

# プログラミング演習 2

## 期末レポート

氏名: 重近 大智 (SHIGECHIKA, Daichi)

学生番号: 09501527

出題日: 2020 年 06 月 17 日

提出日: 2020 年 07 月 28 日

締切日: 2020 年 07 月 29 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 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 %R コマンドと %W コマンド

課題 8 %F コマンド

課題 9 %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、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

課題 7 %R コマンドと %W コマンド

課題 8 %F コマンド

課題 9 %S コマンド

## 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) 追加仕様

1. 名簿データの各項目には、コンマは含まれないものとする。
2. プログラムの入力は、常に半角文字や制御文字のみで構成されており、全角文字 (例えば、日本語の漢字) は含まれないものとする。
3. コマンドの主要な機能は、% に続く 1 文字で区別されるものとする。
4. “何らかのデータ構造”は、“固定長の配列”とする。
5. `int` 型は、32 bit (4 bytes) の整数型を扱える変数とする。

## 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。最終的なプログラムは全部で 627 行からなる。以下では、1 章の課題ごとに、プログラムの主な構造について説明する。なお、特筆のない限り説明は最新のソースコード (8.7 節) に基づく。また、多くの関数において例外対策やユーザフレンドリーなメッセージが実装されているが、それらには 6 章で触れるため、本章での説明は最小限にとどめる。

### 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 49–63 行目で宣言し、`split()` 関数を 65–81 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に 1 を加えたもの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。10–15 行目で、`date` 構造体を定義し、17–24 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、46 行目の `profile_data_store` 変数で、全名簿データを管理し、47 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。最後に、備考を格納するための `char` 型のポインタ `biko` である。文字列の先頭アドレスを格納する。

## 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 83–91 行目で宣言している。構文解析のための `parse_line()` 関数は、93–113 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。標準入力がないときは、呼び出し元に戻り値 0 を返して関数は終了する。制御文字 `ESC` を 1 文字目に入力した場合は、コマンド `%Q` を引数 `r` つきで呼び出した動作（デバッグ用の機能）を行う。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、名簿データを入力した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、`split()` 関数を呼び出し、`line` の 2 文字目のアドレス、ポインタの配列 `ret`、区切り文字である半角スペース、最大区切り数 2 を渡す。そして、`exec_command()` 関数を呼び出し、`ret[0]`、`ret[1]` を引数として渡す。1 文字目が `%` でない場合は、名簿データが既に 10000 件登録されていないことを確認し、名簿データの新規登録を行う `new_profile()` 関数を呼び出す。呼び出しの際に、次に登録する名簿データに対応する `profile_data_store` のアドレスを渡し、渡した後に `profile_data_nitems` をインクリメントする。既に 10000 件登録されている場合は登録を中止し、エラーメッセージを表示する。登録を中止した件数は `int` 型の変数 `i` でカウントする。

### 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(profile_p, line)` 関数を、198–236 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、`input_format_check()` 関数により、文字列 `line` をカンマ、セミコロン、タブのどの文字で区切るかを検討する。いずれかの文字が 4 回 `line` に含まれていれば、カンマ、セミコロン、タブの優先度で区切りを行う用意をする。戻り値は区切る文字の文字コードとなっている。いずれの文字でも区切れないと見込まれる場合、エラーメッセージを表示し、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。それとともに登録を中止した件数を `int` 型の変数 `i` でカウントする。次に、`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。続いて ID が `int` 型の値に `atoi()` 関数で正常に変換できることを `int_value_check()` 関数で確認し、かつ変換後に負の値を取らないことを確認する。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。こちらは `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。誕生日の年、月、日それぞれが `atoi()` 関数で正常に変換できるか、`int_value_check()` 関数で確認を行う。ID とは異なり、こちらは負の値にならないかどうかを確認する必要はない。区切り文字である ‘-’ は、マイナスとハイフンで共通の文字であるからである。続いて、入力された誕生日の情報が実在する日付けかどうか、`day_format_check()` 関数を用いて確認を行う。

処理が中止されなかった場合、次の代入処理を行う。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。予め備考の文字数を `strlen()` 関数で取得し、`int` 型変数 `MAX_BIKO` に格納し、NULL 文字分の 1 を足しておく [10]。次に、`malloc()` 関数にて必要分のメモリを確保する [11]。`malloc()` 関数の戻り値は、アドレスであるが、アドレスは 4 バイト長であるため `int` 型なのか `char` 型なのかといったことが分からない。そのため、キャスト演算子を用いて `char` 型の値を指すアドレスだと明示的に指定する [12]。最後に、`strncpy()` 関数で、先程確保したメモリに文字列をコピーしていく。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

コマンド中継処理を行う `exec_command(cmd, param)` 関数を、454–531 行で宣言している。なお、使用可能なコマンドは表 1 に記載している。

この関数は、`parse_line()` 関数からコマンドの種類を決定する 1-2 文字の文字列の先頭アドレスを格納する `cmd` とコマンドの引数とする文字列の先頭アドレスを格納する `param` を受け取る。`switch` 文で `cmd` の値によって、呼び出す関数の一文字目を選択する。存在しないコマンドを呼び出そうとした場合、`default` の項目が実行され、エラーメッセージを出力する。‘W’ から始まるコマンド群である ‘W’, “WS”, “WT” コマンドに関しては、別途 `strcmp()` 関数を用いて判定を行う。`void` 型の関数であるため、戻り値はない。

