プログラミング演習 1
第 4 回レポート

氏名: 重近 大智 (SHIGECHIKA, Daichi)
学生番号: 09501527

出題日: 2020 年 05 月 20 日
提出日: 2020 年 05 月 23 日
締切日: 2020 年 05 月 27 日

1　概要

本演習では，名簿管理機能を有するプログラムを，C 言語で作成する．このプログラムは，標準入力から「ID, 氏名, 誕生日, 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．ただし，% で始まる入力行はコマンド入力と解釈し，登録してあるデータを表示したり整列したりする機能も持つ．

本レポートでは，演習中に取り組んだ課題として，以下の課題 1 から課題 3 についての内容を報告する．

課題 1　文字列操作の基礎：`subst` 関数と `split` 関数の実装
課題 2　構造体や配列を用いた名簿データの定義
課題 3　標準入力の取得と構文解析
課題 4　`CSV` データ登録処理の実装
課題 5　コマンド中継処理の実装
課題 6　コマンドの実装：`%p` コマンド

また，取り組んだ課題のうち，特に以下の課題については，詳細な考察を行った．

課題 1　文字列操作の基礎：`subst` 関数と `split` 関数の実装
課題 3　標準入力の取得と構文解析
課題 4　`CSV` データ登録処理の実装

2　プログラムの作成方針

本演習で作成したプログラムが満たすべき要件と仕様として，「(1) 基本要件」と「(2) 基本仕様」を示す．

(1) 基本要件

1. プログラムは，その実行中，少なくとも 10,000 件の名簿データをメモリ中に保持できるようにすること．
2. 名簿データは，「ID，氏名，誕生日，住所，備考」を 1 つのデータとして扱えるようにすること．
3. プログラムとしての動作や名簿データの管理のために，以下の機能を持つコマンドを実装すること．
   （a）プログラムの正常な終了
   （b）登録された名簿データのデータ数表示

（c）名簿データの全数表示，および，部分表示

（d）名簿データのファイルへの保存，および，ファイルからの復元

（e）名簿データの検索と表示

（f）名簿データの整列

4. 標準入力からのユーザ入力を通して，データ登録やデータ管理等の操作を可能とすること．
5. 標準出力には，コマンドの実行結果のみを出力すること．

(2) 基本仕様

1. 名簿データは，コンマ区切りの文字列（**CSV 入力**と呼ぶ）で表されるものとし，図 1 に示したようなテキストデータを処理できるようにする．
2. コマンドは，% で始まる文字列（**コマンド入力**と呼ぶ）とし，表 1 にあげたコマンドをすべて実装する
3. 1 つの名簿データは，C 言語の構造体 (`struct`) を用いて，構造を持ったデータとしてプログラム中に定義し，使用する
4. 全名簿データは，“何らかのデータ構造”を用いて，メモリ中に保持できるようにする．
5. コマンドの実行結果以外の出力は，標準エラー出力に出力する．

3　プログラムの説明

プログラムリストは 7 章に添付している．プログラムは全部で 239 行からなる．以下では，1 節の課題ごとに，プログラムの主な構造について説明する．

3.1　文字列操作の基礎：subst 関数と split 関数の実装

まず，汎用的な文字列操作関数として，`subst()` 関数を 44–58 行目で宣言し，`split()` 関数を 60–77 行目で宣言している．また，これらの関数で利用するために，`<stdio.h>`というヘッダファイルをインクルードする．

`subst(str, C1, C2)` 関数は，`str` が指す文字列中の，文字 `C1` を文字 `C2` に置き換える．プログラム中では，`get_line()` 関数内の `fgets()` 関数で文字列の入力を受けるとき，末尾に付く改行文字を `NULL` 文字で置き換えるために使用している．呼び出し元には，文字を置き換えた回数を戻り値として返す．

`split(str, ret[], sep, max)` 関数は，他関数から渡された文字列中に文字変数 `sep` の文字に一致する文

```
1:  5100046,The Bridge,1845-11-2,14 Seafield Road Longman Inverness,SEN Unit 2.0 Open
2:  5100127,Bower Primary School,1908-1-19,Bowermadden Bower Caithness,01955 641225 ...
3:  5100224,Canisbay Primary School,1928-7-5,Canisbay Wick,01955 611337 Primary 56 3...
4:  5100321,Castletown Primary School,1913-11-4,Castletown Thurso,01847 821256 01847...
```

