

プログラミング演習2

期末レポート

氏名: 今田将也 (IMADA, Masaya)
学生番号: 09430509

出題日: 2019 年 04 月 10 日
提出日: 2019 年 xx 月 xx 日
締切日: 2019 年 07 月 29 日

1 概要

プログラミング演習2においては、プログラミング演習1でC言語の実践的なプログラミングの演習を行った際に不足していた機能を追加した。ファイルからのcsv形式のデータの読み込み、並びに書き出し。指定語句でデータを検索する機能と、メモリ中のデータを並び替える機能である。また、完成したプログラムの結果を通して、さらなる不足機能の考察、既存コマンドの改良案と実装方法についての考察を行った。

なお、与えられたプログラムの基本仕様と要件、および、本レポートにおける実装の概要を以下に述べる。プログラムの使用方法についても記載した。

1. 仕様

- (a) 標準入力からID, 学校名, 設立日, 住所, 備考からなるコンマ区切り形式 (CSV 形式) の名簿データを受け付けて、それらをメモリ中に登録する機能を持つ。CSV 形式の例を以下に示す。

```
5100046,The Bridge,1845-11-2,14 Seafield Road Longman Inverness,SEN Unit 2.0 Open  
5100224,Canisbay Primary School,1928-7-5,Canisbay Wick,01955 611337 Primary 56 3.5 Open  
:
```

- (b) ただし%で始まるコマンドを受け付けて、登録してあるデータを表示したり整列したりするなどの機能を持つ。実装するコマンドを表1に示す。

2. 要件

- (a) 名簿データは配列などを用いて少なくとも10000件のデータを登録できるようにする。今回のプログラムでは、構造体 `struct profile` の配列 `profile_data_store[10000]` を宣言して、10000件のデータを格納できるようにする。
- (b) 名簿データは構造体 `struct profile` および構造体 `struct date` を利用して、構造を持ったデータとしてプログラム中に定義して利用する。実装すべきデータ構造は表2である。表中の n bytes とは、 n バイトの `char` 型配列を意味する。

また、本レポートでは以下の考察課題について考察をおこなった。

表 1: 実装するコマンド

コマンド	解説	パラメータ範囲
%C	メモリ中のデータ件数を表示する	パラメータなし
%P	メモリ中データを, n に応じて表示させる	n :-10000~10000 (0:全件表示 $n>0$:前から指定件数正順表示 $n<0$:後ろから指定件数正順表示)
%Q	システムを終了する	パラメータなし
%R filename	filename ファイルから csv データを読みこむ	filename
%W filename	メモリ中のデータを filename ファイルに書き出す	filename
%F word	システムを終了する	word
%S n	システムを終了する	n :1 から 5 までの正整数

表 2: 名簿データ

ID	学校名	設立日	住所	備考
32bit 整数	70 bytes	struct date	70bytes	任意長

1. 不足機能に関する考察
2. エラー処理に関する考察
3. 新規コマンドの実装
4. 既存コマンドの改良

また、発展的な考察として、以下の内容についても考察を行った。

1. 構造体のサイズ
2. 本課題の要件に対する考察
3. コマンドの拡張

2 プログラムの作成方針

プログラムをおおよそ以下の部分から構成することにした。それぞれについて作成方針を立てる。

1. 必要なデータ構造の宣言部 (2.1 節)
2. 標準入力から得た CSV データの解析部 (2.2 節)
3. 構文解析したデータの内部形式への変換部 (2.3 節)
4. 各種コマンド実現部 (2.4 節)

2.1 宣言部

“宣言部” は必要な構造体を宣言する部分である。このレポートでは概要で示した表 2 に基づいて、以下のように宣言する。

```

struct date {
    int y;
    int m;
    int d;
};

struct profile {
    int id;
    char name[70];
    struct date found;
    char add[70];
    char *others;
};

struct profile profile_data_store[10000];

int profile_data_nitems = 0;

```

ここでは、名簿管理に必要なデータを定義している。struct date においては、設立日の設定に必要な変数 y, m, d を定義した。順に、設立年、設立月、設立日を表している。struct profile では、一つ当たりデータの構造を作るために利用している。int id は ID, char name は学校名, struct date found は設立日, char add は住所, char others は備考を設定している。これにより仕様に必要なデータを格納することが可能になっている。

