プログラミング演習 2
期末レポート

氏名: 池田 海斗 (IKEDA, Kaito)
学生番号: 09501502

出題日: 2020 年 04 月 22 日
提出日: 2020 年 07 月 21 日
締切日: 2020 年 07 月 29 日

1　概要

本演習では，C 言語を用いて名簿管理プログラムの制作を行う．このプログラムではコマンド入力によるデータの読み込み · 書き出しを行ったり，整列 · 検索を行うことができる．また，標準入力からコンマで区切られた文字列を入力しデータの登録を行うことができる．

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 10 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
課題 7　コマンドの実装：`%R` コマンドと `%W` コマンド
課題 8　コマンドの実装：`%F` コマンド
課題 9　コマンドの実装：`%S` コマンド
課題 10　独自コマンドの実装

また，取り組んだ課題のうち，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 6　コマンドの実装：`%P` コマンド
課題 7　コマンドの実装：`%R` コマンドと `%W` コマンド
課題 9　コマンドの実装：`%S` コマンド
課題 10　独自コマンドの実装

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，利用する．
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．

```
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 ...
5100224,Canisbay Primary School,1928-7-5,Canisbay Wick,01955 611337 Primary 56 3...
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)	
`%C`	登録内容のチェック	(Check)	1 行目に登録数を必ず表示
`%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
`%R file`	$file$ から読込み	(Read)	
`%W file`	$file$ への書出し	(Write)	
`%F word`	検索結果を表示	(Find)	`%P` と同じ形式で表示
`%S n`	CSV の n 番目の項目で整列	(Sort)	表示はしない

3　プログラムの説明

プログラムリストは 8 章に添付している．プログラムは全部で 679 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 549–565 行目で宣言し，split() 関数を 575–593 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，string.h のヘッダファイルを読み込んでいる．

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() 関数の中で，文字 “\n” を終端ヌル文字 “\0” 置き換えるために，この関数を呼び出している．この関数では，文字 C1 を文字 C2 に置き換えた回数を返り値とする．

split(STR，RET，SEP，MAX) 関数では，STR が指す文字列を，最大 MAX 個まで文字 SEP の箇所で区切り，配列 RET に格納していく．プログラム中では，get_line() 関数で取得した 1 行を，カンマで区切りで配列に格納する際に呼び出される．また，ハイフン区切りで入力される生年月日のデータを，年 · 月 · 日に分けて配列に格納する際にも用いる．この関数では，配列に分割された個数を返り値とする．

3.2　構造体や配列を用いた名簿データの定義

本プログラムでは，構造体の配列に名簿データを格納していく．14–19 行目で，date 構造体を定義し，29–36 行目で，profile 構造体を定義している．ID, 氏名, 誕生日, 住所, 備考の 5 つの組み合わせで 1 つの名簿データとなる．名簿データの誕生日の箇所については date 構造体を用い，年 · 月 · 日を int 型として別々に保存しておく．こうすることで，並び替えなどを行う際に効率よく実行ができる．

そして，44 行目の profile_data_store 変数で全名簿データを管理し，43 行目の profile_data_nitems 変数で，名簿データの個数を管理している．

3.3　標準入力の取得と構文解析

get_line(LINE) 関数は，ファイルポインタが設定されていて，かつ読み込む行がある場合にそのデータを 1 行ずつ取り出し，そうでない場合には標準入力で読み込まれた名簿データを 1 行ずつ取り出し，LINE に格納する．読み込む行がない場合や，文字列が 1024 文字を超える場合に返り値 0 を返す．

parse_line(LINE) 関数では，get_line(LINE) 関数で読み込まれた 1 行が，コマンド入力かデータ入力であるか条件分岐を行う．コマンドの場合，コマンド名と引数に分けてその値を exec_command(CMD，PARAM) 関数に引き渡し，データ入力であった場合は new_profile(PROFILE_DATA_STORE，LINE) 関数で登録を行う．

3.4　CSV データ登録処理の実装

new_profile(PROFILE_DATA_STORE，LINE) 関数内では，グローバル変数 profile_data_store の任意の配列番号のポインタを受け取る．split 関数を用いて 1 行をカンマで 5 つに区切り，その中の 3 番目の要素を更にハイフンで 3 つに区切る．そして，それぞれの要素を ID· 名前 · 誕生日 · 住所 · 備考として，配列 profile_data_store の profile_data_nitems 番目の要素としてデータを保存する．誕生日に関しても同様，構造体 date の型で年 · 月 · 日に細分化しデータを格納する．また，並び替えを行いやすいように，ID· 誕生日に関しては int 型に変換して保存する．

3.5 コマンド中継処理の実装

exec_command(CMD，PARAM) 関数では，コマンド文字と引数を受け取りそれによって関数を呼び出す，案内所のような役割を行っている．引数は文字列に対応させるため，ポインタで取得するようにしてある．また一致するコマンドが見つからない場合，処理は行わないようにしてある．

3.6 コマンドの実装：%P コマンド

cmd_print(CMD，PARAM) 関数内では，入力された引数が 0, 正, 負であるかによって条件分岐される．また，登録されている要素数の絶対値より大きい引数が入力された場合，要素数分の処理のみ実行する．また，出力のフォーマットは db_sample と同じように揃えている．引数が不正な値である場合でも，atoi() 関数を用いることで 0 を入力したのと同様の動作を行うようにしてある．これは db_sample と同じ仕様である．

3.7 コマンドの実装：%R コマンドと %W コマンド

cmd_read 関数内では，入力された引数名のファイルを開いてデータを読み込む処理を行う．またファイル名が不正な値であった場合は，標準エラー出力に書き出すようにしている．正常に読み込みが行えた場合には，get_line() 関数，parse_line() 関数を呼び出して登録 · コマンド処理を行う．

cmd_write 関数内では，入力された引数名のファイルにデータを保存する処理を行う．またファイル名が不正な値であった場合は，標準エラー出力に書き出すようにしている．正常に読み込みが行えた場合には，読み込む CSV ファイルと同じ形式で ID· 名前 · 誕生日 · 住所 · 備考の組み合わせで 1 行ずつ書き出す．

3.8 コマンドの実装：%F コマンド

cmd_find 関数では，入力された引数と同じ要素を持つデータを出力する．データは完璧に入力する必要があり，また日付は 0000-00-00 の形式で入力する必要がある．

また一致するデータを 2 箇所持っていても出力は 1 回のみとなり，一致するデータは全件表示される．

3.9 コマンドの実装：%S コマンド

cmd_sort(CMD，PARAM) 関数では，[名前-1]〜[備考-5] とし引数番目のカラムで並び替えを行う．

cmd_sort(CMD，PARAM) 関数では，新たに quicksort_name 関数，quicksort_id 関数，quicksort_name 関数，quicksort_birthday 関数，quicksort_address 関数，quicksort_note 関数を用意し，それぞれ再起的に呼び出しクイックソートを行うようにした．

そして swap 関数では，profile_data_store[] 内の 2 データの入れ替えを行い，compare_date 関数では返り値の正の数 · 負の数 · ゼロで，日付の大小を比較できるようにした．

3.10 独自コマンドの実装

cmd_match(STRING，FIND) 関数内では，入力された文字列に部分一致するデータを検索して出力する．再起的に自身の関数を呼び出し，一致するものが見つかった場合返り値 1 を返す．また upper(STRING) 関数も用意し，ポインタを渡すだけで文字列を大文字に変換できるようにした．upper(STRING) 関数で利用するために，ctype.h のヘッダファイルを読み込んでいる．入力した文字列をこの関数にかけることで，大文字小文字関係なく探索を行えるようにしている．

4　プログラムの使用法と実行結果

4.1　プログラムの概要

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．コマンドを入力する際には，% で始まる英字を標準入力に打ち込み，引数で詳細な指示を行う．コマンドの詳細は 2 節に記述してある．また標準入力からのデータ入力も可能としている．

4.2　実行環境

プログラムは，MacOS Catalina 10.15.4 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，下記の実行例の行頭に書かれた「%」は，動作確認をした MacOS Catalina 10.15.6 におけるターミナルのプロンプトである．

4.3　コンパイル方法

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 とは警告オプションを全て有効にするためのオプションであり，-o とは実行ファイルの名前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，コードの視認性を高めたり無駄なコードを省くことができ，他のソースコードの実行ファイルとの識別が容易である．