%Q コマンドに対応する `cmd_quit()` 関数を、168–195 行目で宣言している。%Q コマンドは、`exec_command()` 関数の `switch` 文中で `cmd_quit()` 関数を呼び出し、プログラムを終了する役割を持つ。`cmd_quit()` 関数は、終了するかの確認メッセージを出力し、‘y’ か ‘n’ の入力を求める。引数 `r` を入力していれば、確認メッセージは表示しない。正常終了を示す 0 をシェルに返すため、`exit(0)` として、`exit()` 関数を呼び出す。`exit()` 関数を使用するために、`stdlib.h` をインクルードする。`void` 型の関数であるため、戻り値はない。

%C コマンドに対応する `cmd_check()` 関数を、197–200 行目で宣言している。%C コマンドは、`exec_command()` 関数の `switch` 文中で `cmd_check()` 関数を呼び出す。`cmd_check()` 関数に引数はなく、プログラム中に読み込まれている名簿データの総数を表示する関数である。`int` 型のグローバル変数である `profile_data_nitems` に、名簿データの総数を格納しているので、この値を表示する。`void` 型の関数であるため、戻り値はない。

3.6 コマンドの実装：%P コマンド

%P コマンドに対応する `cmd_print()` 関数を、202–244 行目で宣言している。%P コマンドは、引数の条件に従って名簿データの中身を表示する関数である `cmd_print(*param)` 関数を呼び出す。この関数は、`exec_command()` 関数経由で、`parse_line()` 関数からパラメータ部の先頭アドレスを受け取る。この時点では、引数として入力

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)	
%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 から読み込み (Read)	
%W file	file への書出し (Write)	CSV 形式で書出し
%WS file	file への書出し (Write)	SCSV 形式で書出し
%WT file	file への書出し (Write)	TSV 形式で書出し
%W 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

された数値は文字列の状態なので、`atoi()` 関数を用いて、`int` 型の整数値に変換して、`int` 型の変数 `a` に格納する。このとき、`a` の絶対値が `data_profile_nitems` 以上か 0 のとき、`a` に `data_profile_nitems` を代入する。これにより、まだ名簿データが格納されていない部分を表示するのを防止している。それ以外の時はそのまま名簿データの表示処理に進む。`a` が正のとき、先頭から `a` 件の名簿データを昇順で表示する。`a` が負のとき、後ろから `a` 件を昇順で表示する。グローバル変数 `data_profile_store[]` の添字に、`for` ループで変化する `int` 型変数 `i` を用いることで表示する名簿データを指定する。`void` 型の関数であるため、戻り値はない。

### 3.7 %R コマンドと %W コマンド

%R コマンドに対応する `cmd_read(param)` 関数を、246–269 行目で宣言している。%R コマンドは、実行ファイルと同ディレクトリ内にある、`param` で指定された文字列と同じ名前のファイルを開き、その内容を `get_line()` で入力処理する関数である。まず、コマンドの実行に必要な引数が存在するかを確認し、なければエラーメッセージを表示して処理を中止する。また、指定されたファイルが存在せず、開けない場合もエラーメッセージの表示を行い、処理を中止する。次に、`get_line()` 関数で 1 行ずつファイルの内容を読んでいく。引数としてファイルポインタ `fp` と読み込んだ 1 行分の文字列 `char` 型の配列 `LINE` を渡す。次に `parse_line()` 関数で 1 行分の入力を解析し、対応する関数の呼び出しを行う。すべての文字列を読み込んだら、ファイルはもう使用しないため、ファイルポインタを閉じる。`void` 型の関数であるため、戻り値はない。

%W コマンドに対応する `cmd_write(param, sep)` 関数を、271–299 行目で宣言している。%W コマンドは、実行ファイルがあるディレクトリ内に `param` で指定された文字列と同じ名前のファイルを作成し、名簿データを `sep` で指定された文字で区切りながら出力する。%W コマンドを実行した場合、`sep` の文字はカンマになる。まず、コマンドの実行に必要な引数が存在するかを確認し、なければエラーメッセージを表示して処理を中止する。また、指定されたファイルが読み取り専用であったり、開けなかったりする場合もエラーメッセージの表示を行い、処理を中止する。正常にファイルが開けた場合、メモリに登録されている全ての名簿データを `sep` で指定された文字で区切りながらファイルに出力する。出力が終わった後、ファイルポインタを閉じる。`void` 型の関数であるため、戻り値はない。

### 3.8 %F コマンド

%F コマンドに対応する `cmd_find(param)` 関数を、301–338 行目で宣言している。%F コマンドは、`param` で指定された文字列に完全一致するデータを含む名簿データを %P コマンドと同様の体裁で出力するコマンドである。まず、コマンドの実行に必要な引数が存在するかを確認し、なければエラーメッセージを表示して処理を中止する。続いて `profile` 型のポインタである `p` に、次に比較を行う名簿データのアドレスを格納する。名簿データの番号は `int` 型の値 `i` で指定し、全ての名簿データを読み終わるまでインクリメントをする。`sprintf()` 関数を使い、`int` 型の値を `char` 型の文字列に変換する。`int` 型の値は約 21 億までの値を格納できるため、これを `char` 型の文字列に直すと 10 文字の文字列になる。文字列であるため、11 文字目に `NULL` 文字を代入しておく必要があるため、余裕を持って `num1` には 12 の長さを確保している。誕生日を格納する `num2`, `num3` も `int` 型の値が 3 つ横並びになるため、余裕を持って 36 の長さを確保している。なお、誕生日は月日が 1 桁の時に、十の位を 0 で埋めた場合も検索できる。その後、`strcmp()` 関数で `param` に格納された文字列と比較を行う。文字列が完全に一致した場合、`strcmp()` 関数の戻り値が 0 となり、`if` 文内の処理が実行されて、該当する名簿データが表示される。`void` 型の関数であるため、戻り値はない。