図 1　名簿データの CSV 入力形式の例．1 行におさまらないデータは... で省略した．

表 1　実装するコマンド

コマンド	意味	備考
%Q	終了　(Quit)	
%P n	先頭から n 件表示　(Print)	n が 0 → 全件表示， n が負 → 後ろから $-n$ 件表示
*****	（サンプルのため　省略）	

字があった場合，該当文字を NULL 文字で置き換え，該当文字の次の文字が格納されているメモリのアドレスを ret[] に書き込む．なお，ret[0] に格納されるアドレスの値は，split() 関数が呼び出された際の str の値である．以降，sep の文字に一致する文字があった場合，ret[] の添字を 1 ずつ増やしながらアドレスを格納していく．呼び出し元には，アドレスが格納されている ret[] の内，添字が最も大きいものの添字を戻り値として返す．

3.2 構造体や配列を用いた名簿データの定義

本名簿管理プログラムでは，構造体の配列を名簿データとして扱う．9–14 行目で，date 構造体を定義し，16–23 行目で，profile 構造体を定義している．この profile 型の変数 1 つが，名簿データ 1 つに相当する．そして，41 行目の profile_data_store 変数で，全名簿データを管理し，42 行目の int 型変数 profile_data_nitems で，名簿データの個数を管理する．date 構造体の定義にあたっては，年，月，日に分けて情報を管理できるよう，3 つの int 型変数を用意している．profile 構造体は，その要素に date 構造体を含む．

profile 構造体の各要素について説明する．まず，ID を格納するための int 型変数 id である．これは int 型変数の最小値-2147483648 と最大値 2147483647 の間で整数値を格納できる．次に，氏名を格納するための char 型の配列 name である．これは NULL 文字を含めて最大 70 文字の文字列を格納できる．そして，date 型の変数 birthday である．これは前述したように年，月，日の 3 つの int 型のメンバからなる変数である．次に，住所を格納するための char 型の配列 address である．文字列の配列 name 同様，NULL 文字を含めて最大 70 文字の文字列を格納できる．

3.3 標準入力の取得と構文解析

標準入力を取得するための get_line() 関数は 79–85 行目で宣言している．構文解析のための parse_line() 関数は，87–114 行目で宣言している．標準入出力のため，<stdio.h>というヘッダファイルをインクルードする．

get_line(line) 関数は，標準入力 stdin を fgets() 関数で取得し，1024 文字以上を越えた場合は，次の行として処理を行うことでバッファオーバーランを防止している．標準入力が NULL または，制御文字 ESC を 1 文字目に入力した場合は，呼び出し元に戻り値 0 を返す．それ以外の場合，subst() 関数で末尾の改行文字を NULL 文字に置き換えた後，呼び出し元に戻り値 1 を返す．

parse_line(line) 関数は，他関数からの文字列配列をポインタ line で取得し，入力内容が特定のコマンドとその引数であるか，単なる文字列であるかを判定し，それぞれ exec_command() 関数を呼び出すか，new_profile() 関数を呼び出す．get_line() 関数で入力された入力文字列の 1 文字目が % である場合，2 文字目の文字を文字変数 cmd に代入，4 文字目以降の文字列をポインタ param で参照できるようにする．入力文字列がコマンドで引数が与えられなかった場合，ポインタ param は文字列 (Null Parameter) を参照し，ポインタ param が何も指していない状態を防止している．exec_command(cmd, *param) 関数が呼び出す一部の自作関数では引数が必要となるので，この処理を行う．必要に応じて exec_command() 関数内でキャストを行ってから，コマンドの処理をする関数に引数を送る．

3.4 CSV データ登録処理の実装

CSV データ登録処理を行う new_profile(struct profile *profile_p, char *line) 関数を，183–228 行で宣言している．

new_profile() 関数で，profile 型のグローバル変数 profile_data_store[10000] に CSV データの入力情報を登録する．何番目のデータとして入力情報の登録を行うかは，int 型のグローバル変数 profile_data_nitems によって指定する．profile_data_nitems は new_profile() 関数を呼び出す際にインクリメントされるため，重複なくデータの登録が行われる．まず，split() 関数で 1 行分の入力をカンマを基準に ID，氏名，誕生日，住所，備考に分け，ポインタ配列 ret[] にそれぞれの文字列要素の先頭アドレスを格