```
% gcc -Wall -o eop_final_09501502 eop_final_09501502.c
```

4.4　実行方法

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データの入力を模擬するため，CSV ファイルを標準入力により与えることで，実行する例を示している．通常の利用においては，%R file によりデータを読み込む．

```
% ./eop_06_09501502.out < stdin.csv
```

4.5　出力結果

第 8 章に記述してあるプログラムを実行すると，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 stdin.csv，datastore.csv に対して，

```
5100224,Canisbay Primary School,1928-7-5,Canisbay Wick,611337 Primary Open
5100127,Bower Primary School,1908-1-19,Bowermadden Bower Caithness,641225 Primary Open
%P -2
%R datastore.csv
5100046,The Bridge,1845-11-2,14 Seafield Road Longman Inverness,SEN Unit 2.0 Open
%C
%S 3
%W stdout.csv
%F The Bridge
%M bridge
%Q


5100925,Lybster Primary School,1863-7-7,Lybster Wick,721224 Primary Open
5100720,Keiss Primary School,1863-7-3,Keiss Wick Caithness,631269 Primary Open
```

以下のような出力が得られる．

```
Id    : 5100224
Name  : Canisbay Primary School
Birth : 1928-07-05
Addr. : Canisbay Wick
Comm. : 611337 Primary Open

Id    : 5100127
Name  : Bower Primary School
Birth : 1908-01-19
Addr. : Bowermadden Bower Caithness
Comm. : 641225 Primary Open

5 profile(s)

Id    : 5100046
Name  : The Bridge
Birth : 1845-11-02
Addr. : 14 Seafield Road Longman Inverness
Comm. : SEN Unit 2.0 Open

Id    : 5100046
Name  : The Bridge
Birth : 1845-11-02
Addr. : 14 Seafield Road Longman Inverness
Comm. : SEN Unit 2.0 Open
```

以下は，生成された stdout.csv ファイルの中身である．

