパス名

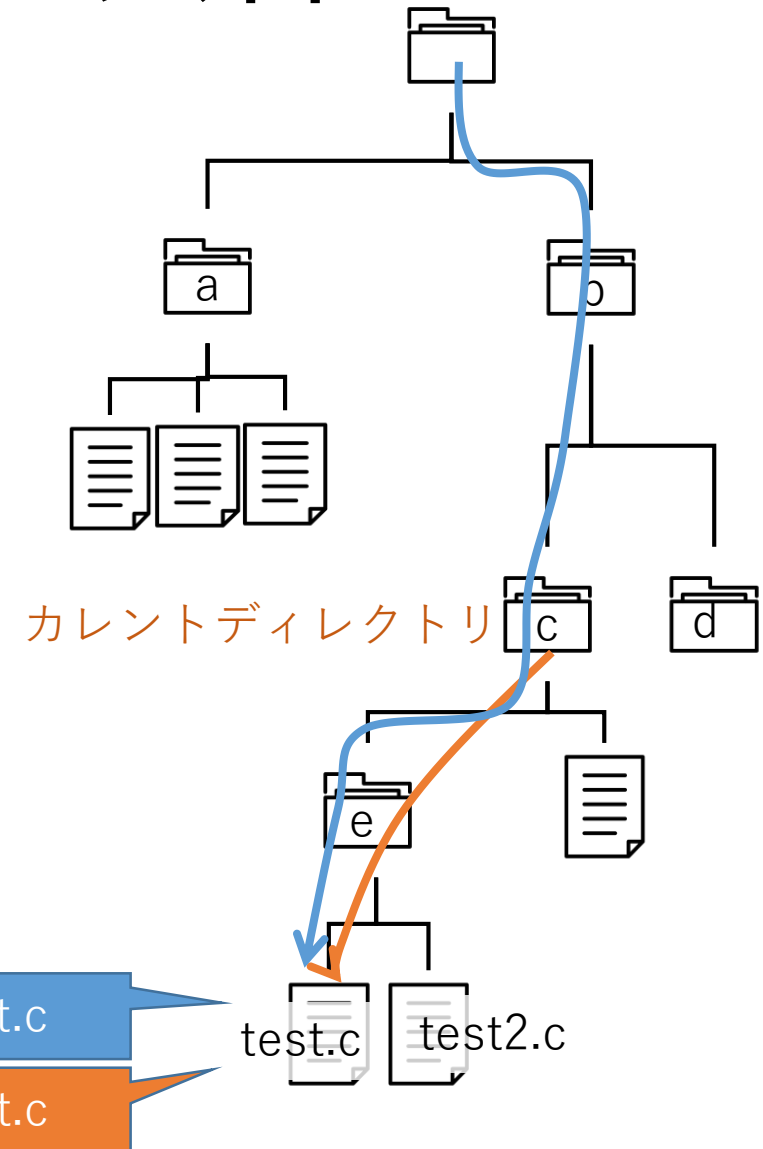
- MacOS(ならびにLinux)では、ディレクトリ（フォルダ）はファイルまたはディレクトリを含み、階層構造にファイルとディレクトリが配置されている
- おおもとのディレクトリをルートディレクトリと呼び、「/（ASCII文字のスラッシュ）」で表す
- ファイル（またはディレクトリ）を表す文字列をパス名と呼び、パス名にはルートディレクトリからそのファイルに至るすべてのディレクトリを列挙した絶対パス（/から始まる）と、/から始まらない相対パスがある



カレントディレクトリと相対パス名

ルートディレクトリ

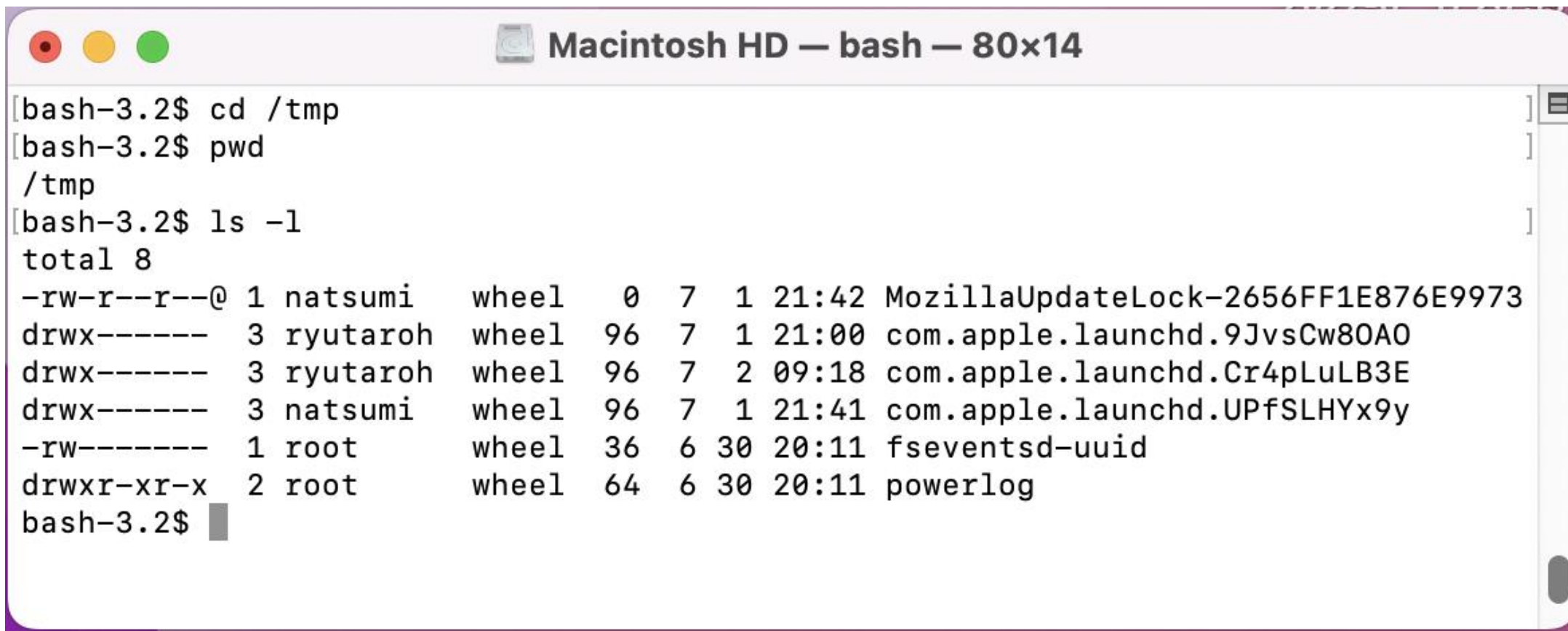
- 相対パス名（／で始まらない、例えばtest.c）は、そのままではどこのディレクトリにあるファイルを指しているのかわからない
- プログラムは実行開始時にカレントディレクトリを定められる。相対パス名は、カレントディレクトリと連結して得られる絶対パス名として解釈される
- ターミナルにおいてカレントディレクトリは「pwd」コマンドで表示できる
- カレントディレクトリの変更は「cd」で行う
- 実行例は次の次のページ



ディレクトリにあるファイル群の表示

- カレントディレクトリにあるファイル群をターミナル内で表示したい場合「ls」コマンドを使う
- 「ls -l」とするとファイルのより詳しい情報を表示する（パワポの余計なお世話でASCII文字のマイナスが謎の文字に置き換えられているからコピペしないこと）
- lsにディレクトリ名を与えるとそのディレクトリ内のファイルを表示する
- 実行例は次ページ