### 3.9 %S コマンド

%S コマンドに対応する `cmd_sort(param)` 関数を、340–420 行目で宣言している。%S コマンドは、`param` で指定された整数値により、ID、氏名、誕生日、住所、コメントのいずれかの項目で昇順になるように名簿データを整列する。まず、コマンドの実行に必要な引数が存在するかを確認し、なければエラーメッセージを表示して

処理を中止する。int\_value\_check() 関数にて、引数が整数値であることを確認した後、atoi() 関数で整数値に変換し、int 型の変数 a\_buff に代入する。a\_buff の値が 1 以上 5 以下でない場合は、対応する名簿データの要素がないため、エラーメッセージを表示して処理を中止する。ソートのアルゴリズムはバブルソートを用いる [1]。まず、int 型の値 a に a\_buff の値を代入する。この a は、比較する 2 つの要素が完全に一致している時に、1 に書き変わるため、a\_buff として、元の引数を維持している。次に、profile 型の sp1, sp2 に比較をする 2 つの名簿データのアドレスを代入する。今後の処理において見やすさを向上させるため、この処理を行う。本プログラムでは、すべての要素を文字列にて比較する。そのため、名簿データの並び替えが怒る処理は 417 行目にだけ、記述してある。int 型の値である id や birthday 構造体のメンバの要素を文字列に変換するために char 型の配列である num1, num2 をそれぞれ使用する。char 型のポインタである cp1, cp2 は文字列を比較する際に使用され、全ての要素の文字列に対応するために用意している。なお、比較する要素が完全に一致していた場合は、ID に基づいて整列を行うように a に 1 を代入する。389 行目などで %10d という表記を多用しているが、これは数値を文字列に変換した後に strcmp() 関数で比較する際に問題が発生するためである。例えば、521 と 65 という値を文字列に変換して、比較するときに strcmp() 関数は、1 文字目である '5' と '6' を比較してしまうため、65 の方が大きいと認定されてしまうことがある。すべての桁を 0 で埋めることで、strcmp() 関数で ASCII コードを用いて数値の大小関係を考えることができる。なお、負の値を許容できないため、本プログラムでは ID や誕生日で負の値を用いることができないようにしている。void 型の関数であるため、戻り値はない。

### 3.10 独自コマンド

独自コマンドとして、%WS コマンドと %WT コマンドを実装した。これらのコマンドは、cmd\_write() 関数の引数である char 型の値 sep に代入する文字をセミコロンや水平 TAB に変更して実行するコマンドである。名簿や住所録などのデータは、最も一般的であるカンマ区切りの CSV ファイル以外に、セミコロン区切りである SCSV ファイル、水平 TAB 区切りである TSV ファイルなども使われている。このコマンドを用いることにより、セミコロン区切りや水平 TAB 区切りにしたファイルに出力できる。

## 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、CentOS 7.8.2003(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、CentOS 7.8.2003(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]。csv ファイルの内容は、図 1 のような体裁である。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
```

以上のようにして、ファイルを標準入力のリダイレクションで与える。

まず、subst 関数の実行結果について説明する。8.1 節のプログラムに対して、次の内容の input.txt をリダイレクションで与えた場合、

```
Apple
p
a
```

以下のような出力が得られる。

```
str:Aaale
count:2
```

入力データについて説明する。最初の 1 行は、基準の文字列である。2 行目に基準の文字列で置き換えたい 1 文字を入力する。3 行目には、置き換え語の 1 文字を入力する。出力結果では、2 行目で入力した 1 文字が、3 行目で入力した 1 文字が置き換わっている。count の項目は、置き換えた文字数である。

次に、split(), get\_line() 関数の実行結果について説明する。8.2 節のプログラムに対して、次の内容の input.txt をリダイレクションで与えた場合、

```
Microsoft,Windows,7,Ultimate,7601,24545
Apple,iMac,Late,2013,A1418
```

以下のような出力が得られる。

```
line number:1
input:"Microsoft,Windows,7,Ultimate,7601,24545"
split[0]:"Microsoft"
split[1]:"Windows"
split[2]:"7"
split[3]:"Ultimate"
split[4]:"7601"
split[5]:"24545"
```

```
line number:2
input:"Apple,iMac,Late,2013,A1418"
split[0]:"Apple"
split[1]:"iMac"
split[2]:"Late"
split[3]:"2013"
split[4]:"A1418"
```

入力データについて説明する。出力ブロックの最初の 1 行は、CSV ファイルを想定した、カンマで区切られた文字列である。出力結果では、何行目の入力かを示す line number の項目と、実際の入力内容を確認で表示し、3 行目以降にカンマで区切られた文字列が出力される。get\_line() 関数は入力が NULL になると入力処理を終了するように 0 を戻り値として返すので、出力結果は line number が 2 の項目までとなっている。

次に、parse\_line(), new\_profile(), cmd\_check(), cmd\_print(), cmd\_quit() 関数の実行結果について説明する。8.5 節のプログラムに対して、次の内容の input.txt をリダイレクションで与えた場合、