2.2 解析部

“解析部”は入力された文字列を判別し処理をおこなう箇所である。しかし、このままでは、仕様を実現するための方法が曖昧であるうえフローチャートも複雑になる懸念があるうえ、今回の仕様の実現には手間が多くかかりそうである。そこで、段階的詳細化の考え方に基づいてさらなる詳細化をおこなって、プロトタイプを作りながらボトムアップによる実装をすることにした。まず、下記の (a) から (e) のように分割することにする。

- (a) 標準入力から読むべき行が残っている間、文字の配列 char line[] に 1 行分を読み込む。
- (b) line の 1 文字目が, ' % ' ならば, 2 文字目をコマンド名, 3 文字目以降をその引数として, 決定されたコマンドを実行する。
- (c) さもなくば line を新規データとみなし, , ' ' を区切りとして 5 つの文字列に分割する。
- (d) 分割してできた 5 つの文字列を変換部に渡し構造体に代入する。
- (e) 次の行を読み込む

コマンドを入力させるか、新規データを入力させるか選択したのちに、以上の処理をさせるように一段階詳細化させることも考慮したが、名簿管理プログラムということが自明であるため、プログラム起動時にコマンドかそうでないかを判別して処理させることで実装した。ここで扱う文字列は最大数が 1024 に限定されているため入力文字数に注意する必要がある。

2.3 変換部

“変換部”は分割された CSV データもしくは新規入力データを項目毎に型変換し、対応する構造体メンバに代入する部分である。メンバとして様々な型を用いているため、適切な代入の使い分けが必要となる。

文字列は関数 `strcpy` を用いて代入する。数値の場合、関数 `strtol` を用いて文字列を数値に変換してから代入する。構造体 `struct date` であるメンバ `y`, `m`, `d` については `split` 関数を実行し、文字列を分割してから代入数値としてする。

なお、構造体への代入については、`strcpy` 関数を用いることで容易に実装することができる。例えば、"2014-10-25"のような文字列を `split` 関数により分割し、`strcpy` 関数によって入力されたデータを `struct profile` 内の `struct date` に年と月と日を格納するという処理は、入力された文字列を','により分割する処理と同じ処理である。従って、区切り文字がCSVの','とは異なり、区切り文字が-になること以外は同様に記述できるはずである。

また、解析部から与えられた文字列はメモリ内に保持されているものではないデータであることにも注意する必要がある。つまり、変換部で文字列を処理する際には、入力された文字列に対して変換を行い、結果を表示するだけではなく、関数 `new_profile` を使って受け取ったデータをメモリ内に保持しておく作業を行わなければならないことに気をつける必要がある。

2.4 各種コマンド実現部

“各種コマンド実現部”は、表1にある実装コマンドの、実際の処理をおこなう部分である。このレポートでは、具体的には、登録されているデータ件数を表示する機能と、指定形式でデータ内容を表示する機能、外部ファイルからデータを読み込む機能、外部ファイルに書き出す機能、メモリ中のデータを並び替える機能、データを検索する機能、また、システムを終了させるための機能の7つを実装している。

登録されているデータ件数を表示するためには(%C)、グローバル変数にて宣言している `profile_data_nitems` の値を表示すればよい。グローバル変数で宣言したのにも理由があり、`main` 関数内でこの変数を宣言してしまうと、別関数で利用する際に値の受け渡しが発生し、手間が増えるためグローバル変数として宣言した。

登録されているデータを表示するには(%P n)は `printf` 関数でメモリ内のデータを各項目毎に表示すればよい。ただし、与えられた引数が負の場合は、逆順ではなくデータを後ろから正順で表示するため、ポインタの位置に注意する必要がある。また、データ件数が0件の場合でも上記コマンドは実行されるが、データがないという表示を行わせている。

外部ファイルからのデータの入力(%R)はファイル構造体のポインタを作成し、`fopen()` 関数を用いた。データの書き出し(%W)も同様に実装したが、オープンモードが異なることに注意しなければならない。なお、書き出しの際は読み込んだファイルのCSVデータと同様の形式で書き出す仕様である。

データ検索(%F)は、引数として入力された語句がそれぞれのデータの要素に完全一致しているかどうかで実装した。設立日については、年月日を分割して保存したため再度文字列に変換してから文字列比較を行った。