納する．split() 関数の戻り値を用いて，要素を ID，氏名，誕生日，住所，備考に分割できていることを確認する．分割できていなかった場合は処理を中止する．そのうち，誕生日は split() 関数でハイフンを基準に年，月，日の要素に分け，ポインタ配列 ret2[] にそれぞれの要素を格納する．こちらも split() 関数の戻り値を用いて，年，月，日分割できていることを確認し，分けられていない場合は処理を中止する．処理を中止した場合，変数 profile_data_store への代入処理を行わないため，data_profile_nitems のみが，new_profile() 関数の呼び出し時にインクリメントされた状態になる．このままだと，要素が入らない状態になってしまうため，new_profile() 関数を終了する前に，profile_data_nitems をデクリメントする．

処理が中止されなかった場合，代入処理を行う．まず，ID の代入を行う．ただし，この ID に対応する変数 profile_data_store のメンバは profile 構造体で，int 型の値として宣言されている．もともとの入力は文字列であるため，これをそのまま代入することはできない．例えば，入力された ID 情報が 437 だったとしても，それは文字’4’，’3’，’7’ のことであり，整数値 437 のことではない．この文字列 437 を int 型の変数に代入するため，atoi() 関数を用いる．atoi() 関数は，文字列で表現された数値を int 型の整数値に変換するものである．変換不能な文字列の場合，結果は 0 となる [8]．次に氏名の情報，これに対応するメンバは name であるので，文字列をそのまま代入することができる．ただし，C 言語において，文字列を=で結んで代入することはできないため，strncpy() 関数を用いる．今回は，代入する文字列の最大長が 70 と予め決まっているため，strncpy() 関数を用いて，70 文字を越えた文字列の代入を阻止している [7]．続いて，誕生日を date 構造体のメンバである y，m，D に分けて代入する．ret2[0]，ret2[1]，ret2[2] がそれぞれ対応する値になっているが，これも ID の場合と同様で文字列であるので，atoi() 関数を用いて，int 型の整数値に変換してから代入する．住所情報の代入の処理は，氏名情報の代入処理と同じ処理を行うため，説明を省略する．最後に備考情報の登録であるが，備考情報には文字数の制約が無いため，何文字であっても処理が行えなければならない．すべての代入処理を終えたあと，new_profile() 関数は終了する．void 型の関数であるため，戻り値はない．

3.5 コマンド中継処理の実装

（省略）

3.6 コマンドの実装：%p コマンド

（省略）

4 プログラムの使用法と実行結果

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，2 節で説明した．

プログラムは，Cent OS 7.6.1810(Core) 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．なお，以降の実行例における，行頭の$記号は，Cent OS 7.6.1810(Core) におけるターミナルのプロンプトである．

まず，gcc でコンパイルすることで，プログラムの実行ファイルを生成する．ここで，-Wall とは通常は疑わしいものとみなされることのない構文に関して警告を出力するためのオプションであり，-o とは出力ファイル名を指定するオプションである．これらのオプションをつけることで，疑わしい構文を発見し，任意の出力ファイル名を指定することができる．

```
$ gcc program1.c -o program1 -Wall
```

次に，プログラムを実行する．以下の実行例は，プログラム実行中の動作例を模擬するため，任意の csv ファイルを標準入力のリダイレクションにより与えることで，実行する例を示している [3]．通常の利用においては，

キーボードから文字列を入力してもよい．

```
$ ./program1 < csvdata.csv
```

プログラムの出力結果として，CSV データの各項目が読みやすい形式で出力される．例えば，下記の test.csv に対して，

```
5100046,The Bridge,1845-11-2,14 Seafield Road Longman Inverness,SEN Unit 2.0 Open
5100127,Bower Primary School,1908-1-19,Bowermadden Bower Caithness,01955 641225 Primary
25 2.6 Open
```

以下のような出力が得られる．

```
line number:1
id:"5100046"
name:"The Bridge"
birthday:"1845-11-2"
address:"14 Seafield Road Longman Inverness"
biko:"SEN Unit 2.0 Open"

line number:2
id:"5100127"
name:"Bower Primary School"
birthday:"1908-1-19"
address:"Bowermadden Bower Caithness"
biko:"01955 641225 Primary 25 2.6 Open"
```

まず，入力データについて説明する．入力中の最初の 2 行で，2 つの CSV データを登録している．CSV データは自動的に分割され，変数 profile_data_store に代入された後，表示される．