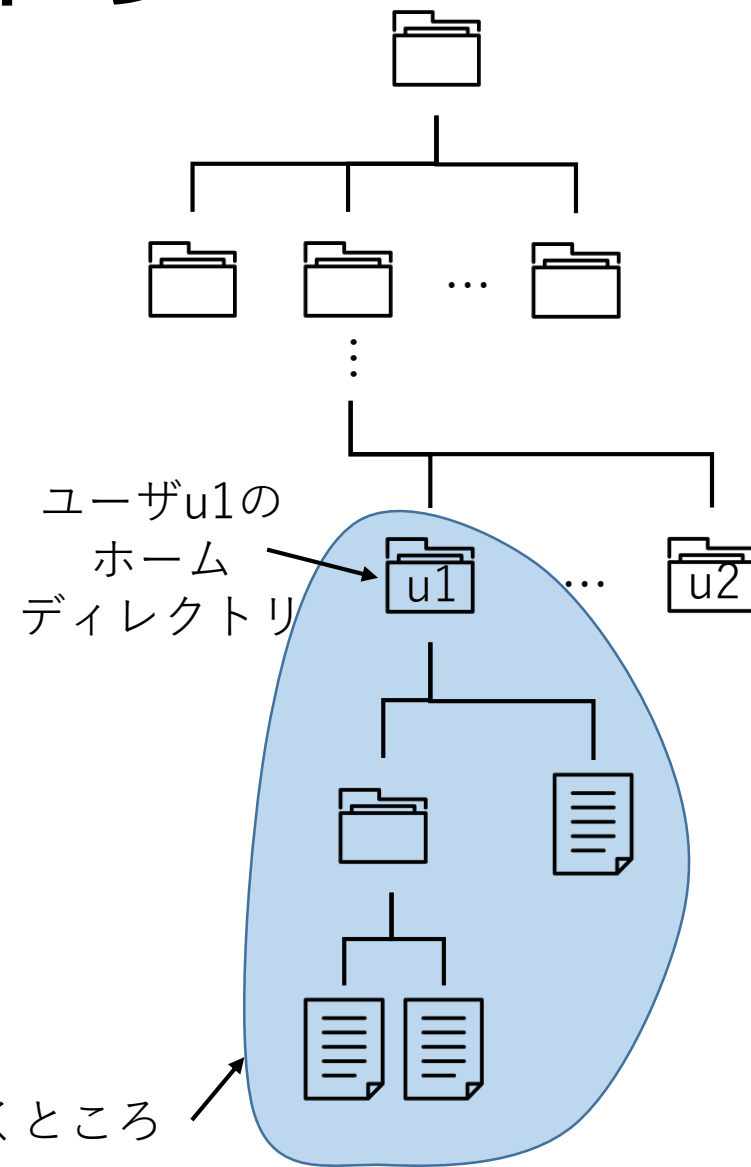
Pwd, lsの実行例



```
[bash-3.2$ cd /tmp
[bash-3.2$ pwd
/tmp
[bash-3.2$ ls -l
total 8
-rw-r--r--@ 1 natsumi    wheel   0   7   1 21:42 MozillaUpdateLock-2656FF1E876E9973
drwx----- 3 ryutaroh   wheel  96  7   1 21:00 com.apple.launchd.9JvsCw80AO
drwx----- 3 ryutaroh   wheel  96  7   2 09:18 com.apple.launchd.Cr4pLuLB3E
drwx----- 3 natsumi    wheel  96  7   1 21:41 com.apple.launchd.UPfSLHYx9y
-rw----- 1 root        wheel  36  6  30 20:11 fseventsd-uuid
drwxr-xr-x  2 root        wheel  64  6  30 20:11 powerlog
bash-3.2$
```

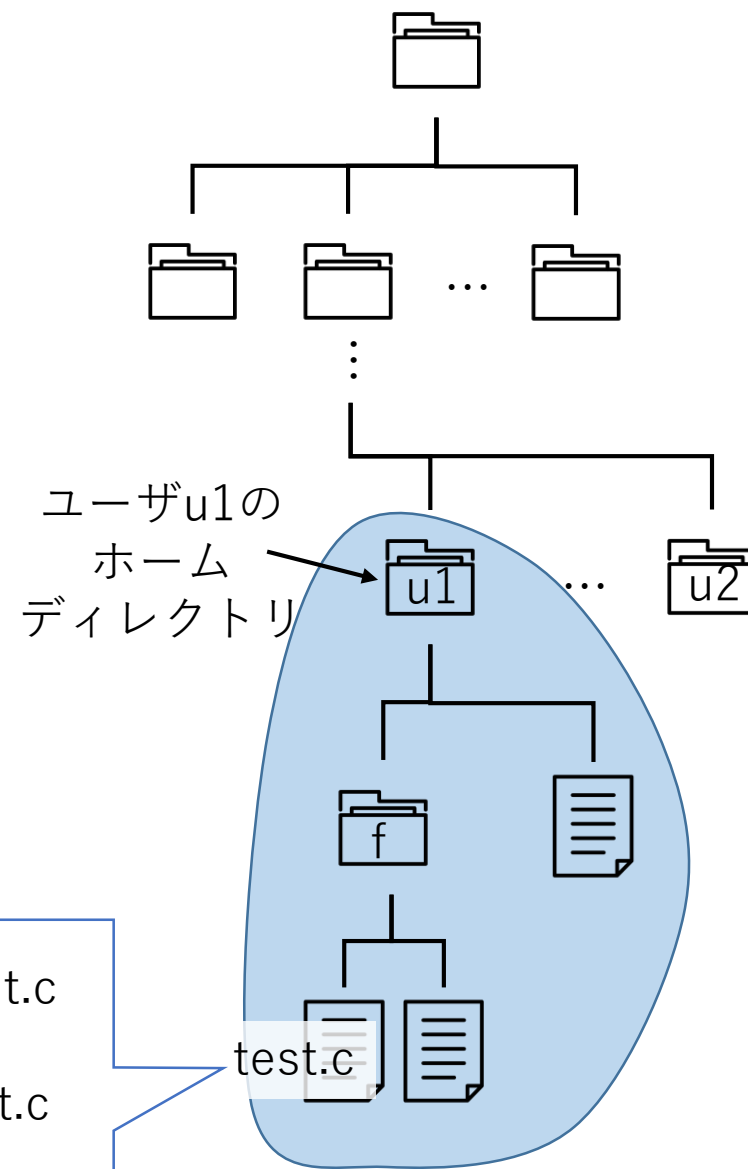
各ユーザーのホームディレクトリ

- MacOSは複数のユーザーで共用できるが、各ユーザーはログイン名で識別されている。ログイン名は「whoami」コマンドで確認できる
- 各ファイルにはその所有者が定められていて、所有者の初期値はファイルの作成者になる。管理者だけが所有者を変更できる。ファイルの所有者は「ls -u」で表示できる
- 各ユーザーが自分のファイルを置くディレクトリとしてホームディレクトリがユーザー毎に定められ、ターミナルから「echo \$HOME」で表示できる（実行例は次の次のページ）



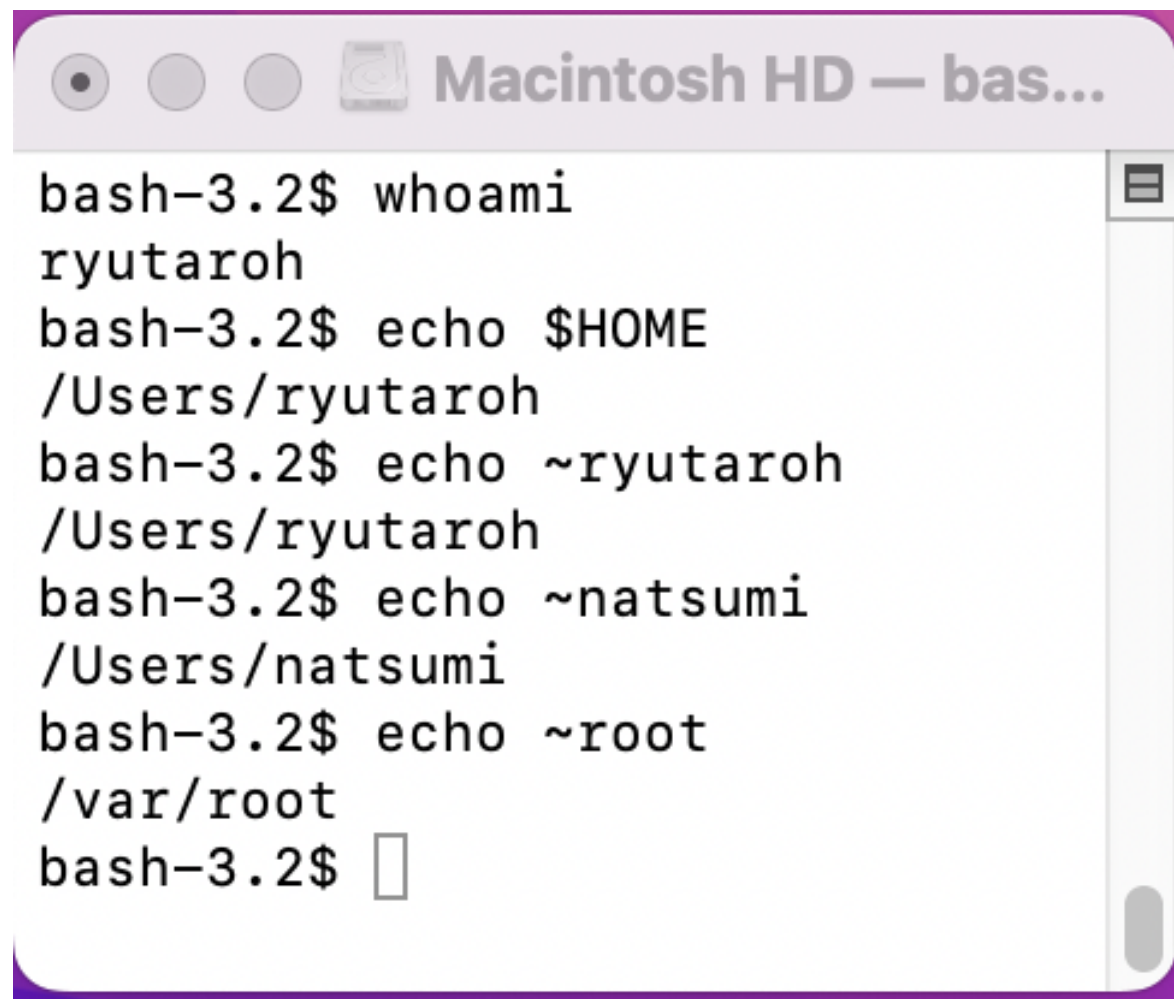
~で始まるパス名

- ~で始まるパス名
(~matsumoto.r.aa/bin/check1.sh等)
は相対パス名に見えるが、実は与えられたコマンドを開始するシェル（MacOSだとzshかbash）が「~ユーザー名」をそのユーザーのホームディレクトリの絶対パス名に置き換えている
- 「echo ~ユーザー名」の実行例は次ページ



絶対パス : / ... /u1/f/test.c
~u1/f/test.c

ログイン名と~で始まるパス名の実行例



```
Macintosh HD — bas...
bash-3.2$ whoami
ryutaroh
bash-3.2$ echo $HOME
/Users/ryutaroh
bash-3.2$ echo ~ryutaroh
/Users/ryutaroh
bash-3.2$ echo ~natsumi
/Users/natsumi
bash-3.2$ echo ~root
/var/root
bash-3.2$
```

特殊なパス名. と. .