```
5100046,The Bridge,1845-11-02,14 Seafield Road Longman Inverness,SEN Unit 2.0 Open
5100720,Keiss Primary School,1863-07-03,Keiss Wick Caithness,631269 Primary Open
5100925,Lybster Primary School,1863-07-07,Lybster Wick,721224 Primary Open
5100127,Bower Primary School,1908-01-19,Bowermadden Bower Caithness,641225 Primary Open
5100224,Canisbay Primary School,1928-07-05,Canisbay Wick,611337 Primary Open
```

まず，入力データについて説明する．コマンドは %Q, %C, %P, %R, %W, %F, %S, %M の全てを記述している．また，この CSV ファイルから更に database.csv ファイルのデータを読み込んでおり，それぞれの項目はカンマで区切られている．出力結果の整合性は，db-sample の出力結果と，diff コマンドを用いて確認してある．

処理内容は，データを 2 件登録，%P -2 コマンドで下から 2 件分表示，%R datastore.csv コマンドで datastore.csv ファイルの読み込み，更に 1 件登録，%C コマンドで登録件数を表示，%S 3 コマンドで誕生日カラムで並び替え，%W stdout.csv コマンドで stdout.csv に登録データの書き出し，%F The Bridge コマンドで The Bridge をもつデータを表示，%M bridge コマンドで bridge 文字列を含むデータを表示，%Q コマンドで終了を行っている．

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 6 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. コマンドの実装：%P コマンド
4. コマンドの実装：%R コマンドと %W コマンド
5. コマンドの実装：%S コマンド
6. 独自コマンドの実装

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

5.1.1 「subst 関数」に関する考察

ここでは subst 関数について考察を行う．文字列の先頭にポインタを合わせて，その文字が置き換える対象の文字かどうか判断をして置換している．ポインタをインクリメントして文字を進めてゆき，終端ヌル文字で終了する．返り値は，置換した回数をカウントしておきその値とする．

5.1.2 「split 関数」に関する考察

次に，split 関数についての考察を行う．メモリを削減するために，1 つの文字列にそれぞれの要素のポインタをつけていく仕様にしてある．カンマを終端文字に置き換えることで，ポインタから終端文字までを 1 つの文字列とするため，別変数に要素をコピーをする必要もない．ポインタアドレスをコピーした後は，ポインタを終端ヌル文字の分 1 つ右へ移動させることで，次の文字列の文頭にポインタを移動している．

5.2 「標準入力の取得と構文解析」に関する考察

5.2.1 「get_line 関数」に関する考察

get_line 関数についての考察を行う．今回重要なポイントとなってくるところは，fgets の最大文字数を 1024 ではなく 1025 に設定したところである．もちろん，main 関数内で line 配列も 1025 文字にしているので，バッファオーバーフローを起こすことはない．strlen 関数などの C 言語の関数は終端ヌル文字をカウントしないことが多いので，終端ヌル文字を別の文字に置き換えたりする場合でもエラーが起こらないように考慮した．

5.2.2 「parse_line 関数」に関する考察

次に，parse_line 関数についての考察を行う．まずコマンドか否かの判別は，文字頭の値が % で始まるかどうかで行っている．これは，データ入力の場合文字列の始めは % 以外で始まるため，上記の条件に含まれることはないからである．658 行目では，ポインタの 4 文字目以降からヌル終端文字までを配列に格納している．

5.3 コマンドの実装：%P コマンド

cmd_print(CMD, PARAM) 関数についての考察を行う．まず，atoi() 関数で引数を int 型に変換する．引数が 0 の場合，正の場合，負の場合で条件分岐を行う．また，314 行目 · 330 行目では，引数の絶対値が要素数よりも大きい場合は，要素数を用いて処理を行う．今回考慮するべきは，引数が入力されていない場合や数字以外が入力された場合であり，atoi() 関数ではそのような値に 0 という値を返すため db_sample の仕様通りになる．

5.4 コマンドの実装：%R コマンドと %W コマンド

5.4.1 「cmd_read 関数」に関する考察

cmd_read(CMD, PARAM) 関数についての考察を行う．まず，ファイルポインタ fp に読み込みファイルのポインタをあわせ，ファイルが存在していれば続行する．これで fp に情報が入っているので，get_line 関数で fp から 1 行を読み取る処理に切り替わる．get_line 関数と parse_line 関数をループするところは main 関数と同じ動作である．

5.4.2 「cmd_write 関数」に関する考察

次に，cmd_read(CMD，PARAM) 関数についての考察を行う．ファイルポインタ fp の取り扱いは cmd_read 関数と同じであり，ファイルが存在すれば上書き，または新規作成となる．fprintf 関数とは，1 つ目の引数（この場合 fp）に 2 つ目の引数の文字列を書き込むという関数である．

cmd_read 関数も cmd_write 関数も，ファイルが存在しない · 書き込み権限がない際の警告は標準エラー出力に書き出すようにしている．

5.5 コマンドの実装：%S コマンド

cmd_sort(CMD，PARAM) 関数についての考察を行う．cmd_sort(CMD，PARAM) 関数では引数に応じて，switch 文で各クイックソート関数を呼び出している．各クイックソート関数では自身を再起的に呼び出し，並び替えが完了すると処理が終了する．

また，誕生日カラムのクイックソートについて，日付の大小を比べるために compare_date(D1，D2) 関数を用意した．「年」の時点で大小が決まればその差を返し，もし同じだった場合には「月」を比べ，最終的には「日」の差を返す．そのため関数の返り値で判別するには，正の数が返れば d1 の方が大きく，0 が返れば日付が一致，負の数が返れば d2 の方が大きいと処理する．

5.6 独自コマンドの実装

cmd_match(CMD，PARAM) 関数についての考察を行う．まずこの関数の 475 行目 · 479 行目で，探索するデータを 1 行に戻しておき，探索回数をできるだけ少なくするように工夫した．

また，upper(STRING) 関数という，ポインタで渡された文字列を大文字に変換する関数を用意し，部分一致検索に用いる変数を全て通してある．これは検索時に大文字 · 小文字関係なく入力できるようにするためである．

あとこの関数内では，match(STRING，FIND) 関数に渡し true が返った場合，そのデータを出力するだけである．

match(STRING，FIND) 関数では，引数 string · find を受け取る．これはそれぞれ，探索対象となる文字列と探し出したい文字列のポインタである．まず，251 行目 · 252 行目でポインタのオリジナルを保存しておく．仕様としては，string のポインタを 1 文字ずつ進めてゆき，find の先頭文字と一致すれば string と find の両方のポインタを 1 文字ずらし，再び一致するかを繰り返していく．一致しなかった場合は find ポインタが先頭に戻され，string が終端ヌル文字まで到達かつ find が終端ヌル文字まで到達していない場合，返り値 0 で終了する．find が終端ヌル文字まで到達した場合は返り値 1 で終了する．

ここで問題となるのが，string が aabcd，find が abcd の時である．2 文字目が一致しないので，find ポインタが先頭に戻されてしまう．そのため，非同期処理で（オリジナルの）find 文字列の 1 文字目が出てくるかの確認を随時行い，検出された場合はその文字を先頭とする文字列とオリジナルの find で再起呼び出しを行う．

6 発展課題

7 感想