データ整列(%S)は整列の条件式でデータの大小を比較してバブルソートにて実装していたがクイックソートへ変更した。変更した理由については後ほど述べる。比較の際、文字列データについては2つの文字列の大小を比較できる `strcmp()` 関数を用いた。設立日は年月日ごとに比較を行わせた。数値データは単純に減算し大小比較をしている。

3 プログラムおよびその説明

プログラムリストは9節に添付している。プログラムは全部で===行からなる。以下では、前節の作成方針における分類に基づいて、プログラムの主な構造について説明する。

3.1 汎用的な関数の宣言（===行目から===行目）

まず、汎用的な文字列操作関数として、`subst()` 関数を===から===行目で宣言し、`split()` 関数を===から===行目で宣言、さらに `get_line()` 関数を===から===行目で宣言している。

`subst` は、引数の `str` が指す文字列中の `c1` 文字を `c2` に置き換える。プログラム中では、入力文字列中の末尾に付く改行文字をヌル文字で置き換えるために使用している。

`split` は 引数の `str` が指す文字列を区切文字 `c` で分割し、分割した各々の文字列を指す複数のポインタからなる配列を返す関数である。プログラム中では、CSV を', 'で分割し、分割後の各文字列を返すのに使用されている。また、“2004-05-10” のような日付を表す文字列を '-' で分割して、`struct date` を生成する際にも使用している。

`get_line()` は、標準入力からの入力を受け付ける処理を当初実装していたが、ファイルポインタからの入力に対応ができていなかった。そのため、それに対応した `get_line_fp()` 関数をファサードした。 `get_line_fp()` 関数はファイルポインタからの読み込みを行う関数で、`get_line()` 関数ではその引数として `stdin` を渡して標準入力からも受け取れるようにした。

構造体のデータを一件出力するための関数として `printdata()` を、構造体を入れ替える関数として `swap_struct()` 関数を宣言した。

3.2 変換部（===行目から===行目）

～行目は `struct date` 型の宣言部である。メンバについては、変数 `y` は設立年、変数 `m` は設立月、変数 `d` は設立日にそれぞれ対応させている。

～行目は `struct profile` 型の宣言部、～行目はそれを扱う関数 `new_profile` である。メンバについては、設立日を入れ子構造にしている。こうすることで、要素を管理しやすくなる。なお、備考に対応する文字列 `*others` は任意長を許すようにしているため、`malloc` 関数と `strlen` 関数を用いて文字列を動的に格納できるようにした。文字列から各データ型への変換を担う関数は、`struct new_profile` とすることで、変換部であることを明確にした。具体的な処理内容としては、受け取った文字列 `str` を分割し、分割した文字列を `ret1[]` に格納し、その後要素ごとに対応する構造体メンバにエラー検出のある `strncpy` 関数を用いて格納している。設立日については、`ret2[]` を用意し、各メンバに対応するよう格納させている。

3.3 各種コマンド実現部（aaa 行目から aaa 行目）

a 行目からの各種コマンド実現に必要な関数群は、`cmd_処理名` という名前に統一することで、関数であることを明確にした。コマンド `%P` は `cmd_print()`、コマンド `%C` は `cmd_check()`、コマンド `%Q` は `cmd_quit()`、コマンド `%R` は `cmd_read()`、コマンド `%W` は `cmd_write()`、コマンド `%F` は `cmd_find()`、コマンド `%S` は `cmd_sort()` にそれぞれ対応している。

124-144 行目は、`%P,%C,%Q,%R,%W,%F,%S` のコマンドを解釈して適切な関数を呼び出す部分である。

`%P` に対応する関数 `cmd_print()` の処理内容としては、a から a 行目に記載してある。内容は、表 1 に記載した。

`%C`、`%Q` はそれぞれ、a 行目からと a 行目からに処理内容を記述した。`%R,%W` はそれぞれ、a 行目からと a 行目からに処理内容を記述した。`%F,%S` はそれぞれ、a 行目からと a 行目からに処理内容を記述した。

3.4 解析部 (aaa 行目から bbb 行目)

a から b 行目は main() 関数で、a から b 行目は、parse_line() 関数であり、作成方針で説明した解析部の動作におおよそ相当する。ただし (c) の 5 つの文字列に分割する部分は、解析部の main() 関数では実現せず、処理内容を明確にするために変換部である new_profile() 関数中で split を呼出し、各要素ごとに分割を行うことにしている。