- 「.」（ASCII文字のピリオド 1 つ）はカレントディレクトリを表す
- 「. .」（ASCII文字のピリオド 2 つ）はカレントディレクトリの親ディレクトリ（ルートディレクトリに向かって一つ階層を上がったディレクトリ）を表す
- 本スライドで出てくる「. / a . o u t」は「a . o u t」と表すファイルは同一であるが、シェルの仕様上そう入力しないとカレントディレクトリの「a . o u t」を実行しない

カレントディレクトリとfopen()第一引数

- fopen()の第一引数文字列はパス名（絶対パス名または相対パス名）であり、提出時には採点の都合上ディレクトリ名を含めない相対パス名とするように指示されていた
- 課題プログラムのカレントディレクトリは、ターミナルから起動したときはターミナルのカレントディレクトリ（pwdで確認可能）を引き継ぐ
- 統合開発環境から実行した課題プログラムのカレントディレクトリは統合開発環境の種類によりバラバラで、その確認方法は第1回「参考資料」フォルダに与えた

ウィンドウズにおける違い

- パス名におけるディレクトリ区切りに ¥ (ASCII文字) を用いる
- ドライブ名 (C:など) とカレントドライブという概念があり、ドライブ名を含めた絶対パス名は C:¥Users¥松本隆太郎¥Documents¥hello.c のようになる。
- ドライブ名が省略されたパス名は、カレントドライブが補われて解釈される

Cコンパイラ(clang or gcc)の引数

ターミナルからCコンパイラを直接起動する場合、ファイル名（myprogram.cなど）の前に空白文字で区切ってオプション引数を与えることで動作を変えられる。代表的なものを紹介する（パワポがハイフンなどを全角文字に変換しているから以下をコピーしても動かない）

- 最適化：「-O3」は最適化を行い実行ファイルの動作を速くし、「-O0（ゼロ）」は最適化を禁止しデバッグしやすくする
- 警告：「-Wall -Wextra」を付与すると怪しい部分を教えてくれる
- サニタイザ：「-fsanitize=undefined,address」は配列範囲外アクセスなどの未定義動作を実行時に検査をする
- 統合開発環境(ただしWindows Eclipse以外)でも上記のことは可能で、第1回の「参考資料（読まなくてよい）」内の開発環境便利な使い方集で紹介されている

分割コンパイル

- printf(), qsort() などC言語標準関数もC言語で書かれているが、一般使用者はその関数定義を自作プログラムに含める必要がない（必要があったら相当使いにくいですよ…）
- C言語標準関数printfなどの引数や返り値の型などを自作プログラムのコンパイル時にわかるようにするために `#include <stdio.h>` と書いている。前述の「プリプロセス」がstdio.h を自作プログラムに含めている
- 一般使用者も自作プログラムを複数の .c ファイルに分割できる。複数名で開発するときには分割できないと編集が大変

分割コンパイル（続き）

- 総行数が数百万行にのぼるプログラムが1つのファイルになっているとコンパイルに数十分以上かかるから辛い（参考：トヨタのレクサスやみずほ銀行システムの総行数は10億行）
- プログラム全体を複数のファイルに分割して作成し、一旦コンパイルしてすべて .o ファイルにして、その後に更新して変更された .c ファイルだけを改めてコンパイルして .o ファイルを作ると時間がかからない
- このように、プログラム全体を複数のファイルに分割してからコンパイルすることを**分割コンパイル**と呼ぶ。

リンキング

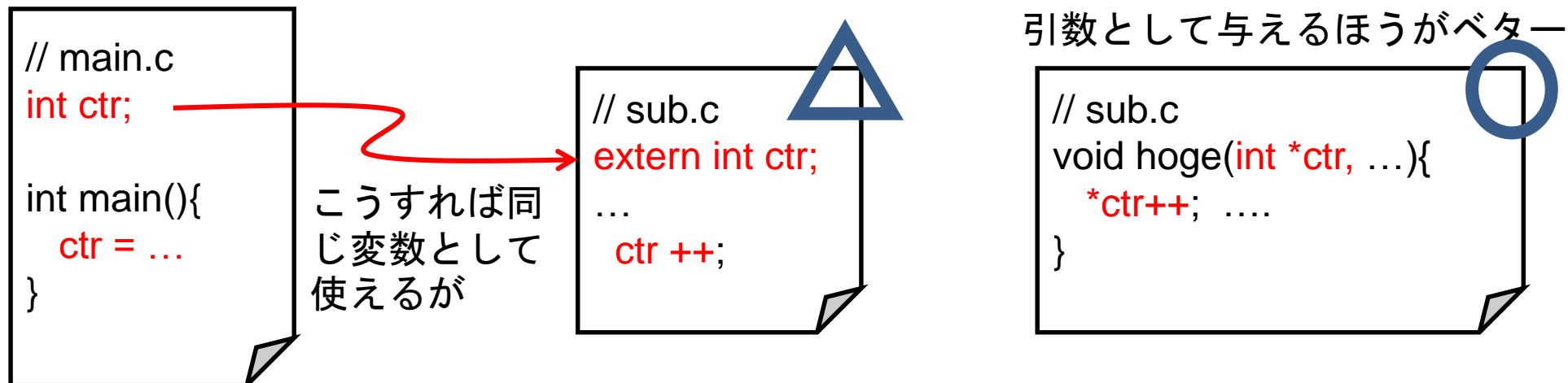
- 複数の .c ファイルや .o ファイルを clang の引数として与えると自動的にくっつけて一つの実行ファイルにまとめてくれる。 くっ付けることを**リンク (link)** と呼ぶ
- なお、clang で実行ファイルを作るときに、fabs(), pow() など math.h で定義される数学関数を使っているなら “-lm” をつけて数学ライブラリをリンクしないとエラーがおきる。Xcode, Eclipse はこれを自動的に行っている
- 統合開発環境でも分割コンパイル・リンキングをできる

名前の衝突と名前空間

- 一つの関数内で同じ名前の変数を2つ宣言すると怒られる
- 一方異なる関数で同じ名前の変数を2つ宣言しても怒られない
- これは関数内で宣言した変数（局所変数）の名前を登録する空間（名前空間, name space）が関数ごとに異なるためである
- 一方、関数名や大域変数は共通の名前空間を用いる
- 複数のファイルで同じ関数名や大域変数名を用いると怒られる

他のファイルで定義される大域変数

- 複数のファイルで同一の大域変数を宣言するのは間違いであるが、変数を宣言しないと使用できない
- ほかのファイルで宣言される大域変数には `extern` をつける。例「`extern int hoka_no_file;`」
- ファイルを跨いで使える大域変数は、その値がどこでどのように変わっていくのか追跡が極めて困難になるから**使わないで済めば使わないほうが良い**。大域ではない変数として宣言し、それを関数の引き数として渡せば大域変数使用を避けられる。
- C言語の`goto`(教えていない)と同様に、大域変数をあえて使ったほうが処理をすっきり記述できることはある



ファイルごとの名前空間

- 大域**変数**（関数の外側で宣言する変数）ならびに**関数**の前に「static」を付けると、その大域変数や関数はファイルごとの名前空間に登録され、異なるファイルから見えなくなり、異なるファイルにおいて同一の名前を使うことができるようになる
- 例： static int file_local_number;
- 関数内でstaticを付けた変数（静的局所変数）と同様に関数外で宣言した変数もstatic有無に関わらずプログラム開始時から終了時まで存在しつづける

```
// main.c
static int ctr;

int main(){
    ctr = ...
}
```

名前は同じだが、
異なる変数

```
// sub.c
static int ctr;
void hoge1(){...
    ctr ++; ...
}
void hoge2(){...
    ctr ++; ...
}
```

これらは同じ変数

関数random()