5 考察

3 章のプログラムの説明，および，4 章の使用法と実行結果から，演習課題として作成したプログラムが，1 章で述べた基本要件と基本仕様のいずれも満たしていることを示した．ここでは，個別の課題のうち，以下の 3 つの項目について，考察を述べる．

1. 文字列操作の基礎：subst 関数と split 関数の実装
2. 標準入力の取得と構文解析
3. CSV データ登録処理の実装

5.1 「文字列操作の基礎：subst 関数と split 関数の実装」に関する考察

subst() 関数が，他関数から文字列の配列を受け取ることを想定して，ポインタを使用して他関数内の配列を参照できるようにした．また，入力文字列の途中に NULL 文字が出現することは標準入力では起こらないため，文字置き換えのループ処理の継続条件に NULL 文字を用いることで，確実に入力文字列の終端である NULL 文字手前まで文字の置き換えを行えるようにした．c1 と c2 に同じ文字が入力された場合，文字の置き換え処理は実質行われないに等しい．50 行目に break 文を書くことにより，文字の置き換えと置き換えられた文字のカウントを行わず，処理の簡略化及び高速化ができた．

split() 関数では，ポインタの配列 ret[] の添字を管理する int 型変数 c の取扱いに注意が必要だと考えた．他関数からポインタの配列 ret[] と int 型変数 max を引数として渡すときに，余裕なく渡していると，c のインクリメントを行うタイミングによっては，存在しない添字の ret[] を一時的に指定してしまう状態が発生する．

73 行目のアドレスの代入分の前に c のインクリメントの処理を書くことで，これを防止している．

5.2 「標準入力の取得と構文解析」に関する考察

get_line() 関数内の fgets() 関数で標準入力を取得する際，直接入力では Ctrl+D で標準入力を NULL にすることができるが，デバッグ用の機能として ESC を 1 文字目に入力することにより，入力待ちを終了できるように 82 行目に戻り値の条件を追加した．ESC は制御文字であり，ファイルリダイレクションによりファイルから入力されることはないため，これを追加した [4]．ファイルリダイレクションを fgets() 関数により取得した文字列は，終端が改行文字になってしまうため，subst() 関数を呼び出し，改行文字を NULL 文字に置き換える処理も含めている．この処理を行うことで，subst() 関数のループ処理の継続条件，split() 関数のループ処理の継続条件や printf() 関数による文字列の出力や atoi() 関数の処理などで文字列が扱いやすくなった．parse_line() 関数では，例外が発生する可能性が多いので，3 文字目以降に入力がない場合の対策が必要だと考えた．3 文字目以降に入力がない場合を考慮し，仮の (Null Parameter) を設定している．また parse_line() 関数自体が，他関数から文字列の配列の先頭アドレスを受け取り，ポインタ line に格納するが，他関数にポインタ line をそのまま渡す可能性もあったため，ポインタの値自体をインクリメントしたり，デクリメントしたりすることは避けるとともに，表記を subst() 関数や split() 関数と統一している．

5.3 「CSV データ登録処理の実装」に関する考察

new_profile() 関数内で，profile 構造体のメンバである int 型の変数 id に ID 情報を代入する際に，型変換のためキャスト演算子を使用した代入が行えるのではないかと考えたが，キャスト演算子が対象にできるのは単一の値であるため，ID の文字列の一例 437 では，初めの文字である'4' の部分しかキャストできない．これは，2 つの配列間において，代入演算子による文字列の代入ができないのと同様の理由が原因である．また，キャストを行うと文字'4' ではなく，文字コードである 52 が代入されてしまうため，値を-48 するなどの対策も必要になる．また，atoi() 関数を用いた場合と異なり，'A'（文字コード 65）と言った本来整数値ではないものの代入を行ってしまう可能性もある．以上より，キャスト演算子ではなく atoi() 関数を使うべきとの結論に至った．ただし，atoi() 関数は入力値が NULL の場合コアダンプが起こるので，入力が NULL にならないよう注意する必要がある [9]．profile 構造体のメンバである氏名，住所，備考への文字列の代入では，strcpy() 関数と，strncpy() 関数を使い分けている．文字数の上限が決まっていない備考では，strcpy() 関数を用いることで，任意長の文字列が代入できる．逆に，氏名と住所では，70 字という字数制限があるので，strncpy() 関数を用いることで，字数制限を超えない代入が可能となっている．