```
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
5100224,Canisbay Primary School,1928-7-5,Canisbay Wick,01955 611337 Primary 56 3.5 Open
%C
%P 2
%P -2
%P 5
%P 0
```



%Q

以下のような出力が得られる.

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

Id : 5100127  
Name : Bower Primary School  
Birth : 1908-01-19  
Addr. : Bowermadden Bower Caithness  
Comm. : 01955 641225 Primary 25 2.6 Open

Id : 5100127  
Name : Bower Primary School  
Birth : 1908-01-19  
Addr. : Bowermadden Bower Caithness  
Comm. : 01955 641225 Primary 25 2.6 Open

Id : 5100224  
Name : Canisbay Primary School  
Birth : 1928-07-05  
Addr. : Canisbay Wick  
Comm. : 01955 611337 Primary 56 3.5 Open

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

Id : 5100127  
Name : Bower Primary School  
Birth : 1908-01-19  
Addr. : Bowermadden Bower Caithness  
Comm. : 01955 641225 Primary 25 2.6 Open

Id : 5100224  
Name : Canisbay Primary School  
Birth : 1928-07-05  
Addr. : Canisbay Wick  
Comm. : 01955 611337 Primary 56 3.5 Open

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

Id : 5100127  
Name : Bower Primary School  
Birth : 1908-01-19  
Addr. : Bowermadden Bower Caithness  
Comm. : 01955 641225 Primary 25 2.6 Open

Id : 5100224  
Name : Canisbay Primary School  
Birth : 1928-07-05  
Addr. : Canisbay Wick  
Comm. : 01955 611337 Primary 56 3.5 Open

入力データについて説明する。出力ブロックの最初の3行は、CSV ファイルを想定した、カンマで区切られた文字列である。以降の行では、% から始まるコマンドの呼び出しを行っている。出力結果では、最初の1行に名簿データの登録数を出力する `cmd_check()` 関数による `3 profile(s)` という内容が出力されている。それ以降のブロックでは、%P 2 という引数を含む `cmd_print()` 関数の呼び出しで、名簿データの初めから2件が表示され、次に %P -2 により、名簿データの後ろから2件を降順に表示している。次の %P 5 は名簿データが3件しか登録されていないため、%P 3 に読み替えて処理が行われる。引数がない場合や0の場合も同様となる。最後に %Q により、プログラムを終了している。

次に、`cmd_read()``cmd_write()`、`cmd_find()`、`cmd_quit("r")` 関数の実行結果について説明する。8.7 節のプログラムに次の内容の `input.txt` をリダイレクションで与えた場合、

```
%R input.csv
%C
%F 5100127
%W output.csv
%Q r
```

なお、`input.csv` の中身は図2に示す。

以下のような出力が得られる。

```
3 profile(s)
Id      : 5100127
Name    : Bower Primary School
Birth   : 1908-01-19
Addr.   : Bowermadden Bower Caithness
Comm.   : 01955 641225 Primary 25 2.6 Open
```

正常終了。

入力データについて説明する。まず、%R コマンドにより、ファイル `input.csv` を読み込む。その次に %C コマンドにより、名簿データの登録数を表示する。次に、%F コマンドにより、5100127 に一致する要素を持つ名簿情報を表示する。そして、%W コマンドにより、ファイル `output.csv` に名簿データを書き込む。最後に、%Q r により、確認メッセージなしでプログラムを終了している。出力結果では、最初の1行に名簿データの登録数を出力する `cmd_check()` 関数による `3 profile(s)` という内容が出力されている。これにより、%R コマンドにより、正常に名簿データが登録されていることが分かる。7行目には、%Q r により呼び出された `cmd_quit()` 関数のメッセージが出力されている。この操作を実行した後、シェル上で `ls` コマンドを使ってファイルを確認すると次のような状態になっている。

```
[user@localhost 10]$ ls
input.csv  input.txt  output.csv  program.out
[user@localhost 10]$ diff -q -s input.csv output.csv
ファイル input.csv と output.csv は同一です
```

```
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
5100224,Canisbay Primary School,1928-7-5,Canisbay Wick,01955 611337 Primary 56 3.5 Open
```

図2 input.csv

diff コマンドを用いて、元の `input.csv` と `output.csv` を比較すると、全く同じ内容になっていることが分かるので、`%W` コマンドが正しく動作していることが分かる。

最後に、`cmd_sort()` 関数の実行結果について説明する。8.7 節のプログラムに次の内容の `input.txt` をリダイレクションで与えた場合、

```
%R input.csv
%S 1
%P
%S 2
%P
%S 3
%P
%S 4
%P
%S 5
%P
%Q r
```

以下のような出力が得られる。

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

```
Id      : 5100127
Name    : Bower Primary School
Birth   : 1908-01-19
Addr.   : Bowermadden Bower Caithness
Comm.   : 01955 641225 Primary 25 2.6 Open
```

```
Id      : 5100224
Name    : Canisbay Primary School
Birth   : 1928-07-05
Addr.   : Canisbay Wick
Comm.   : 01955 611337 Primary 56 3.5 Open
```

```
Id      : 5100127
Name    : Bower Primary School
Birth   : 1908-01-19
Addr.   : Bowermadden Bower Caithness
Comm.   : 01955 641225 Primary 25 2.6 Open
```

```
Id      : 5100224
Name    : Canisbay Primary School
Birth   : 1928-07-05
Addr.   : Canisbay Wick
Comm.   : 01955 611337 Primary 56 3.5 Open
```

```
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
```

```
Id      : 5100127
```

```

Name : Bower Primary School
Birth : 1908-01-19
Addr. : Bowermadden Bower Caithness
Comm. : 01955 641225 Primary 25 2.6 Open

Id : 5100224
Name : Canisbay Primary School
Birth : 1928-07-05
Addr. : Canisbay Wick
Comm. : 01955 611337 Primary 56 3.5 Open

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

Id : 5100127
Name : Bower Primary School
Birth : 1908-01-19
Addr. : Bowermadden Bower Caithness
Comm. : 01955 641225 Primary 25 2.6 Open

Id : 5100224
Name : Canisbay Primary School
Birth : 1928-07-05
Addr. : Canisbay Wick
Comm. : 01955 611337 Primary 56 3.5 Open