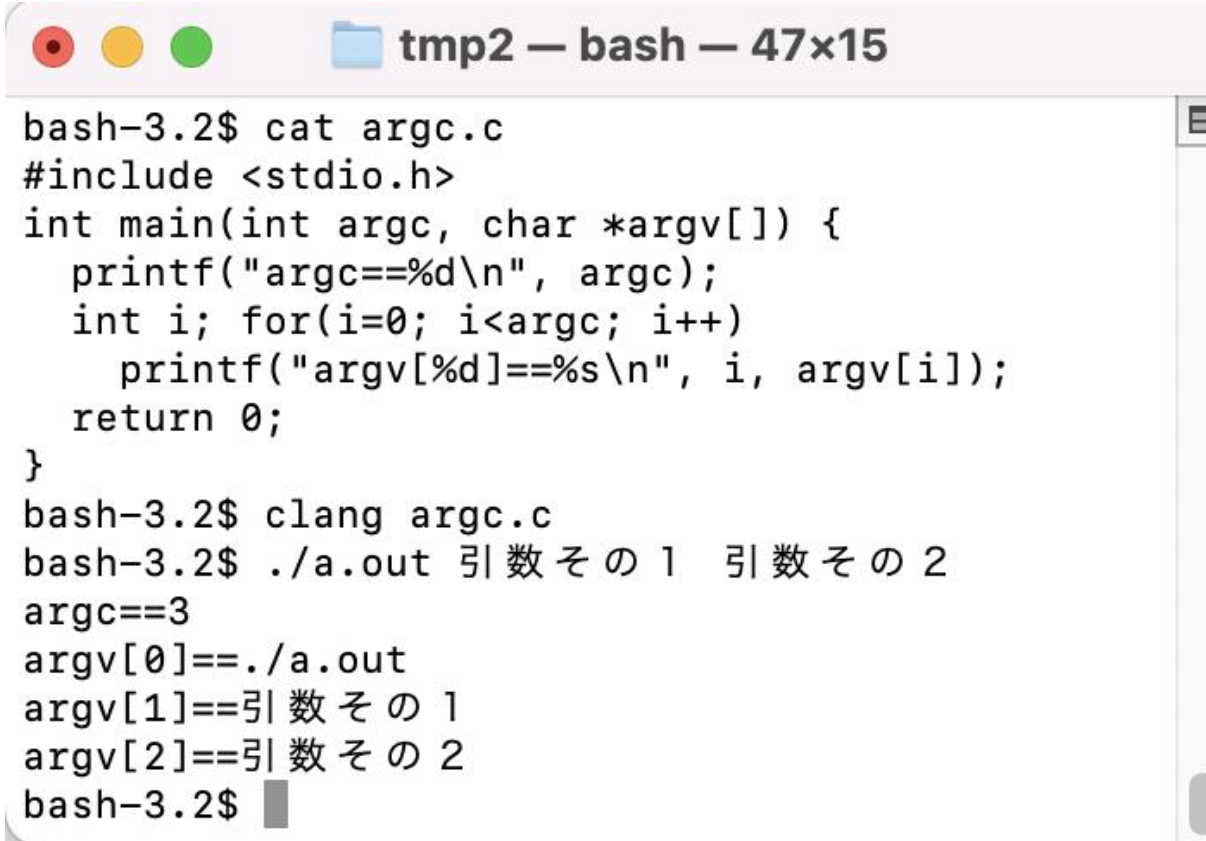
- 以前、自動化採点システムがLinuxで動作していて、LinuxのC言語ではrandom()という関数が定義されていて、提出物でもrandom()を定義されると名前の衝突でエラーになるから使わないようにお願いした
- この不都合は、提出物内でrandom()を定義する場合には必ず「static」を付けるようにお願いすることでも解消することができた

ヘッダーファイル

- プログラムを複数のファイルに分割するときに、ほかのファイルにある関数定義をいちいちファイル内に書くのは面倒くさい
- ファイルを跨いで使える大域変数の `extern` 宣言も面倒臭い
- 関数定義や`extern`による大域変数の定義をひとまとめのナントカ.h として作り各ファイルで `#include` すると手間と間違いが少なくなる。これをヘッダーファイルと呼ぶ
- 自作のヘッダーファイルは `#include "自作.h"` のように<>の代わりに半角二重引用符でファイル名を囲んで使う。このページの二重引用符は半角ではありません

main()の引数

- main()には二つの引数があり、プログラムに渡されたオプションが入る（下記参照）



```
tmp2 — bash — 47x15
bash-3.2$ cat argc.c
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf("argc==%d\n", argc);
    int i; for(i=0; i<argc; i++)
        printf("argv[%d]==%s\n", i, argv[i]);
    return 0;
}
bash-3.2$ clang argc.c
bash-3.2$ ./a.out 引数その1 引数その2
argc==3
argv[0]==./a.out
argv[1]==引数その1
argv[2]==引数その2
bash-3.2$
```

main()の引数 2

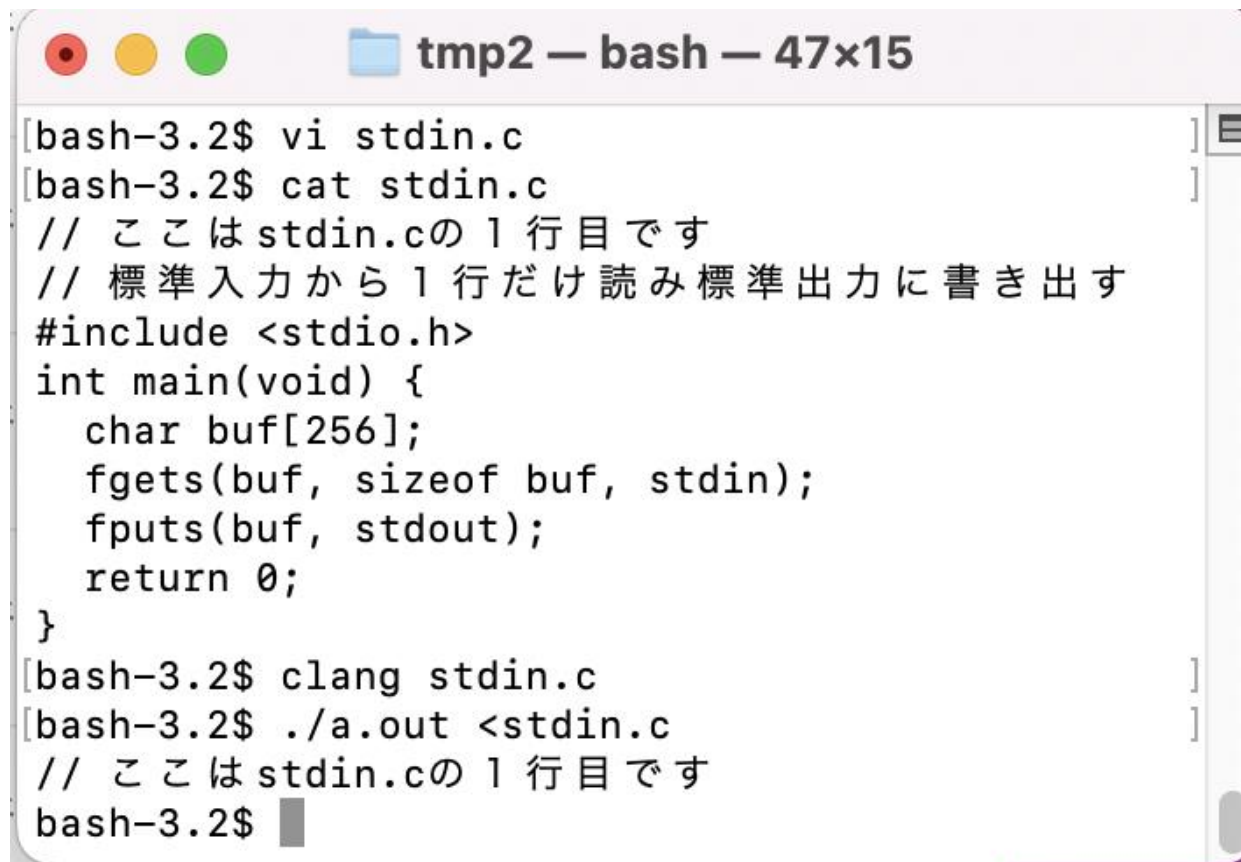
- argc には引数の数より 1 多い数が入る
- argv[0] にはその実行ファイルを起動するために呼び出した名前が入る
- argv[1], argv[2] ..., argv[argc-1]には実行ファイルに渡された引数が文字列として入る
- 統合開発環境でも argv を与えることはできる
- 最終課題での入力は大部分が argv[i] により与えられる

標準入出力

- printfは出力先を変えられないが、fprintfはprintfと同じ機能を提供し第一引数にファイルポインタ(fopenで返されるポインタ)を与えて出力先を変えられる
- scanfは入力元を変えられないが、fscanfはscanfと同じ機能を提供し第一引数にファイルポインタ(fopenで返されるポインタ)を与えて入力元を変えられる
- 実は stdin, stdout, stderr というファイルポインタが常に存在し、printfの出力先はstdout、scanf の入力元はstdin に固定されている。

標準入出力の変更

- コマンドプロンプトやターミナルでstdin, stdout, stderrをファイルに変更することができる（以下例）



```
tmp2 — bash — 47x15
[bash-3.2$ vi stdin.c
[bash-3.2$ cat stdin.c
// ここはstdin.cの1行目です
// 標準入力から1行だけ読み標準出力に書き出す
#include <stdio.h>
int main(void) {
    char buf[256];
    fgets(buf, sizeof buf, stdin);
    fputs(buf, stdout);
    return 0;
}
[bash-3.2$ clang stdin.c
[bash-3.2$ ./a.out <stdin.c
// ここはstdin.cの1行目です
bash-3.2$
```