6 感想

毎回の講義でプログラムのソースコードを追記していくとき，どの関数を上から順に書けばいいかを考えていたが，煩雑化してしまうため，関数プロトタイプ宣言をすることで見やすくなり作業がしやすくなった [2]．今までの自作プログラムでは，関数プロトタイプ宣言が必要になるほど自作関数を用意することがなかったので，実際のプログラミングの経験になった．また，プログラム全体を通して表記のゆれが少なくなるように，多くの自作関数において，ポインタの指し先をずらすために int 型変数 i を使用したり，各関数内で似たような役割を持つ変数の変数名を統一したりした．今回の課題プログラムの作成を通して，ポインタへの理解が一層深まり，ポインタのポインタやポインタの配列に関して理解することができたと思う [5, 6]．printf() 関数の書式指定 %s の引数として，任意のアドレスを代入するが，ポインタの配列*ret[] の場合，どのような形で書けばいいのか悩んだ．printf() 関数の書式指定 %c の引数では，引数が文字コードなので，ポインタを利用して引数を指定する場合*を付ける必要があった．一方書式指定 %s の場合は，引数がアドレスのため*は必要なく，単に ret[] と書くので混乱した．

7　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．なお，1 章に示した課題については，4 章で示したようにすべて正常に動作したことを付記しておく．