Id : 5100224
Name : Canisbay Primary School
Birth : 1928-07-05
Addr. : Canisbay Wick
Comm. : 01955 611337 Primary 56 3.5 Open

Id : 5100127
Name : Bower Primary School
Birth : 1908-01-19
Addr. : Bowermadden Bower Caithness
Comm. : 01955 641225 Primary 25 2.6 Open

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

```

正常終了.

入力データについて説明する。まず，%R コマンドにより，ファイル `input.csv` を読み込む。その次に %S コマンドにより，ID，氏名，誕生日，住所，備考の順に，それぞれ並び変える処理と表示する処理を繰り返す。最後に，%Q r により，確認メッセージなしでプログラムを終了している。出力結果では，18 行毎に指定された要素によるソート後の結果を出力している。初めの 18 行で ID に関する並び替えを行ったもの，その次の 18 行で氏名に関する並び替えを行ったものというように，表示している。全ての要素は ASCII コードの昇順になるように並び替えが行われる。正しく並び替えが行われていることを確認するために，%S 1 により並び替えを行ったものを出力したファイル `output.csv` ものと `sort` コマンドにより並び替えを行ったファイル `sort.txt` を比較した。

```

[user@localhost 10]$ sort input.csv sort.txt
[user@localhost 10]$ diff -q -s output.csv sort.txt
ファイル output.csv と sort.txt は同一です

```

よって、%S コマンドにより、正しく並び替えが行えていることが確認できた。

## 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. CSV データ登録処理の実装
4. %R コマンドと %W コマンド
5. %F コマンド
6. %S コマンド

### 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

C 言語では、配列を関数の戻り値とすることはできないので、ポインタを用いて呼び出し元関数の文字列を参照するか、配列専用の構造体を用意する必要がある。例えば、関数の宣言時に構造体名、関数名と書き、構造体のメンバに `char a[70]` などを指定しておくと、70 文字までの要素を戻り値として呼び出し元に返すことも可能である（8.8 節に実装例を示す。）。ただ、二重に代入処理を行うことになるため、扱うデータ量が多くなると効率的ではない。そのため、`subst()` 関数が、他関数から文字列の配列の先頭アドレス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、入力文字列の途中で 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ずに等しい。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をしている。`split()` 関数の実装に当たっては、呼び出し元関数で `char` 型の二次元配列を作り、そこに文字列情報を複写する方法を考えたが、配列である以上、仮に文字が代入されなくてもメモリを使用してしまうので、メモリを浪費する。メモリを浪費しないために、ポインタと文字列の先頭アドレスのセットで情報を管理することにした。ただ、この方法は、元の入力文字列の情報を破壊するため、場合によっては使用できない。元の入力文字列情報を残す場合は、予め別の変数に文字列をコピーしておく必要がある。

### 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、%Q コマンドが未実装の場合、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、デバッグのときの終了が不便である。デバッグをより早く行うために、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（8.4 節のソースコード。8.5 節以降のソースコードでは、`cmd_quit()` 関数の呼び出しに機能を変更。）。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から `fgets()` 関数により取得された文字列は、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。1 文字の代入処理であるから、手動で最後のみ NULL 文字を代入できなくもないが、入力文字列が常に 1024 文字とは限らないため、改行

文字が配列上のどこに来るかは定かではない。そのため、ここでは `subst()` 関数を用いる。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。プログラムの動作を `db-sample` に合わせるため、プログラミング演習 1 では対策を見送った。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()` 関数は引数が `NULL` の場合コアダンプが起こるので、引数が `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、住所への文字列の代入では、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。`fgets()` 関数同様、`strncpy()` 関数の文字数には、`NULL` 文字はカウントされていないので、引数は 69 とした。備考は、文字数が任意であるため、予め配列などで文字列を長を決めてしまうと、メモリを浪費してしまうため、備考情報の文字列長を `strlen()` 関数で取得し、それに `NULL` 文字分の 1 を加えた分のメモリを `malloc()` 関数で新たに確保し、その先頭アドレスを構造体のメンバである `*biko` に代入することで、メモリの浪費を防止できる。

### 5.4 「%R コマンドと %W コマンド」に関する考察

%R コマンドの実装にあたり、グローバル変数として `int` 型の変数 `flag` とファイルポインタ `fp` を用意することで、`get_line()` 関数とのデータのやり取りを可能にする方法を考えた。`get_line()` 関数内で `flag` を用いた `if` 文による分岐で、`stdin` か `fp` を読み取り元にするか指定する。しかし、%R コマンドにより読み取られる 1 行分の入力には `parse_line()` 関数を経由するため、その入力に % で始まる入力があればコマンド入力として処理が行われる。ここでもう一度 %R コマンドを呼び出してしまうと、既に開いているグローバル変数上のファイルポインタを上書きしてしまい、意図しない動作を引き起こす。ファイルポインタの上書きを避けるために、今回のプログラムでは、ローカル変数としてファイルポインタを用意し、`get_line()` 関数の引数を増やした。

%W コマンドの実装にあたり、全体のメモリ使用量を削減するために、%R コマンドでも使用しているグローバル変数のファイルポインタ `fp` を利用することを検討したが、これも先の例と同様で、%R コマンドで読み込んだ入力行からコマンドを実行できるため、ファイルポインタを上書きする恐れがある。そのため、同様にローカル変数としてファイルポインタを用意した。

### 5.5 「%F コマンド」に関する考察

%F コマンドの実装にあたり、引数である `param` を `int` 型の値に変換可能であれば、`atoi()` 関数で変換して別の変数に代入する方法を考えたが、一致する要素を探す `if` 文の分岐が複数になったり、誕生日の要素の比較がしにくいなどの問題があり、すべて `char` 型の文字列に変換して、`strcmp()` 関数で比較することにした。これにより、比較の `if` 文は一つだけになった。

## 5.6 「%S コマンド」に関する考察

# 6 発展課題

## 6.1 エラー処理についての考察

## 6.2 コマンド入力仕様の拡張

## 6.3 既存機能や既存コマンドの拡張

# 7 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わかった。 `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。アドレスなのか値なのか、今後プログラムを書くときに注意しておきたい。構造体のメンバに他の構造体を含む、構造体の宣言は今回の演習で初めて行ったが、ポインタを用いた構造体のメンバの指定の仕方が、 `(profile_p->birthday).y` となり、 `(profile_p->birthday)->y` という書き方ができないのが何故なのか疑問点のままで。プログラム中では `profile_p` は、ポインタであり、ポインタの指し先のメンバとなる `birthday` との関係は、ポインタとその指し先のメンバだが、 `birthday` とそのメンバである `y` との関係は、構造体とそのメンバの関係となるので、“.” でつながることができるのかもしれない。8.2 節のプログラムにおける `split()` 関数の作成で、配列の添字に合わせて、戻り値が分けられた個数より 1 小さくなっていたのを `main()` 関数のループの条件で矯正していたが、これが最終的にプログラムの仕様を満たさなくなっていたということがあり、プログラミング演習 1 の最後で行った基本関数のテストで NG となった。このため、最終的にソースコードの修正を行った。修正後の最終プログラムは 8.6 節に記載している。 `split()` 関数内で起きたトラブルを他関数側の処理で矯正するのは避けようと思った。

# 8 作成したプログラム

作成したプログラムリストを以下に添付する。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。

## 8.1 プログラミング演習 1 第 1 回講義のプログラム

`subst()` 関数を加えたプログラムのソースコード (41 行)