4 プログラムの使用方法

本プログラムは名簿データを管理するためのプログラムである。CSV 形式のコンマ区切りのデータと % で始まるコマンドを標準入力から受け付け、処理結果を標準出力に出力する。入力形式の詳細については、第 1 節を参照されたい。

プログラムは、CentOS で動作を確認しているが、一般的な UNIX で動作することを意図している。gcc でコンパイルした後、標準入力から入力ファイルおよびデータを与える。

```
% gcc -Wall -o program1 program1.c
% ./program1 < test.txt
```

プログラムの出力結果としては CSV データの各項目を読みやすい形式で出力する。例えば、下記の test.txt に対して、

```
111,The Bridge,1845-11-2,Okayama,SEN Unit 2.0 Open
222,Bower School,1908-1-19,Kagawa,01955 641225 Primary 25 2.6 Open
333,Canisbay School,1928-7-5,Tokyo,01955 611337 Primary 56 3.5 Open
%C
%P 0
%Q
```

以下のような出力を得る。

```
param is 0.
*****print record data*****
data :      1 -----
Id    : 111
Name  : The Bridge
Birth : 1845-11-02
Addr  : Okayama
Com.  : SEN Unit 2.0 Open
-----
data :      2 -----
Id    : 222
Name  : Bower School
Birth : 1908-01-19
Addr  : Kagawa
Com.  : 01955 641225 Primary 25 2.6 Open
-----
data :      3 -----
Id    : 333
Name  : Canisbay School
Birth : 1928-07-05
Addr  : Tokyo
Com.  : 01955 611337 Primary 56 3.5 Open
-----
```

入力中の%C はこれまでの入力データの件数を表示することを示し，%P 0 は入力したデータのうち，全件のデータを表示することを示している．なお，%Q はシステムを終了することを示す．

5 作成過程における考察

第2節で述べた実装方針に基づいて，第3節ではその実装をおこなった．しかし，実装にあたっては実装方針の再検討が必要になる場合があった．本節では，名簿管理プログラムの作成過程において検討した内容，および，考察した内容について述べる．

5.1 関数 `split` についての考察

関数 `split` については方針通りに実装することができたが，容易に実装することはできなかった．当初はコンマまでの文字列を別の配列に保存することを繰り返して実装しようとしていたが，これではコンマの数で文字列を判断することになるため失敗した．そこで文字列を破壊的に分割し別途規定数用意した文字配列にアドレスを格納することで実装できた．文字列を丸ごとコピーすることも考えられたが，その方法は，入力した倍のメモリ量が必要な上に使わなくなったメモリを開放する手間が増えるため用いなかった．

5.2 関数 `get_line` についての考察

標準入力からの入力について当初は，`main` 関数の中で `while` 文繰り返し入力を行わせて，入力の度に入力内容が `NULL` でないか調べ関数 `subst` を適用する方法をとっていたが，`while` 文を脱する処理も記述しなければならないため手間が増えた．そこで，今回は入力内容に問題がなければ1を，あれば0を返す方針で実装を行った．これで，もし別の関数内で標準入力からの入力を行う際でも使いまわすことができ汎用性を持たせることができる．

5.3 関数 `new_profile` についての考察

関数 `new_profile()` の実装では，単に文字列を受け取り，その文字列を操作した後に，用意している配列 `profile_data_store` にコピーする方法も考えられたが，値を渡すことになり使用するメモリの量が増えると考えた．そのため，ポインタによるアドレス渡しによって実装を行った．また，配列を構造体配列として宣言しているので，ここでは構造体を返り値として設定した．そして，文字列を数値に変換する際にはエラー検出のある `strtol` 関数を用いた．

5.4 関数 `exec_command` についての考察

仕様を満たす実装はできた．標準入力からのデータは文字列であるため，各種コマンドへの引数を数値に変換する作業を行っている．この際，`atoi` 関数だとうまく変換されないことがあったため `strtol` 関数を用いている．また，定義されていないコマンドが入力された際は該当するコマンドがないという表示を出すようにした．

5.5 関数 `cmd_print` についての考察

まず、受け取った `param` の値が正か負か0を判断させなくてはならない。その後、正ならば指定件数分前から順に表示させ、0ならば全件表示を行わせて、負ならば `param` の値を一度正に戻し指定件数分ポインタを移動させた後に正のとき同様に表示処理を行わせる手順で実装を行った。表示させる部分については、`find` 関数にて利用するため `printdata` 関数を別途作成し、当該関数内で表示させた。