作成したプログラムのソースコード (239 行)

```
#include <stdio.h>
#include <string.h>                                          /*strncpy 関数等用*/
#include <stdlib.h>                                          /*exit 関数用*/

#define ESC 27                                               /*文字列 ESC を ESC
の ASCII コードで置換*/
#define MAX_LINE 1025                                        /*文字配列 LINE の
最大入力数の指定用*/

/*構造体宣言*/
struct date
{
  int y; /*年*/
  int m; /*月*/
  int d; /*日*/
};

struct profile
{
  int id;               /*ID*/
  char name[70];        /*名前*/
  struct date birthday; /*誕生日 (date 構造体)*/
  char address[70];     /*住所*/
  char biko[70];        /*備考*/
};

/*関数プロトタイプ宣言（煩雑化防止）*/
int subst(char *str, char c1, char c2);
int split(char *str, char *ret[], char sep, int max);
int get_line(char *line);
void parse_line(char *line);
void exec_command(char cmd, char *param);
void cmd_quit();
void cmd_check();
void cmd_print();
void cmd_read();
void cmd_write();
void cmd_find();
void cmd_sort();
void new_profile(struct profile *profile_p, char *line);

/*グローバル変数宣言*/
struct profile profile_data_store[10000];                    /*profile 情報を
格納*/
int profile_data_nitems = 0;                                 /*profile 情報の
保存数を格納*/

int subst(char *str, char c1, char c2)
{
  int i;                                                     /*for ループ用*/
  int c = 0;                                                 /*置き換えた文字
数のカウント用*/
  for(i = 0; *(str + i) != '\0'; i++)                        /*入力文字列の
終端に辿り着くまでループ*/
    {
      if(c1 == c2) break;                                    /*見た目上文字列
に変化がないとき*/
      if(*(str + i) == c1)                                   /*(str + i) の
文字が c1 の文字と同じとき*/
```

```
    52                {
    53                  *(str + i) = c2;                                    /*(str + i) の
文字を c2 の文字に置き換える*/
    54                  c++;                                                /*置き換えた文字
を数える*/
    55                }
    56            }
    57          return c;                                                   /*置き換えた文字
数を戻り値とする. */
    58        }
    59
    60        int split(char *str, char *ret[], char sep, int max)
    61        {
    62          int i;                                                      /*for ループ用*/
    63          int c = 0;                                                  /*ポインタの配列
の指定用*/
    64
    65          ret[0] = str;                                               /*ret[0] に str
の先頭アドレスを代入*/
    66
    67          for(i = 0; *(str + i) != '\0'&& c < max; i++)               /*c が max より小
さいかつ入力文字列の終端に辿り着いていないときループ*/
    68            {
    69              if(*(str + i) == sep)                                   /*(str + i) が
sep のとき*/
    70                {
    71                  *(str + i) = '\0';                                  /*(str + i) に
NULL を代入*/
    72                  c++;
    73                  ret[c] = str + (i + 1);                             /*ret[c] に NULL
文字の"次の"アドレスを代入*/
    74                }
    75            }
    76          return c;                                                   /*文字列をいく
つに分割したかを戻り値とする*/
    77        }
    78
    79        int get_line(char *line)
    80        {
    81          if(fgets(line, MAX_LINE, stdin) == NULL) return 0;          /*入力文字列が
空のとき，0 を戻り値とする. 入力文字列は 1024 文字*/
    82          if(*line == ESC) return 0;                                  /*直接入力の
とき，入力文字列を空にできないため，ESC キーの単独入力により 0 を戻り値とする（デバッグ用）*/
    83          subst(line, '\n', '\0');                                    /*subst 関数に
より，入力の改行文字を終端文字に置き換える*/
    84          return 1;                                                   /*入力文字列が
存在したとき，1 を戻り値とする*/
    85        }
    86
    87        void parse_line(char *line)
    88        {
    89          char cmd;                                                   /*% の次の 1 文字
を格納用*/
    90          char *param;                                                /*コマンドの
パラメータとなる文字列へのポインタ用*/
    91          char *buffer = "(Null Parameter)";                          /*例外処理用*/
    92
    93          if(*line == '%')                                            /*入力文字列
の 1 文字目が % のとき*/
    94            {
    95              cmd = *(line + 1);                                      /*cmd に入力
文字列の 2 文字目の値を代入*/
    96              if(*(line + 3) != '\0')                                 /*パラメータ部
があるとき*/
    97                {
    98                  if(*(line + 2) != ' ')                              /*3 文字目が空白
でないとき*/
    99                    {
```

```
100                     param = line + 2;
101                     printf("引数を要するコマンド入力の場合，3 文字目は空白である必要が
あります．\n 処理を中止しました．\n\n");
102                     return;
103                   }
104                 else
105                 param = line + 3;                                   /*ポインタ line
に 3 を足したアドレスをポインタ param に代入*/
106               }
107             else param = buffer;                                    /*入力文字列に
パラメータ部が無いとき，文字列"(Null Parameter)"のアドレスをポインタ param に代入*/
108             exec_command(cmd, param);
109           }
110         else                                                        /*入力がコマンド
ではないとき*/
111           {
112             new_profile(&profile_data_store[profile_data_nitems++] ,line);
113           }
114       }
115
116       void exec_command(char cmd, char *param)
117       {
118         switch (cmd) {
119         case 'Q': cmd_quit();   break;
120         case 'T': printf("Parameter test:\"%s\"\n", param); break;   /*ポインタ param の参照先から後ろに向かっ
121         case 'C': cmd_check();  break;
122         case 'P': cmd_print();  break;
123         case 'R': cmd_read();   break;