```
1      #include <stdio.h>
2
3      int subst(char *str, char c1, char c2)
4      {
5          int i;
6          int c = 0;
```

```

7      for(i = 0; *(str + i) != '\0'; i++)
8      {
9          if(c1 == c2) break;
10         if(*(str + i) == c1)
11         {
12             *(str + i) = c2;
13             c++;
14         }
15     }
16     return c;
17 }
18
19 int main(void)
20 {
21     char str[100] = {0};
22     char c1 = 0;
23     char c2 = 0;
24     char dummy;
25     int c = 0;
26
27     printf("Input str.\n");
28     scanf("%s", str);
29     printf("Input c1.\n");
30     scanf("%c", &dummy);
31     scanf("%c", &c1);
32     printf("Input c2.\n");
33     scanf("%c", &dummy);
34     scanf("%c", &c2);
35
36     c = subst(str, c1, c2);
37
38     printf("\nstr:%s\ncount:%d\n", str, c);
39
40     return 0;
41 }

```

## 8.2 プログラミング演習 1 第 2 回講義のプログラム

subst(), get\_line() 関数を加えたプログラムのソースコード (70 行)

```

1      #include <stdio.h>
2
3      #define ESC 27                                /*文字列 ESC を ESC の ASCII
コードで置換*/
4      #define MAX_LINE 1025                          /*文字配列 LINE の
最大入力数の指定用*/
5
6      int subst(char *str, char c1, char c2)
7      {
8          int i;                                    /*for ループ用*/
9          int c = 0;                                /*置き換えた文字数の
カウント用*/
10         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*/
11         {
12             if(c1 == c2) break;                    /*見た目上文字列に変化が
ないとき*/
13             if(*(str + i) == c1)                  /*(str + i) の文字が c1 の
文字と同じとき*/
14             {
15                 *(str + i) = c2;                  /*(str + i) の文字を c2 の
文字に置き換える*/
16                 c++;                                /*置き換えた文字を数える*/
17             }
18         }
19         return c;                                /*置き換えた文字数を戻り値

```



```

とする。*/
20     }
21
22     int split(char *str, char *ret[], char sep, int max)
23     {
24         int i;                                /*for ループ用*/
25         int c = 0;                            /*ポインタの配列の指定用*/
26
27         ret[0] = str;                        /*ret[0] に str の先頭アドレス
を代入*/
28
29         for(i = 0; *(str + i) != '\0' && c < max; i++)    /*c が max より小さいか、入力
文字列の終端に辿り着いていないときループ*/
30         {
31             if(*(str + i) == sep)            /*(str + i) が sep のとき*/
32             {
33                 *(str + i) = '\0';          /*(str + i) に NULL を代入*/
34                 c++;
35                 ret[c] = str + (i + 1);      /*ret[c] に NULL 文字の"次の"
アドレスを代入*/
36             }
37         }
38         return c;                            /*文字列をいくつに分割したか
を戻り値とする*/
39     }
40
41     int get_line(char *line)
42     {
43         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*/
44         if(*line == ESC) return 0;            /*直接入力するとき、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*/
45         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*/
46         return 1;                            /*入力文字列が存在したとき、
1 を戻り値とする*/
47     }
48
49     int main(void)
50     {
51         char line[MAX_LINE] = {0};            /*入力文字列は最大 1024 文字*/
52         char *ret[10];
53         char sep = ',';                      /*csv ファイルからの入力を
想定しているため、カンマ*/
54         int max = 10;
55         int c, i, a = 1;
56
57         while(get_line(line))                /*get_line 関数を呼び出し、
戻り値が 0 ならループを終了*/
58         {
59             printf("line number:%d\n", a);
60             printf("input: \"%s\"\n", line);
61             c = split(line, ret, sep, max);    /*split 関数を呼び出す*/
62             for(i = 0; i <= c; i++)
63             {
64                 printf("split[%d]: \"%s\"\n", i, ret[i]);
65             }
66             printf("\n\n");                  /*見やすさのために改行*/
67             a++;
68         }
69         return 0;
70     }

```

### 8.3 プログラミング演習 1 第 3 回講義のプログラム

parse\_line(), exec\_command 関数を加えたプログラムのソースコード (169 行)