標準入出力のリダイレクト

- 標準入出力をデフォルトのキーボードや画面以外に変更することを「リダイレクト」と呼ぶ
- これは自動化に大変便利である。本科目における、ウェブ上の検査システムや、自動化された採点システムは、`stdin`, `stdout`, `stderr` を適切にリダイレクトすることで実現されている
- `stdin` のリダイレクトは「`<in-file`」、`stdout` は「`>out-file`」、`stderr` は「`2>error-file`」で行う。
- ターミナルの進んだ使い方はT2Schola 1 回目の参考資料内にあり

リダイレクトやargv[i]の便利さ

- それらがないと人間の入力を自動化できなくて、例えばみなさんの提出物の評価時に、いちいち採点者がキーボードから入力するのは時間の無駄である
- 特別課題研究で行う研究のためのプログラム作成でも、計算機に張り付いてキーボードから入力しないと研究が進まないといつらい。自動化できると夜のうちに計算させて翌朝結果を見られる
- 計算機はやりたくない作業を計算機にやらせて人間を幸せにするのが目的の一つなので、自動化できるところは自動化して欲しい

サニタイザ

- 配列の範囲外アクセスや、整数演算のオーバーフロー・アンダーフローはコンパイルして実行ファイルを作る際にコンパイラは教えてくれない（ことが多い）
- 上記のような問題を実行時に検出する仕組みが「サニタイザ」で、コンパイル時に “-fsanitize=undefined,address” をCコンパイラにオプションとして渡すと使える(gcc ならさらに -fstack-protector-all を渡す)
- Check1.shと採点システムはサニタイザで提出物の間違いを探している。具体的な用法はcheck1.shをテキストエディット等で読むとわかる
- Xcode、Visual Studio 2022でサニタイザを使う手順はT2Schola第1回目「参考資料」フォルダ内にあり
- ウィンドウズ版Eclipseと共にインストールされるgccはサニタイザ使えません…

C言語と他の言語の比較

- C言語は配列の範囲外アクセスを咎めないし動的に割り当てたメモリを解放することもプログラマーが明示的に指示する必要がある
- その他のほとんどの計算機言語では範囲外アクセスはエラーとして検出され、動的割り当てメモリの解放は自動である。整数は無限の桁数を持ち桁あふれがない（Python等）。
- scanfへの間違った型指定のように、変数や配列に割り当てられていないメモリ領域の破壊はC言語以外ではほぼ不可能である
- C言語はCPUでできることをすべて表現できることを目指して設計されており、何の制限も安全装置も無い。そのためプログラミングの誤りが例えばPython, Javaなどに比べて起きやすい

C言語をカリキュラムで扱う理由

- CPUや装置を直接扱うプログラムの製作にはC言語が最も適している
- 研究室や将来の職場で、CPUや装置を直接扱うプログラムに従事する学生が多いから
- なお、C言語には安全装置がないため、基本的には「自分が何をやっているか」理解している人のための計算機言語であり、ウィンドウズや統合開発環境の「**中身を隠し、わかってない人も作業できるようにする**」設計思想の真逆にある

今回課題への注意

- MacのターミナルまたはWindowsパワースhellで課題に取り組むことを想定しているが、（やり方がわかるなら）統合開発環境を使用しても構わない

Ex 番号無し

- 適当なCのプログラム（例えば下記）を作って統合開発環境を用いずにコンパイルし実行する。これを**講義時間中になるべく終わらせ**、終わらない場合は相談する（友人と相談するのももちろんよい）。**提出物無し**

```
#include <stdio.h>
```

```
int main(void) { printf("Hello¥n"); return 0; }
```

なお、上記をコピペしても二重引用符などが半角ではないからうまくいかないので注意

Ex 12-1 (argv[i]と最適化を使う演習)

1. Ex 2-2の小問4で作成したex2_2_4.c を改造する。
 - ソートする配列が実行開始時刻に依存しないようにする。
 - ソートする配列の要素数をmain()関数の引数argv[1]から得る。
 - ソートする配列の要素数が一千万以上でも動作するようにする。
2. 最大限の最適化(-O3)と最適化禁止(-O0)のコンパイルオプションそれぞれについて、オプションを付けて作成した実行可能ファイルを、引数argv[1]として一千万以上を指定して実行し、実行時間を何等かの方法で測定する。（具体例は後述）
3. それぞれの測定した時間が表示されているスクリーンキャプチャを貼りこんだPDFを提出せよ。

提出物 ex12_1.pdf

Ex 12-1つづき

- 次ページにあるように、最適化無しと最大最適化を施した実行可能ファイルが、同一のソート結果・比較回数・入れ替え回数を表示するようにすること。
- ex2_2_4.cへのT2Schola上フィードバックで動作異常の指摘があっても、配列を正しくソートできていれば修正しなくてよい。
- Macのターミナル以外を用いたスクリーンキャプチャを用いる場合は、PDF中に使用したコンパイラを明記すること。
- MacOSターミナル内で `/usr/bin/time -p` を用いることで、起動したプログラムの各種処理時間（単位は秒）を表示できる。ソートの処理時間は「user」の部分に現れる。

Ex 12-1誤答例（要素数少なすぎ, 比較・交換回数が要素数に対して多すぎ）

```

ubuntu-22.04 $ clang -O0 -Wall ex12_1_bubble_sort.c
ubuntu-22.04 $ /usr/bin/time -p ./a.out 40000
p[0]=0.017332, p[20000]=4962.649902, p[39999]=9999.935547
p[0]=0.284174, p[20000]=5002.801270, p[39999]=9999.991211
p[0]=0.162465, p[20000]=5029.536621, p[39999]=9999.538086
p[0]=0.290829, p[20000]=4989.457520, p[39999]=9999.781250
p[0]=0.220481, p[20000]=4927.680176, p[39999]=9999.687500
Average number of comparisons= 799980000.0, Average number of swaps= 399847894.8
real 23.30
user 23.29
sys 0.00
ubuntu-22.04 $ clang -O3 -Wall ex12_1_bubble_sort.c
ubuntu-22.04 $ /usr/bin/time -p ./a.out 40000
p[0]=0.017332, p[20000]=4962.649902, p[39999]=9999.935547
p[0]=0.284174, p[20000]=5002.801270, p[39999]=9999.991211
p[0]=0.162465, p[20000]=5029.536621, p[39999]=9999.538086
p[0]=0.290829, p[20000]=4989.457520, p[39999]=9999.781250
p[0]=0.220481, p[20000]=4927.680176, p[39999]=9999.687500
Average number of comparisons= 799980000.0, Average number of swaps= 399847894.8
real 10.18
user 10.17
sys 0.00
ubuntu-22.04 $
```

最適化により動作が変わる？

- コンパイルオプションの違い（最適化無しと最大最適化など）により得られる実行可能ファイルの動作が異なることがある
- この原因はプログラム内の未定義動作部分の動作が、最適化の度合いによって変わるためであることが多い（珍妙な例がブログ等で数多く紹介されていて興味深い（次ページ））
- 課題12-1において最大最適化を行ったときにソート結果が最適化無しと異なる場合、コンパイルオプションに依存せず正しくソートできるようにex2_2_4.cを修正したのちに、最適化無しと最大最適化の処理時間を測定すること
- 未定義動作部分の発見にはex2_2_4.cへのT2Schola上フィードバックならびに~matsumoto.r.aa/bin/check-final.shを使用できる
- 最終課題は最適化しないと耐え難く動作が遅くなると思われる

最適化により動作が変わる奇妙な例

- 「最適化バグですか？ いいえ未定義の挙動です。」
<https://qiita.com/yoh2/items/489d8749e196e9092349>
- 「C言語で未初期化のローカル変数にはゴミ値が入ってるとは限らない」
<https://qiita.com/fujitanozomu/items/cc0d578114ee5b825b43>
- 「// なぜかこの行がないと動かない」
<https://qiita.com/BlueRayi/items/996e09ac50880d8c9f53>
- 以上のように最適化を掛けたらまともに動作しないことがC言語およびC++ではよくあるが、ここ数年はサニタイザの出現により随分デバッグが簡単になった（以前はあらゆる行にprintfを挿入して未定義動作を探した）

Ex 12-2 (分割コンパイルの演習)

- Ex3-3 で作成した ex3_3_2.c を, 3つのファイル ex12_2_1.c ex12_2_2.c ex12_2.hに分割せよ。
- ex12_2_1.c には, main()関数と #include文のみを入れ、typedefや complex_number cmp_add(complex_number a, complex_number b); のような関数宣言を含めないこと。
- ex12_2_2.c には, main()関数以外の関数と #include文のみを入れること。
- typedef、関数宣言、構造体定義などはすべてex12_2.hにまとめ, .cファイルからこのヘッダファイルを #include して使うこと (自作.hファイルの#includeの仕方を思い出して下さい)
- コマンドプロンプト (Macならターミナル) において, 「clang ex12_2_1.c ex12_2_2.c -lm」で実行可能ファイルが作られることを確認することが望ましい
- 提出物: ex12_2_1.c ex12_2_2.c ex12_2.h