5.6 関数 `cmd_check` についての考察

この関数の実装はあらかじめ、登録件数を保存するための変数をグローバル変数にて宣言することで容易に実装することができた。誤ったデータが入力された際も当初は件数が増えてしまう実装であったため、増やさないように改変を行った。

5.7 関数 `cmd_quit` についての考察

この関数の実装は、`stdio.h` にある `exit` 関数を利用することにより実装を行った。

5.8 関数 `cmd_read` についての考察

外部ファイルからのデータ入力、`stdio.h` にある `fopen` 関数並びに `fclose` 関数を利用することで実装した。ファイルポインタの内部の仕組みを理解できてはいるが、作成中にファイル名が異なると開けずにシステムが強制終了するため回避する処理が必要だった。

5.9 関数 `cmd_write` についての考察

上記の `cmd_read` 同様に実装を行った。ただし、データを書き込むためオープンモードを書き込み状態にした。 `fprintf` を用いてファイルポインタとして開いたファイルにコンマ区切りで書き込むため、データ形式には気を付けた。1データの終わりに改行を置くことを当初忘れていたため望む実装ができず苦戦した。

5.10 関数 `cmd_find` についての考察

本関数は、各要素が `%F` の引数として与えられたデータを完全に一致しているかどうかで実装をしようとしたが、入力される引数が文字データであることを考慮しておらず苦悩した。構造体 `profile` の要素の `name`, `add`, `others` は文字であるから `strcmp` 関数にて比較可能で、`id` についても `strtol` 関数を用いた比較が可能である。しかし、`found` は年月日ごとに別の構造体に数値としてバラバラにあるため、一度 `csv` データと同じ `'-'` で繋がった形式に別途関数を介し変換させた後に別のデータ同様に文字列として比較を行った。

5.11 関数 `cmd_sort` についての考察

ソートについては苦労が大きく実装に手間取った。文字列の比較がわからなかったが `strcmp` 関数が適していると知りなんとか実装ができた。要素の大小の結果をソートの比較条件に利用した。交換の回数をカウントしバブルソートとクイックソートの比較を行った（第 6.4 節）結果、クイックソートを実装した。

6 結果に関する考察

演習課題のプログラムについて仕様と要件をいずれも満たしていることをプログラムの説明および使用法における実行結果例によって示した。ここでは、概要で挙げた以下の項目について考察を述べる。

1. 不足機能についての考察
2. エラー処理についての考察

6.1 不足機能についての考察

不足機能については、以下の内容が考えられる。

1. 入力後のデータ修正機能
2. 指定要素のみ表示させる機能

6.1.1 入力後のデータ修正機能

現在の機能では、データの形式さえあっていれば名簿データとして追加されるため、データの内容を間違えて入力しても追加される。これを回避するために、6.3.1 節にて指定データを削除する機能を実装している。しかし、削除に再度長いデータを入力する必要があり不便極まりない。

名前を間違えたら名前のみを、住所を再度編集したい場合は住所のみを書き換えるという機能があればより現実的に利用ができる名簿管理プログラムになるのではないだろうか。

6.1.2 指定要素のみ表示させる機能

これは `cmd_print` を改良すれば実装ができそうである。`cmd_print` はすべての要素を表示させている。そこに一つ引数を増やすかもしくは別のコマンドを作成し、表示させたい要素と数値を紐付けする。例えば 1 ならば ID のみを %P のように表示させる実装が可能そうだ。この機能は 6.3.3 節にて実装を行ったため該当する節を参照されたい。

6.2 エラー処理についての考察

1. CSV データ処理中のエラー処理
2. `new_profile` 関数におけるエラー処理

6.2.1 CSV データ処理中のエラー処理

CSV データ中に、不正なデータが含まれていた場合の処理について考察する。

エラーのあった行を指摘せず、終了または無視するという方法も考えられるが、正常終了との区別が付かない上に、どの状態でエラーが発生しているのか確認をとることができないため実用的でないと考えた。今回は、エラーのあった行を指摘して、無視する方法で実装を行っている箇所が多々ある。