今回のプログラムでは，split() 関数で苦労した．私は最初，カンマ区切りの要素数よりも max 値の方が小さければ，配列 ret の max 番目の要素には模範解答のように残りの文字列をむりやり入れるような仕様にはしていなかった．

8 章添付のコードのように，受け取った文字列 str をまず初めに subst() 関数にかけることで，カンマを全て終端 NULL 文字に置き換え，ポインタを用いて一気に文字列を代入できる方法を思いついた．カンマを終端 NULL 文字に置き換えるという発想自体は良かったのだが，これだと仕様を満たしていなかったためテストを通過でき

なかった．

また，C 言語にもともと用意されている関数を使う場面も多く，リファレンスから行いたい処理をしてくれる関数を見つけ出す検索力もプログラミングには必要だなと感じた．希望する動作を行ってくれる関数を探し出す力，バグの原因と解決策を自分で見つけ出す力など，行く行くはプログラムを行なっていく際に自分で解決する力を養うことが今の僕たちには大切だなと感じた．

8 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．

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX_LINE_LEN 1024 /* Maximum characters per line */

/*
Overview: Structure for specifying date.
@type: {int} y - Year.
@type: {int} m - Month.
@type: {int} d - Day.
*/
struct date
{
    int y;
    int m;
    int d;
};

/*
Overview: Structure for specifying user profiles.
@type: {int} id - ID.
@type: {char} name - Name.
@type: {struct date} birthday - Birthday.
@type: {char} address - Address.
@type: {char} note - Others.
*/
struct profile
{
    int id;
    char name[70];
    struct date birthday;
    char address[70];
    char note[1024];
};

/*
@type: {int} profile_data_nitems - Total number of registered items.
@type: {struct profile} profile_data_store[] - For storing registered data.
@type: {FILE *} fp - Pointer for writing/reading.
*/
int profile_data_nitems = 0;
struct profile profile_data_store[10000];
FILE *fp;

int get_line(char *line);
void parse_line(char *line);

/*
Overview: Swap elements in profile_data_store array.
@argument: {struct profile*} source - Replacement source.
@argument: {struct profile*} destination - Replace destination.
```

```
@return: No return
*/
void swap(struct profile *source, struct profile *destination)
{
    struct profile tmp;

    tmp = *source;
    *source = *destination;
    *destination = tmp;
}

/*
Overview: Quick sort profile_data_store array by id column.
@argument: {int} low - Left edge of quick sort.
@argument: {int} high - Right edge of quick sort.
@return: No return
*/
void quicksort_id(int low, int high)
{
    if (low < high)
    {
        int mid = (low + high) / 2;
        int x = profile_data_store[mid].id;
        int i = low;
        int j = high;
        while (i <= j)
        {
            while (profile_data_store[i].id < x)
                i += 1;
            while (profile_data_store[j].id > x)
                j -= 1;
            if (i <= j)
                swap(&profile_data_store[i++], &profile_data_store[j--]);
        }
        quicksort_id(low, j);
        quicksort_id(i, high);
    }
}

/*
Overview: Quick sort profile_data_store array by name column.
@argument: {int} low - Left edge of quick sort.
@argument: {int} high - Right edge of quick sort.
@return: No return
*/
void quicksort_name(int low, int high)
{
    if (low < high)
    {
        int mid, i, j;
        char x[70];
        mid = (low + high) / 2;
        strcpy(x, profile_data_store[mid].name);
        i = low;
        j = high;
        while (i <= j)
        {
            while (strcmp(profile_data_store[i].name, x) < 0)
                i += 1;
            while (strcmp(profile_data_store[j].name, x) > 0)
                j -= 1;
            if (i <= j)
                swap(&profile_data_store[i++], &profile_data_store[j--]);
        }
        quicksort_name(low, j);
        quicksort_name(i, high);
    }
}
```

```
122
123 /*
124 Overview: Compares two dates and returns the difference.
125 @argument: {struct date *} d1 - Date1.
126 @argument: {struct date *} d2 - Date2.
127 @return: {int} (Date1 - Date2) - Positive or negative or zero.
128 */
129 int compare_date(struct date *d1, struct date *d2)
130 {
131     if (d1->y != d2->y)
132         return d1->y - d2->y;
133     if (d1->m != d2->m)
134         return d1->m - d2->m;
135     return d1->d - d2->d;
136 }
137
138 /*
139 Overview: Quick sort profile_data_store array by birthday column.
140 @argument: {int} low - Left edge of quick sort.