Ex 12-3 (プリプロセスを体験する)

- Cコンパイラ(Macのclang等)に「-E」を与えるとCソースファイルをプリプロセッサが前処理した結果を標準出力に表示する
- 次ページのプログラム（文字が全角でコピペで動作しない可能性あり）について、自分のCコンパイラの**プリプロセッサの前処理**により、stdin およびNULLがどういう値に置き換えられているか調べPDFに記入せよ。前処理された結果が表示されている**スクリーンキャプチャもPDFに含めること**。前処理されたCソースは、標準出力を適当な名前のファイルにリダイレクトして、テキストエディット・メモ帳などで閲覧するとよい。**計算機室以外で課題を解いた場合には、使用しているCコンパイラの名前とバージョン番号も書くこと。** 提出物 [ex12_3.pdf](#)

この課題の狙いは統合開発環境では理解できないプリプロセスを理解することにある

Ex 12-3続き

```
#include <stdio.h>
#define BUFFER_SIZE 256
Int main(void)
{
    char buffer[BUFFER_SIZE];
    fscanf(stdin, "%s", buffer);
    fprintf(stdout, "to stdout: NULL is %p¥n", NULL);
    fprintf(stderr, "to stderr: %s¥n", buffer);
    return 0;
}
```

上記のファイルはT2Scholaにあり

Check1.shの引数対応

- `~matsumoto.r.aa/bin/check1.sh` は引数`argv[1]`等を与えられる課題プログラムに対応していない
- `~matsumoto.r.aa/bin/check-final.sh` プログラム.c 引数1 引数2 ... のようにターミナルから実行すると、プログラム.cがコンパイルされて得られたa.outに引数1 引数2 ... が与えられて`check1.sh`と同様の検査が行われる
- 未定義動作を排除して高得点を目指す人は活用して欲しい

提出ファイル名一覧

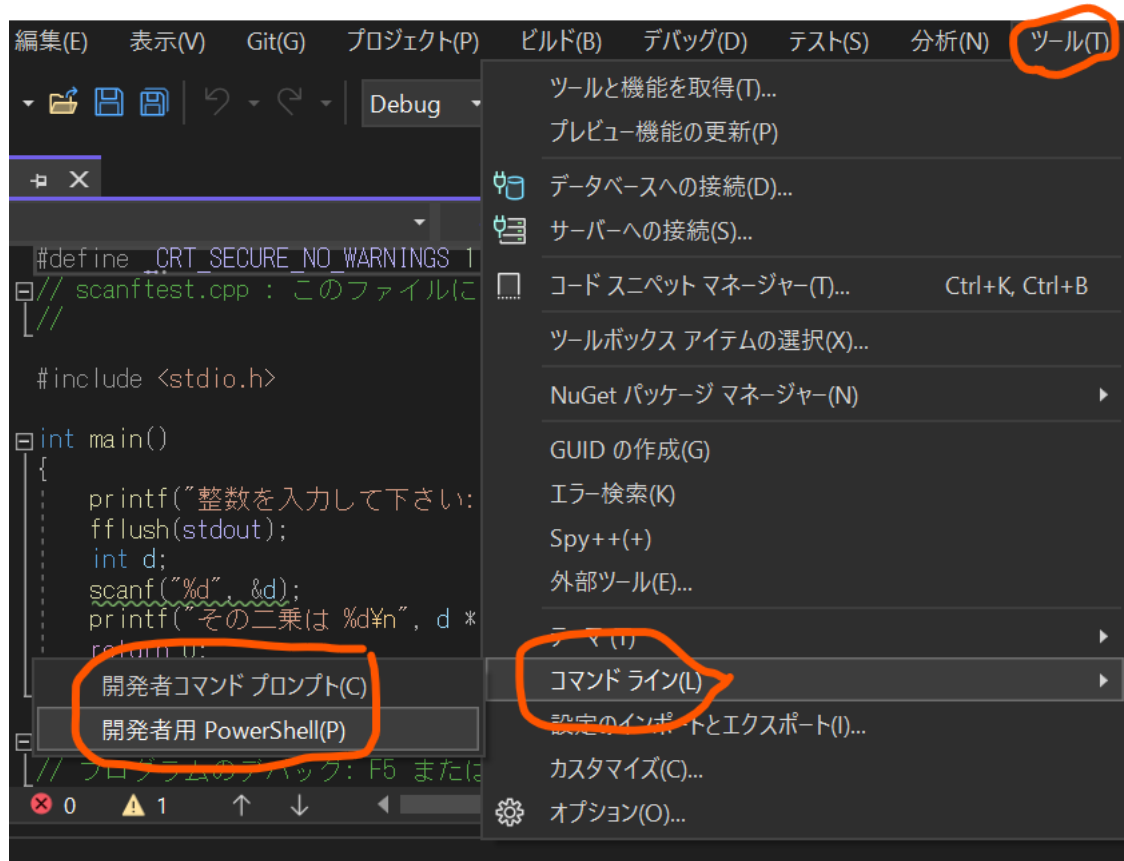
- ex12_1.pdf
- ex12_2_1.c ex12_2_2.c ex12_2.h
- ex12_3.pdf

事情により自宅学習する場合のサポートページ

- この回は計算機室に来て実習することを強く推奨しますが、これ以降はこられない人のため内容になります。計算機室にいる人は見なくて構いません。
- 自宅でMacを使っている人は今までの内容がほぼそのまま使えるので自宅Macで実習して下さい
- Linuxを使っている人には何も支援は要らないはずです
- これ以降はWindowsを自宅でする人向けの話になります。まずT2Schola第1回の「参考資料（読まなくてよい）」のフォルダにある指示書に従いVisual Studio 2022(以降VS2022)を用いC言語のプログラムの実行をできるようにして下さい
- VS2022はVS Codeとは異なります