プログラム中に `ERROR:` で始まる表示を書いてエラー目印としている。また、エラーのあった内容を指摘するためには、`enum` という機能を利用してどのエラーなのかユーザが一度見て理解できるように、標準エラー出力を利用してエラーの内容を表示させている。

しかし、これはデータの型や仕様に指定されたデータ件数時にしかエラー処理を行っていないため他にも実装の余地はあると考える。例えば設立年月日について見ると、設立年はマイナス値、一桁や二桁は不自然だろうし設立月についても 12 より大きい数字は現在の暦では利用されていないはずである。利用する上でのエラーというものを考慮する必要があると考察する。

6.2.2 new_profile 関数におけるエラー処理

本プログラムは、データがすでに 10000 件ある状態で新規入力が行われた場合でも一度 `new_profile` 関数を実行する。といっても、関数のはじめにデータ件数を確認しており 10000 件を超えるデータは保存されないようになっている。

その状態だと条件を調べるために `new_profile` 関数を介しメモリを確保するなど無駄な作業が発生しているため、内部的な処理目的のためのエラー処理を導入してもよいのではと考えた。

実装案としては、`parse_line` を実行時に条件分岐を行う際に `profile_data_nitems` の中身を確認し、10000 件を超えるようであれば処理を行わなくすれば良い。以下実装案のプログラムである。

```
void parse_line(char *line){
    if(line[0]=='%'){
        exec_command(line[1], &line[3]);
    }
    else if(profile_data_nitems==10000){
        //件数エラーと表示するもしくは、本関数から抜ける。
    }
    else{
        new_profile(&profile_data_store[profile_data_nitems],line);
    }
}
```

6.3 新規コマンドの実装

新規コマンドについては以下の実装を行った。

1. 指定データの削除コマンド
2. どのようなコマンドあるか表示するコマンド
3. 指定要素のみ表示させるコマンド

6.3.1 指定データの削除コマンド

6.3.2 どのようなコマンドあるか表示するコマンド

6.3.3 指定要素のみ表示させるコマンド

6.4 既存コマンドの改良

既存コマンドの改良は以下の内容で行った.

1. 語句の検索の部分一致

6.4.1 語句の検索の部分一致

現在の `cmd_find` は文字列の完全一致による実装であるため、システムとして利用するには不便である. 完全一致ではなく部分一致を行うことができれば検索機能が柔軟になると考えた.

実装方法としては、検索される文字列と探したい文字列を比較を行う. 検索される文字列の中に探したい文字列の先頭文字があるかどうか探す. そこから各文字列の文字を一つずつ見ていき、探したい文字列がすべて見終わったら部分文字列が一致していることになるため検索が可能になると考える. 以下考察したプログラムである. 部分文字列が一致すると 0 を返し、それ以外は 1 を返す.

```
int find_kai(char *s, char * cp)
{
    char *s1, *s2;
    if( *cp == '\0') return s; /*cp の文字列長が 0 なら s を返す*/

    while( *s != '\0'){
        while(*s != '\0' && *s != *cp) { /*先頭文字が合うまで探す*/
            s++;
        }
        if(*s == '\0') return 1; /*見つからない*/
        s1 = s;
        s2 = cp;
        while ( *s1 == *s2 && *s1 != '\0'){ /*cp の先頭以降の文字列が一致するか*/
            s1++;
            s2++;
        }
        if( *s2 == '\0'){ /* cp の文字列は、全て一致した*/
            return 0;
        }
        s++; /*次の位置から、調べ直す*/
    }
    return 1; /*見つからない*/
}
```

7 発展課題

8 感想

課題が与えられた際は、実装方針が全くわからず完成するか不安だった。しかし、いきなりプログラムを組むのではなく、日本語で段階的に流れを組み、徐々に詳細化していき、プログラムをしていくという方法を学んだため、頭でイメージを立てながらプログラムを組むことができた。しかし、メモリ使用量などデータ構造についてはさらに検討の余地があると感じる。また、C言語は、オブジェクト指向型言語ではないがポインタや構造体を用いることでこれに近い動きをできることに驚いた。しかし、文字列の代入や値の受け渡しについては最近の言語とは異なることが多いように感じた。今回の課題を通して、ポインタと構造体に関する理解を深められたように思うが、まだまだ足りないため、考察内容を実装する中でさらに理解を深められるようにしたい。

9 作成したプログラム

作成したプログラムを以下に添付する。

ここにプログラム