141 @argument: {int} high - Right edge of quick sort.
142 @return: No return
143 */
144 void quicksort_birthday(int low, int high)
145 {
146     if (low < high)
147     {
148         int mid = (low + high) / 2;
149         struct date x = profile_data_store[mid].birthday;
150         int i = low;
151         int j = high;
152         while (i <= j)
153         {
154             while (compare_date(&profile_data_store[i].birthday, &x) < 0)
155                 i += 1;
156             while (compare_date(&profile_data_store[j].birthday, &x) > 0)
157                 j -= 1;
158             if (i <= j)
159                 swap(&profile_data_store[i++], &profile_data_store[j--]);
160         }
161         quicksort_birthday(low, j);
162         quicksort_birthday(i, high);
163     }
164 }
165
166 /*
167 Overview: Quick sort profile_data_store array by address column.
168 @argument: {int} low - Left edge of quick sort.
169 @argument: {int} high - Right edge of quick sort.
170 @return: No return
171 */
172 void quicksort_address(int low, int high)
173 {
174     if (low < high)
175     {
176         int mid, i, j;
177         char x[70];
178         mid = (low + high) / 2;
179         strcpy(x, profile_data_store[mid].address);
180         i = low;
181         j = high;
182         while (i <= j)
183         {
184             while (strcmp(profile_data_store[i].address, x) < 0)
185                 i += 1;
186             while (strcmp(profile_data_store[j].address, x) > 0)
187                 j -= 1;
188             if (i <= j)
189                 swap(&profile_data_store[i++], &profile_data_store[j--]);
```

```
        }
        quicksort_address(low, j);
        quicksort_address(i, high);
    }
}

/*
Overview: Quick sort profile_data_store array by note column.
@argument: {int} low - Left edge of quick sort.
@argument: {int} high - Right edge of quick sort.
@return: No return
*/
void quicksort_note(int low, int high)
{
    if (low < high)
    {
        int mid, i, j;
        char x[1024];
        mid = (low + high) / 2;
        strcpy(x, profile_data_store[mid].note);
        i = low;
        j = high;
        while (i <= j)
        {
            while (strcmp(profile_data_store[i].note, x) < 0)
                i += 1;
            while (strcmp(profile_data_store[j].note, x) > 0)
                j -= 1;
            if (i <= j)
                swap(&profile_data_store[i++], &profile_data_store[j--]);
        }
        quicksort_note(low, j);
        quicksort_note(i, high);
    }
}

/*
Overview: Converts lowercase letters to uppercase.
@argument: {char *} string - The string to convert.
@return: No return
*/
void upper(char *string)
{
    while (*string)
    {
        *string = toupper(*string);
        string++;
    }
}

/*
Overview: Search by partial match.
@argument: {char *} string - string to search.
@argument: {char *} find - string to find.
@return: No return
*/
int match(char *string, char *find)
{
    int i;
    char *string_tmp, *find_tmp;

    string_tmp = string;
    find_tmp = find;

    for (i = 0; i <= strlen(string); i++)
    {
        if (!*find_tmp)
            return 1;
```

```
        if (*string_tmp == *find && *string_tmp != *find_tmp)
            if (match(string_tmp, find))
                return 1;
        if (*string_tmp == *find_tmp)
            find_tmp++;
        else
            find_tmp = find;
        string_tmp++;
    }
    return 0;
}

/*
Overview: Exit the program.
@return: No return
*/
void cmd_quit(void)
{
    exit(0);
}

/*
Overview: Output the number of registrations.
@argument: {char} cmd - Command alphabet.
@return: No return
*/
void cmd_check(char cmd)
{
    printf("%d profile(s)\n", profile_data_nitems);
}

/*
Overview: Output data according to argument.
@argument: {char} cmd - Command alphabet.
@argument: {char *} param - Command argument.
@return: No return
*/
void cmd_print(char cmd, char *param)
{
    int count, num = atoi(param);