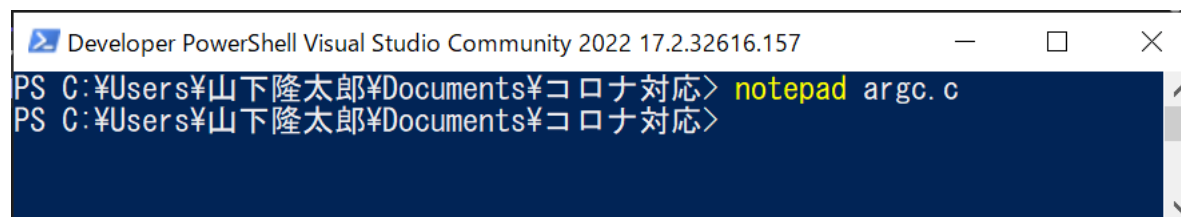
ターミナルに相当するものの起動

- VS2022の適当なプロジェクトから以下のようにパワーシェルを起動

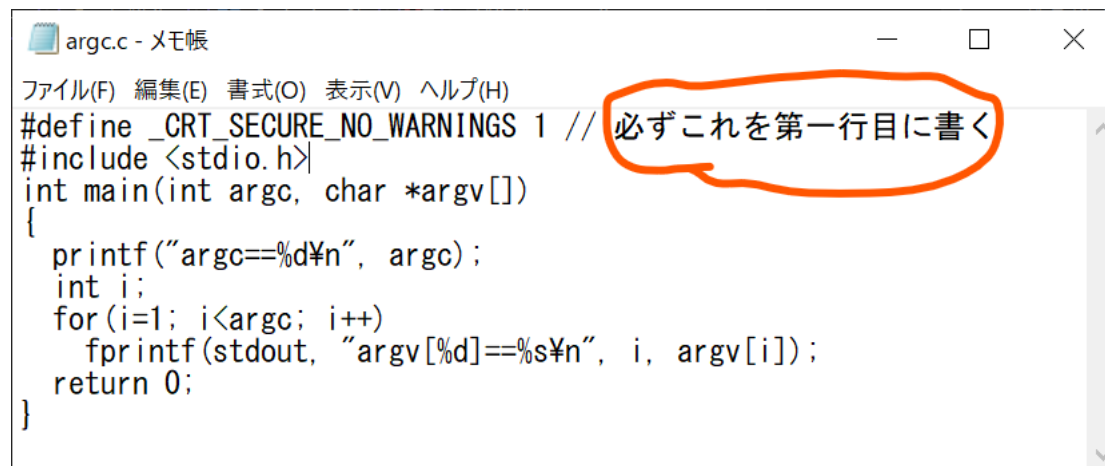


テキストエディットに相当するものの起動

- パワーシェル内から「notepad ナントカ.c」と入力し「メモ帳」を起動



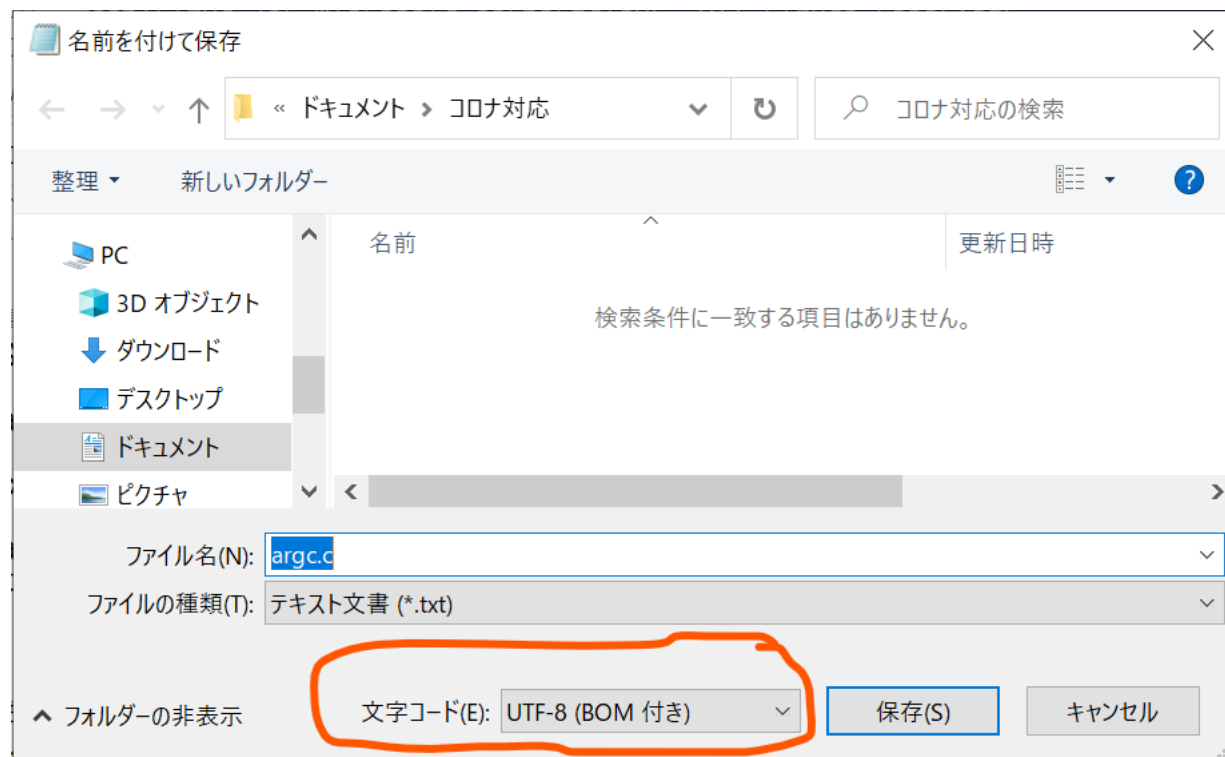
```
Developer PowerShell Visual Studio Community 2022 17.2.32616.157
PS C:\Users\山下隆太郎\Documents\コロナ対応> notepad argc.c
PS C:\Users\山下隆太郎\Documents\コロナ対応>
```



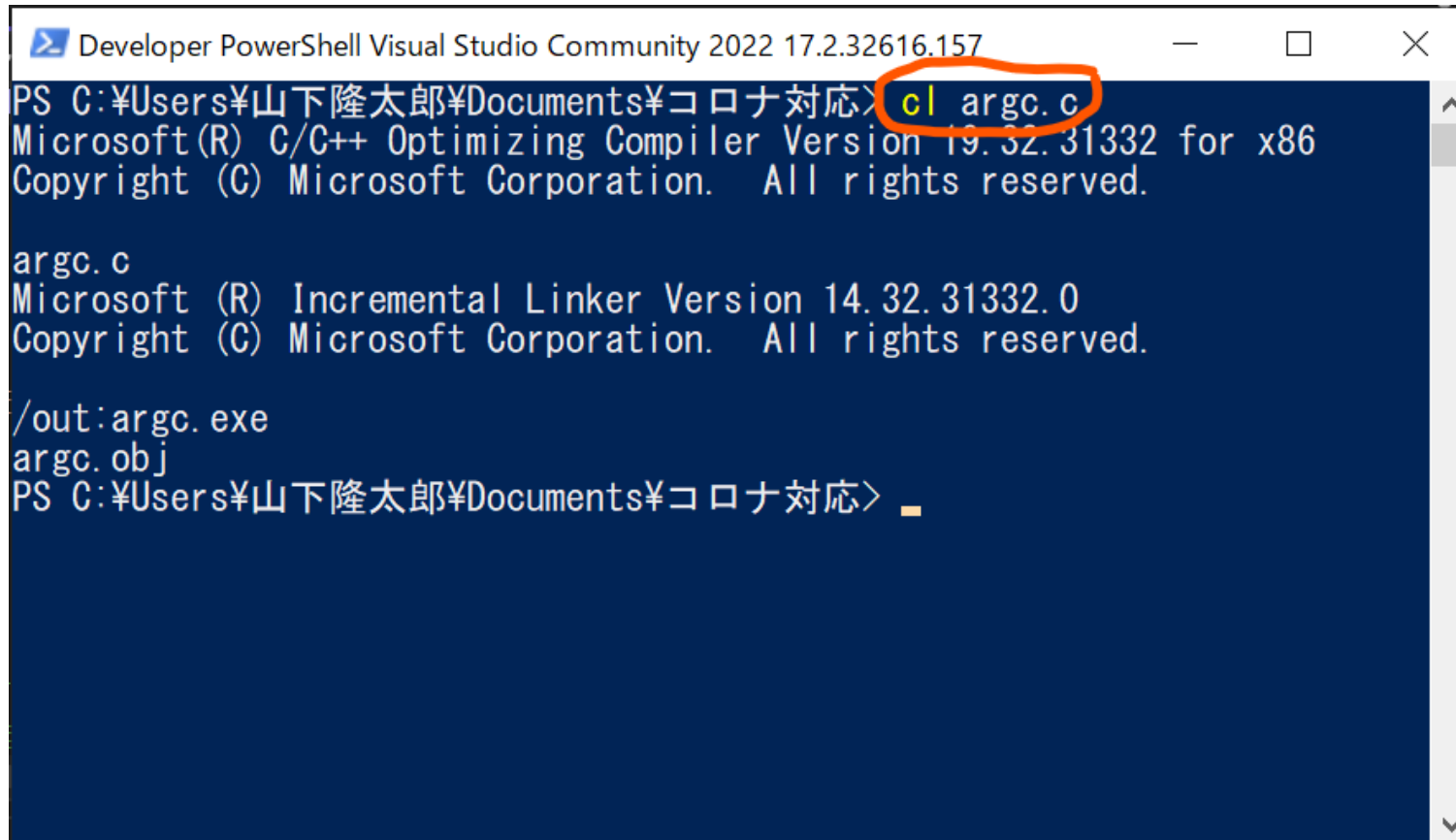
```
argc.c - メモ帳
ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)
#define _CRT_SECURE_NO_WARNINGS 1 // 必ずこれを第一行目に書く
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("argc==%d\n", argc);
    int i;
    for(i=1; i<argc; i++)
        fprintf(stdout, "argv[%d]==%s\n", i, argv[i]);
    return 0;
}
```

メモ帳保存形式はBOM付きUTF-8にする

- そうしないとVS2022から警告が出てくる



clを用いてコンパイルする



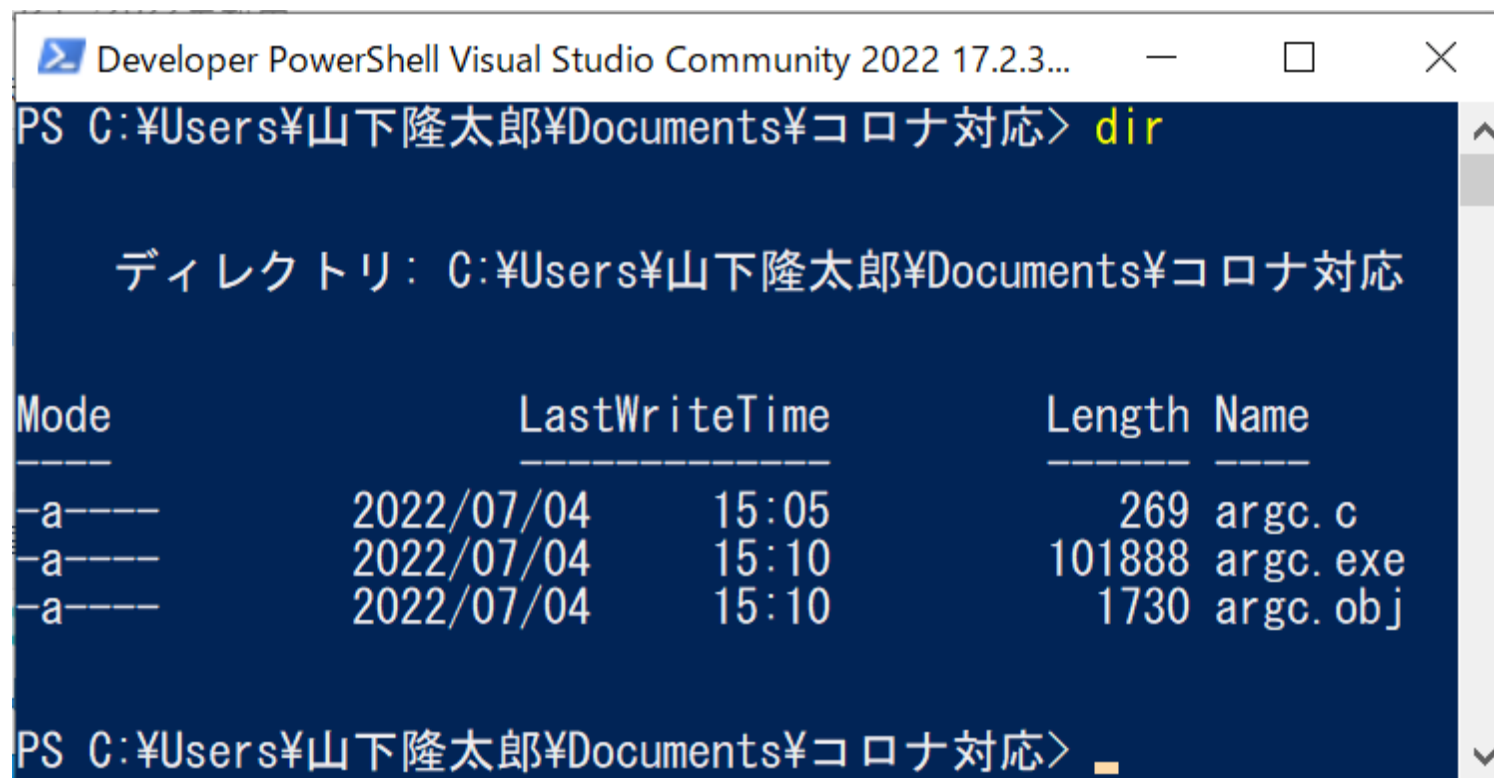
```
Developer PowerShell Visual Studio Community 2022 17.2.32616.157
PS C:\Users\山下隆太郎\Documents\コロナ対応> cl argc.c
Microsoft(R) C/C++ Optimizing Compiler Version 19.32.31332 for x86
Copyright (C) Microsoft Corporation. All rights reserved.

argc.c
Microsoft (R) Incremental Linker Version 14.32.31332.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:argc.exe
argc.obj
PS C:\Users\山下隆太郎\Documents\コロナ対応>
```