```

1      #include <stdio.h>
2      #include <stdlib.h>
3
4      #define ESC 27
コードで置換*/
5      #define MAX_LINE 1025
の指定用*/
6
7      /*関数プロトタイプ宣言（煩雑化防止）*/
8      int subst(char *str, char c1, char c2);
9      int split(char *str, char *ret[], char sep, int max);
10     int get_line(char *line);
11     void parse_line(char *line);
12     void exec_command(char cmd, char *param);
13     void cmd_quit();
14     void cmd_check();
15     void cmd_print();
16     void cmd_read();
17     void cmd_write();
18     void cmd_find();
19     void cmd_sort();
20     void new_profile(char *line);
21
22     int subst(char *str, char c1, char c2)
23     {
24         int i;
25         int c = 0;
ント用*/
26         for(i = 0; *(str + i) != '\0'; i++)
着くまでループ*/
27         {
28             if(c1 == c2) break;
29             if(*(str + i) == c1)
文字と同じとき*/
30             {
31                 *(str + i) = c2;
文字に置き換える*/
32                 c++;
33             }
34         }
35         return c;
とする. */
36     }
37
38     int split(char *str, char *ret[], char sep, int max)
39     {
40         int i;
41         int c = 0;
42
43         ret[0] = str;
スを代入*/
44
45         for(i = 0; *(str + i) != '\0' && c < max; i++)
文字列の終端に辿り着いていないときループ*/
46         {
47             if(*(str + i) == sep)
48             {
49                 *(str + i) = '\0';
50                 c++;
51                 ret[c] = str + (i + 1);
アドレスを代入*/
52             }
53         }
54         return c;
を戻り値とする*/
55     }
56

```

```

/*exit 関数用*/

/*文字列 ESC を ESC の ASCII

/*文字配列 LINE の最大入力数

/*for ループ用*/
/*置き換えた文字数のカウ

/*入力文字列の終端に辿り

/*見た目上文字列に変化が

/*(str + i) の文字が c1 の

/*(str + i) の文字を c2 の

/*置き換えた文字を数える*/

/*置き換えた文字数を戻り値

/*for ループ用*/
/*ポインタの配列の指定用*/

/*ret[0] に str の先頭アドレ

/*c が max より小さいかつ入力

/*(str + i) が sep のとき*/

/*(str + i) に NULL を代入*/

/*ret[c] に NULL 文字の"次の"

/*文字列をいくつに分割したか

```

```

57     int get_line(char *line)
58     {
59         if(fgets(line, MAX_LINE, stdin) == NULL) return 0;
60         if(*line == ESC) return 0;
61         subst(line, '\n', '\0');
62         return 1;
63     }
64
65     void parse_line(char *line)
66     {
67         char cmd;
68         char *param;
69         char *buffer = "(Null Parameter)";
70
71         if(*line == '%')
72         {
73             cmd = *(line + 1);
74             if(*(line + 3) != '\0') param = (line + 3);
75             else param = buffer;
76             exec_command(cmd, param);
77         }
78         else
79         {
80             new_profile(line);
81         }
82     }
83
84     void exec_command(char cmd, char *param)
85     {
86         switch (cmd) {
87             case 'Q': cmd_quit(); break;
88             case 'T': printf("Parameter test: \"%s\"\n", param); break;
89             case 'C': cmd_check(); break;
90             case 'P': cmd_print(); break;
91             case 'R': cmd_read(); break;
92             case 'W': cmd_write(); break;
93             case 'F': cmd_find(); break;
94             case 'S': cmd_sort(); break;
95             default: fprintf(stderr, "%%%c command is invoked with arg: \"%s\"\n", cmd, param);
96         }
97     }
98
99     void cmd_quit()
100    {
101        printf("Do you want to quit?(y/n)\n");
102        if(getchar() == 'y')
103        {
104            printf("quit success.\n");
105            exit(0);
106        }
107        getchar();
108        printf("quit cancelled.\n");
109    }
110
111    void cmd_check()
112    {
113        fprintf(stderr, "Check command is invoked.\n");

```

/\*入力文字列が空のとき、0を  
 /\*ESCを1文字目に入力すること  
 /\*subst関数により、入力の  
 /\*入力文字列が存在したとき、  
 /\*%の次の1文字を格納用\*/  
 /\*コマンドのパラメータとなる  
 /\*例外処理用\*/  
 /\*入力文字列の1文字目が%のとき\*/  
 /\*cmdに入力文字列の2文字目  
 /\*ポインタlineに3を足した  
 /\*入力文字列にパラメータ部が  
 /\*入力がコマンドではないとき\*/  
 /\*ポインタparamの参照先  
 /\*確認メッセージ\*/  
 /\*getcharでの入力時に改行文字が