    if (num == 0)
    {
        count = 0;
        while (count < profile_data_nitems)
        {
            printf("Id    : %d\n", profile_data_store[count].id);
            printf("Name  : %s\n", profile_data_store[count].name);
            printf("Birth : %04d-%02d-%02d\n", profile_data_store[count].birthday.y,
                profile_data_store[count].birthday.m, profile_data_store[count].birthday.d);
            printf("Addr. : %s\n", profile_data_store[count].address);
            printf("Comm. : %s\n\n", profile_data_store[count].note);
            count++;
        }
    }
    else if (num > 0)
    {
        if (num > profile_data_nitems)
            num = profile_data_nitems;
        count = 0;
        while (count < num)
        {
            printf("Id    : %d\n", profile_data_store[count].id);
            printf("Name  : %s\n", profile_data_store[count].name);
            printf("Birth : %04d-%02d-%02d\n", profile_data_store[count].birthday.y,
                profile_data_store[count].birthday.m, profile_data_store[count].birthday.d);
            printf("Addr. : %s\n", profile_data_store[count].address);
            printf("Comm. : %s\n\n", profile_data_store[count].note);
            count++;
```

```
326         }
327     }
328     else if (num < 0)
329     {
330         if (num < -profile_data_nitems)
331             num = -profile_data_nitems;
332         count = profile_data_nitems + num;
333         while (count < profile_data_nitems)
334         {
335             printf("Id    : %d\n", profile_data_store[count].id);
336             printf("Name  : %s\n", profile_data_store[count].name);
337             printf("Birth : %04d-%02d-%02d\n", profile_data_store[count].birthday.y,
338                 profile_data_store[count].birthday.m, profile_data_store[count].birthday.d);
339             printf("Addr. : %s\n", profile_data_store[count].address);
340             printf("Comm. : %s\n\n", profile_data_store[count].note);
341             count++;
342         }
343     }
344 }
345 
346 /*
347 Overview: Read data and register in array.
348 @argument: {char} cmd - Command alphabet.
349 @argument: {char *} param - Command argument.
350 @return: No return
351 */
352 void cmd_read(char cmd, char *param)
353 {
354     char line[MAX_LINE_LEN + 1];
355     fp = fopen(param, "r");
356     if (fp != NULL)
357     {
358         while (get_line(line))
359         {
360             parse_line(line);
361         }
362     }
363     else
364     {
365         fprintf(stderr, "Enterd file cannot be opened.\n");
366     }
367     fclose(fp);
368 }
369 
370 /*
371 Overview: Export registered data.
372 @argument: {char} cmd - Command alphabet.
373 @argument: {char *} param - Command argument.
374 @return: No return
375 */
376 void cmd_write(char cmd, char *param)
377 {
378     int i;
379     fp = fopen(param, "w");
380     if (fp != NULL)
381     {
382         for (i = 0; i < profile_data_nitems; i++)
383         {
384             fprintf(fp, "%d,%s,%04d-%02d-%02d,%s,%s\n", profile_data_store[i].id,
385                 profile_data_store[i].name, profile_data_store[i].birthday.y,
386                 profile_data_store[i].birthday.m, profile_data_store[i].birthday.d,
387                 profile_data_store[i].address, profile_data_store[i].note);
388         }
389     }
390     else
391     {
392         fprintf(stderr, "Enterd file cannot be opened.\n");
393     }
```

```
394     fclose(fp);
395 }
396
397 /*
398 Overview: Search for matching data from registered data and output.
399 @argument: {char} cmd - Command alphabet.
400 @argument: {char *} param - Command argument.
401 @return: No return
402 */
403 void cmd_find(char cmd, char *param)
404 {
405     int i;
406     char id_tmp[9];
407     char birthday_tmp[11];
408     struct profile *p;
409     for (i = 0; i < profile_data_nitems; i++)
410     {
411         p = &profile_data_store[i];
412         sprintf(id_tmp, "%d", p->id);