コンパイルして出来るファイルは…

- argc.objは「オブジェクトファイル」でありargc.exeは「実行可能ファイル」である



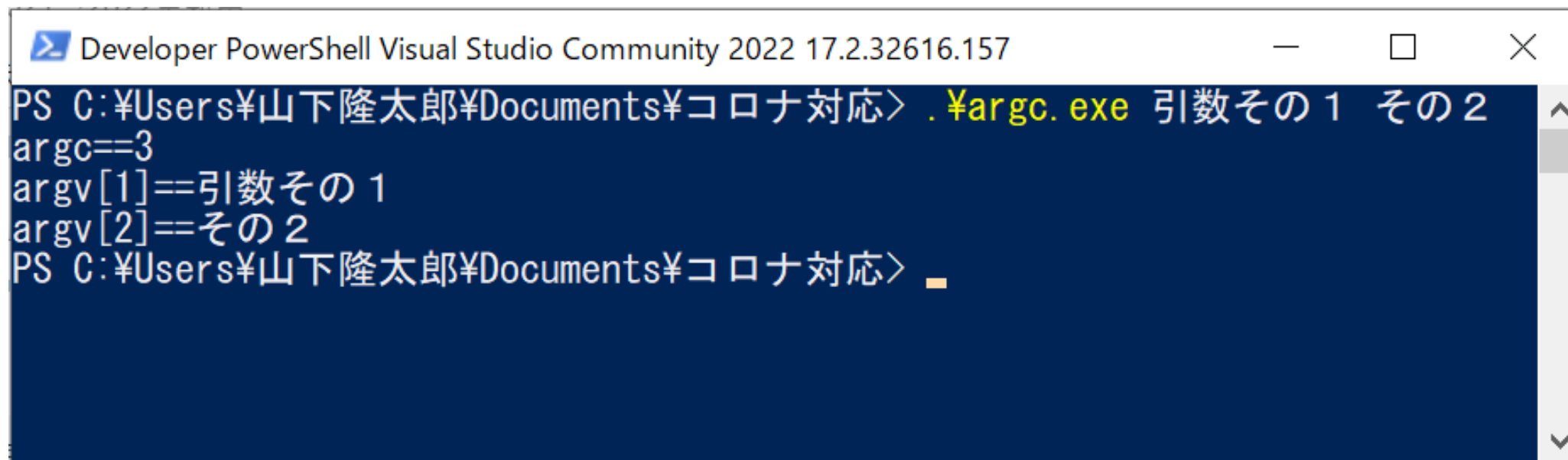
```
Developer PowerShell Visual Studio Community 2022 17.2.3...
PS C:\Users\山下隆太郎\Documents\コロナ対応> dir

ディレクトリ: C:\Users\山下隆太郎\Documents\コロナ対応

Mode                LastWriteTime         Length Name
----                -
-a----          2022/07/04    15:05         269 argc.c
-a----          2022/07/04    15:10       101888 argc.exe
-a----          2022/07/04    15:10        1730 argc.obj

PS C:\Users\山下隆太郎\Documents\コロナ対応>
```

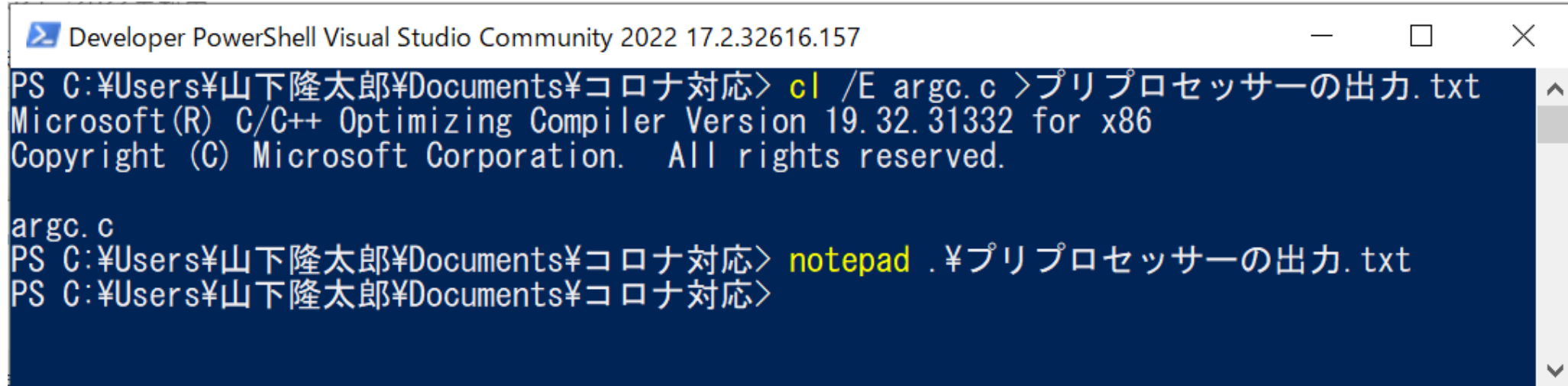
実行する



```
Developer PowerShell Visual Studio Community 2022 17.2.32616.157
PS C:\Users\山下隆太郎\Documents\コロナ対応> .\argc.exe 引数その 1 その 2
argc==3
argv[1]==引数その 1
argv[2]==その 2
PS C:\Users\山下隆太郎\Documents\コロナ対応> █
```

プリプロセッサ出力は-Eでなく/Eで確認

- 課題12-3で必要。出力ファイル名を.txtで終わらせる



```
Developer PowerShell Visual Studio Community 2022 17.2.32616.157
PS C:\Users\山下隆太郎\Documents\コロナ対応> cl /E argc.c >プリプロセッサの出力.txt
Microsoft(R) C/C++ Optimizing Compiler Version 19.32.31332 for x86
Copyright (C) Microsoft Corporation. All rights reserved.

argc.c
PS C:\Users\山下隆太郎\Documents\コロナ対応> notepad .\プリプロセッサの出力.txt
PS C:\Users\山下隆太郎\Documents\コロナ対応>
```


Cコンパイラ(cl.exe)の引数

パワースhellからCコンパイラを直接起動する場合、ファイル名（myprogram.cなど）の前に空白文字で区切ってオプション引数を与えることで動作を変えられる。代表的なものを紹介する（パワポがハイフンなどを全角文字に変換しているから以下をコピーしても動かない）

- 最適化：「/O2」は最適化を行い実行ファイルの動作を速くし、「/Od」は最適化を禁止しデバッグしやすくする
- 警告：「/Wall」を付与すると怪しい部分を教えてくれる
- サニタイザ：「/fsanitize=address /Ge /GZ」は配列範囲外アクセスなどの実行時検査をする
- 「/help」で他の色々なオプション引数が表示される
- Visual Studio 2022で上記のことをやる手順は、第1回の「参考資料（読まなくてよい）」内の開発環境便利な使い方集で紹介されている

Cコンパイラ(clang-cl.exe)

cl.exeは第5回で説明された可変長引数を使えない。可変長引数を使いたい場合は第1回の「参考資料」で説明される手順に従いclang-clをインストールする。そうするとclang-cl.exe (MacOSのclangとほぼ同じ) が使えるようになる