124         case 'W': cmd_write();  break;
125         case 'F': cmd_find();   break;
126         case 'S': cmd_sort();   break;
127         default: fprintf(stderr, "%%%c command is invoked with arg: \"%s\"\n", cmd, param); break;/*エラー
128         }
129       }
130
131       void cmd_quit()
132       {
133         char c;
134
135         while(1)
136           {
137             printf("Do you want to quit?(y/n)\n");                  /*確認メッセージ*/
138             c = getchar();
139             getchar();                                              /*getchar での入力
時に改行文字が残ってしまうため*/
140             if(c == 'y')
141               {
142                 printf("quit success.\n\n");
143                 exit(0);
144               }
145             else if(c == 'n')
146               {
147                 printf("quit cancelled.\n\n");
148                 break;
149               }
150           }
151       }
152
153       void cmd_check()
154       {
155         fprintf(stderr, "Check command is invoked.\n");
156       }
157
158       void cmd_print()
159       {
160         fprintf(stderr, "Print command is invoked.\n");
161       }
162
```

```
163         void cmd_read()
164         {
165           fprintf(stderr, "Read command is invoked.\n");
166         }
167
168         void cmd_write()
169         {
170           fprintf(stderr, "Write command is invoked.\n");
171         }
172
173         void cmd_find()
174         {
175           fprintf(stderr, "Find command is invoked.\n");
176         }
177
178         void cmd_sort()
179         {
180           fprintf(stderr, "Sort command is invoked.\n");
181         }
182
183         void new_profile(struct profile *profile_p, char *line)
184         {
185           char *ret[10];
186           char *ret2[4];                                          /*誕生日の情報
を分割し，その先頭アドレスを保存*/
187           char sep = ',';                                         /*csv ファイル
からの入力を想定しているため，カンマ*/
188           char sep2 = '-';                                        /*誕生日の入力
文字列にあるハイフンで区切るため，ハイフン*/
189           int max = 10;
190           int max2 = 4;
191           int c, birth_c;
192           static int a = 1;                                       /*値を main 関数
終了時まで保持する必要があるため，static int 型*/
193
194           printf("line number:%d\n", a);                          /*行番号表示（デ
バッグ用）*/
195           //printf("input:\"%s\"\n", line);                       /*入力文字列をそ
のまま表示（デバッグ用）*/
196           c = split(line, ret, sep, max);                         /*ID，名前などの
情報を分割する*/
197           if(c <= 2)                                              /*備考以外の入力
がない場合*/
198             {
199               printf("情報は ID，名前，誕生日，住所，備考の順で入力される必要があり，備考以外
は必須項目です．\n 処理を中止しました．\n\n");
200               profile_data_nitems--;                              /*処理中止により，
構造体に情報を書き込まないため*/
201               return;
202             }
203           birth_c = split(ret[2], ret2, sep2, max2);              /*誕生日の年，月，
日を分割する*/
204           if(birth_c != 2)                                        /*誕生日の年，月，
日を正常に分割できない場合*/
205             {
206               printf("誕生日は\"年-月-日\"の形で入力される必要があります．\n 処理を中止しまし
た．\n\n"); /*年，月，日に分割できない場合，処理を停止*/
207               profile_data_nitems--;                              /*処理中止により，
構造体に情報を書き込まないため*/
208               return;
209             }
210
211           /*構造体への情報の書き込み処理*/
212           profile_p->id = atoi(ret[0]);                           /*ID の書き込み*/
213           strncpy(profile_p->name, ret[1], 70);                   /*名前の書き込み*/
214           (profile_p->birthday).y = atoi(ret2[0]);                /*誕生年の書き込み*/
215           (profile_p->birthday).m = atoi(ret2[1]);                /*誕生月の書き込み*/
216           (profile_p->birthday).d = atoi(ret2[2]);                /*誕生日の書き込み*/
```

```
   217           strncpy(profile_p->address, ret[3], 70);                  /*住所の書き込み*/
   218           if(ret[4] != NULL)
   219             strncpy(profile_p->biko, ret[4], 70);                   /*備考があるときのみ,
備考の書き込み*/
   220
   221           printf("id:\"%d\"\n", profile_p->id);
   222           printf("name:\"%s\"\n", profile_p->name);
   223           printf("birthday:\"%d-%d-%d\"\n", (profile_p->birthday).y,
(profile_p->birthday).m, (profile_p->birthday).d);
   224           printf("address:\"%s\"\n",profile_p->address);
   225           printf("biko:\"%s\"\n\n",profile_p->biko);
   226
   227           a++;
   228         }
   229
   230         int main(void)
   231         {
   232           char LINE[MAX_LINE] = {0};                                /*入力文字列 (1 行分) は
main 関数で管理*/
   233
   234           while(get_line(LINE))                                     /*文字配列 LINE に文字列
を入力する (get_line 関数)*/
   235             {
   236               parse_line(LINE);                                     /*入力文字列がある場合,
構文解析を行う (parse_line 関数)*/
   237             }
   238           return 0;
   239         }
```

参考文献

[1] 平田富雄，アルゴリズムとデータ構造，森北出版，1990.

[2] 林晴比古，明快入門 C，SB クリエイティブ，2013

[3] 入出力のリダイレクションとパイプ，http://web.sfc.keio.ac.jp/ manabu/command/contents/pipe.html，2020．05．14.

[4] IT 用語辞典 E-Words ASCII 文字コード，http://e-words.jp/p/r-ascii.html，2020．05．14

[5] １０－３．ポインタと文字列，http://www9.plala.or.jp/sgwr-t/c/sec10-3.html，2020．05．14.

[6] 4.3 ポインタ配列，http://cai3.cs.shinshu-u.ac.jp/sugsi/Lecture/c2/e_04-03.html，2020．05．14.

[7] strncpy，http://www9.plala.or.jp/sgwr-t/lib/strncpy.html，2020．5．21.

[8] 文字列を数値に変換する - C の部屋，http://www.t-net.ne.jp/ cyfis/c/stdlib/atoi.html，2020．5．21.

[9] IT 用語辞典 E-Words コアダンプ，http://e-words.jp/w/%E3%82%B3%E3%82%A2%E3%83%80%E3%83%B3%E3%83%97.html，2020．5．22.