413         sprintf(birthday_tmp,"%04d-%02d-%02d", p->birthday.y, p->birthday.m, p->birthday.d);
414         if (
415             strcmp(id_tmp, param) == 0 ||
416             strcmp(p->name, param) == 0 ||
417             strcmp(birthday_tmp, param) == 0 ||
418             strcmp(p->address, param) == 0 ||
419             strcmp(p->note, param) == 0)
420         {
421             printf("Id    : %d\n", profile_data_store[i].id);
422             printf("Name  : %s\n", profile_data_store[i].name);
423             printf("Birth : %04d-%02d-%02d\n", profile_data_store[i].birthday.y,
424                 profile_data_store[i].birthday.m, profile_data_store[i].birthday.d);
425             printf("Addr. : %s\n", profile_data_store[i].address);
426             printf("Comm. : %s\n\n", profile_data_store[i].note);
427         }
428     }
429 }
430
431 /*
432 Overview: Specify the column with an argument and sort the registered data.
433 @argument: {char} cmd - Command alphabet.
434 @argument: {char *} param - Command argument.
435 @return: No return
436 */
437 void cmd_sort(char cmd, char *param)
438 {
439     switch (atoi(param))
440     {
441     case 1:
442         quicksort_id(0, profile_data_nitems - 1);
443         break;
444     case 2:
445         quicksort_name(0, profile_data_nitems - 1);
446         break;
447     case 3:
448         quicksort_birthday(0, profile_data_nitems - 1);
449         break;
450     case 4:
451         quicksort_address(0, profile_data_nitems - 1);
452         break;
453     case 5:
454         quicksort_note(0, profile_data_nitems - 1);
455         break;
456     default:
457         break;
458     }
459 }
460
461 /*
```

```
462 Overview: Output partial match data in array.
463 @argument: {char} cmd - Command alphabet.
464 @argument: {char *} param - Command argument.
465 @return: No return
466 */
467 void cmd_match(char cmd, char *param)
468 {
469     int i;
470     for (i = 0; i < profile_data_nitems; i++)
471     {
472         char string[2][1024];
473         char find[1024];
474 
475         sprintf(string[0], "%d, %s, %d-%d-%d, %s, %s\n", profile_data_store[i].id,
476             profile_data_store[i].name, profile_data_store[i].birthday.y,
477             profile_data_store[i].birthday.m, profile_data_store[i].birthday.d,
478             profile_data_store[i].address, profile_data_store[i].note);
479         sprintf(string[1], "%d, %s, %d-%02d-%02d, %s, %s\n", profile_data_store[i].id,
480             profile_data_store[i].name, profile_data_store[i].birthday.y,
481             profile_data_store[i].birthday.m, profile_data_store[i].birthday.d,
482             profile_data_store[i].address, profile_data_store[i].note);
483 
484         strcpy(find, param);
485 
486         upper(string[0]);
487         upper(string[1]);
488         upper(find);
489 
490         if (match(string[0], find) || match(string[1], find))
491         {
492             printf("Id    : %d\n", profile_data_store[i].id);
493             printf("Name  : %s\n", profile_data_store[i].name);
494             printf("Birth : %04d-%02d-%02d\n", profile_data_store[i].birthday.y,
495                 profile_data_store[i].birthday.m, profile_data_store[i].birthday.d);
496             printf("Addr. : %s\n", profile_data_store[i].address);
497             printf("Comm. : %s\n\n", profile_data_store[i].note);
498         }
499     }
500 }
501 
502 /*
503 Overview: Calls functions when the command is input.
504 @argument: {char} cmd - Command alphabet.
505 @argument: {char *} param - Command argument.
506 @return: No return
507 */
508 void exec_command(char cmd, char *param)
509 {
510     switch (cmd)
511     {
512     case 'Q':
513         cmd_quit();
514         break;
515     case 'C':
516         cmd_check(cmd);
517         break;
518     case 'P':
519         cmd_print(cmd, param);
520         break;
521     case 'R':
522         cmd_read(cmd, param);
523         break;
524     case 'W':
525         cmd_write(cmd, param);
526         break;
527     case 'F':
528         cmd_find(cmd, param);
529         break;
```

```
    case 'S':
        cmd_sort(cmd, param);
        break;
    case 'M':
        cmd_match(cmd, param);
        break;
    default:
        fprintf(stderr, "Unregistered Command Is Entered.\n");
        break;
    }
}

/*
Overview: Replaces c1 in the string with c2.
@argument: {char *} str - String.
@argument: {char} c1 - Replaced.
@argument: {char} c2 - Replace.
@return: {int} diff - Number of replacements.
*/
int subst(char *str, char c1, char c2)
{
    int diff = 0;
    char *p;

    p = str;
    while (*p != '\0')
    {
        if (*p == c1)
        {
            *p = c2;
            diff++;
        }
        p++;
    }
    return diff;
}

/*
Overview: Separate string by the specified number of characters/times.
@argument: {char *} str - String.
@argument: {char *} ret[] - Separated string.
@argument: {char} sep - Delimiter.
@argument: {int} max - Maximum number to divide.
@return: Number of divisions
*/
int split(char *str, char *ret[], char sep, int max)
{
    int count = 1;
    ret[0] = str;

    while (*str)
    {
        if (count >= max)
            break;
        if (*str == sep)
        {
            *str = '\0';
            ret[count++] = str + 1;
        }
        str++;
    }

    return count;
}

/*
Overview: Get line from file or standard input.
@argument: {char *} line - Full text.
```

```
598 @return: Whether there is next line.
599 */
600 int get_line(char *line)
601 {
602     if (fp != NULL && fgets(line, MAX_LINE_LEN + 1, fp) != NULL)
603     {
604         subst(line, '\n', '\0');
605         return 1;
606     }
607     if (fgets(line, MAX_LINE_LEN + 1, stdin) == NULL)
608     {
609         return 0;
610     }
611     else
612     {
613         subst(line, '\n', '\0');
614         return 1;
615     }
616 }
617
618 /*
619 Overview: New data registration.
620 @argument: {struct profile *} profile_data_store - Pointer to store the new data.
621 @argument: {char *} line - One line to register.
622 @return: Successful or not.
623 */
624 int new_profile(struct profile *profile_data_store, char *line)
625 {
626     int max_line = 5, max_date = 3;
627     char *ret[80] = {0}, *date[80] = {0}, sep_line = ',', sep_date = '-';
628
629     if (split(line, ret, sep_line, max_line) != 5)
630     {
631         return -1;
632     }
633
634     split(line, ret, sep_line, max_line);
635     split(ret[2], date, sep_date, max_date);
636
637     profile_data_store->id = atoi(ret[0]);
638     strcpy(profile_data_store->name, ret[1]);
639
640     profile_data_store->birthday.y = atoi(date[0]);
641     profile_data_store->birthday.m = atoi(date[1]);
642     profile_data_store->birthday.d = atoi(date[2]);
643
644     strcpy(profile_data_store->address, ret[3]);
645     strcpy(profile_data_store->note, ret[4]);
646     return 0;
647 }
648
649 /*
650 Overview: Check new data registration or command.
651 @argument: {char *} line - One line.
652 @return: No return
653 */
654 void parse_line(char *line)
655 {
656     if (*line == '%')
657     {
658         exec_command(line[1], &line[3]);
659     }
660     else
661     {
662         new_profile(&profile_data_store[profile_data_nitems++], line);
663     }
664 }
665
```

```
/*
Overview: Main function.
@return: Successful or not.
*/
int main(void)
{
    char line[MAX_LINE_LEN + 1];
    while (get_line(line))
    {
        parse_line(line);
    }
    return 0;
}

```

添付コード

```
int split(char *str, char *ret[], char sep, int max)
{
    int i, count = 0;
    subst(str, sep, '\0'); // カンマを NULL 終端に置き換え
    for (i = 0; i < max; i++)
    {
        ret[i] = str;
        str += strlen(str) + 1;
        count++;
    }
    return count;
}
```

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] C 言語の atoi で出来ること, https://arma-search.jp/article/clanguage-atoi, 2020/05/20.