```

114     }
115
116     void cmd_print()
117     {
118         fprintf(stderr, "Print command is invoked.\n");
119     }
120
121     void cmd_read()
122     {
123         fprintf(stderr, "Read command is invoked.\n");
124     }
125
126     void cmd_write()
127     {
128         fprintf(stderr, "Write command is invoked.\n");
129     }
130
131     void cmd_find()
132     {
133         fprintf(stderr, "Find command is invoked.\n");
134     }
135
136     void cmd_sort()
137     {
138         fprintf(stderr, "Sort command is invoked.\n");
139     }
140
141     void new_profile(char *line)
142     {
143         char *ret[11];
144         char sep = ',';
145         int max = 10;
146         int c, i;
147         static int a = 1;
148
149         printf("line number:%d\n", a);
150         printf("input: \"%s\"\n", line);
151         c = split(line, ret, sep, max);
152         for(i = 0; i <= c; i++)
153         {
154             printf("split[%d]: \"%s\"\n", i, ret[i]);
155         }
156         printf("\n");
157         a++;
158     }
159
160     int main(void)
161     {
162         char LINE[MAX_LINE] = {0};
163
164         while(get_line(LINE))
165         {
166             parse_line(LINE);
167         }
168         return 0;
169     }

```

いるため、カンマ\*/  
 /\*csv ファイルからの入力を想定して  
 /\*値を main 関数終了時まで保持する  
 必要があるため、static int 型\*/  
 /\*split 関数を呼び出す\*/  
 /\*見やすさのために改行\*/  
 /\*入力文字列 (1 行分) は main 関数で  
 管理\*/  
 /\*文字配列 LINE に文字列を入力する  
 (get\_line 関数)\*/  
 /\*入力文字列がある場合、構文解析  
 を行う (parse\_line 関数)\*/

## 8.4 プログラミング演習 1 第 4 回講義のプログラム

profile, date 構造体と new\_profile() 関数を加えたプログラムのソースコード (239 行)

```

1      #include <stdio.h>
2      #include <string.h>                                /*strncpy 関数等用*/
3      #include <stdlib.h>                                /*exit 関数用*/
4
5      #define ESC 27                                     /*文字列 ESC を ESC
の ASCII コードで置換*/
6      #define MAX_LINE 1025                             /*文字配列 LINE の
最大入力数の指定用*/
7
8      /*構造体宣言*/
9      struct date
10     {
11         int y; /*年*/
12         int m; /*月*/
13         int d; /*日*/
14     };
15
16     struct profile
17     {
18         int id;          /*ID*/
19         char name[70];    /*名前*/
20         struct date birthday; /*誕生日 (date 構造体)*/
21         char address[70]; /*住所*/
22         char biko[70];    /*備考*/
23     };
24
25     /*関数プロトタイプ宣言 (煩雑化防止) */
26     int subst(char *str, char c1, char c2);
27     int split(char *str, char *ret[], char sep, int max);
28     int get_line(char *line);
29     void parse_line(char *line);
30     void exec_command(char cmd, char *param);
31     void cmd_quit();
32     void cmd_check();
33     void cmd_print();
34     void cmd_read();
35     void cmd_write();
36     void cmd_find();
37     void cmd_sort();
38     void new_profile(struct profile *profile_p, char *line);
39
40     /*グローバル変数宣言*/
41     struct profile profile_data_store[10000];           /*profile 情報を
格納*/
42     int profile_data_nitems = 0;                       /*profile 情報の
保存数を格納*/
43
44     int subst(char *str, char c1, char c2)
45     {
46         int i;                                         /*for ループ用*/
47         int c = 0;                                     /*置き換えた文字
数のカウント用*/
48         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*/
49             {
50                 if(c1 == c2) break;                   /*見た目上文字列
に変化がないとき*/
51                 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;            /*直接入力の際、
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て、NULL まで文字列を表示する (デバッグ用) */
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; /*エラーメッセージを表示*/
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ます, 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     }

```

## 8.5 プログラミング演習 1 第 5 回講義のプログラム

本プログラムは、UNIX 用 db-sample に合わせた仕様となっている。また、プログラムの一部機能を//で無効化している。

C, P コマンドを加えたプログラムのソースコード (248 行)

```

1         #include <stdio.h>
2         #include <string.h>                                       /*strncpy 関数等用*/
3         #include <stdlib.h>                                       /*exit 関数用*/
4
5         #define ESC 27                                           /*文字列 ESC を ESC
の ASCII コードで置換*/
6         #define MAX_LINE 1025                                    /*文字配列 LINE の
最大入力数の指定用*/
7
8         /*構造体宣言*/
9         struct date
10        {
11            int y; /*年*/
12            int m; /*月*/
13            int d; /*日*/
14        };
15
16        struct profile
17        {
18            int id;          /*ID*/
19            char name[70];    /*名前*/
20            struct date birthday; /*誕生日 (date 構造体)*/
21            char address[70]; /*住所*/
22            char *biko;      /*備考*/
23        };
24
25        /*関数プロトタイプ宣言（煩雑化防止）*/
26        int subst(char *str, char c1, char c2);
27        int split(char *str, char *ret[], char sep, int max);
28        int get_line(char *line);
29        void parse_line(char *line);
30        void exec_command(char cmd, char *param);
31        void cmd_quit(void);
32        void cmd_check(void);
33        void cmd_print(char *param);
34        void cmd_read();

```

```

35     void cmd_write();
36     void cmd_find();
37     void cmd_sort();
38     void new_profile(struct profile *profile_p, char *line);
39
40     /*グローバル変数宣言*/
41     struct profile profile_data_store[10000];           /*profile 情報を
格納*/
42     int profile_data_nitems = 0;                       /*profile 情報の
保存数を格納*/
43
44     int subst(char *str, char c1, char c2)
45     {
46         int i;                                         /*for ループ用*/
47         int c = 0;                                     /*置き換えた文字
数のカウント用*/
48         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*/
49             {
50                 if(c1 == c2) break;                   /*見た目上文字列
に変化がないとき*/
51                 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) cmd_quit();                  /*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
92         if(*line == '%')                        /*入力文字列
の 1 文字目が % のとき*/
93         {
94             cmd = *(line + 1);                  /*cmd に入力
文字列の 2 文字目の値を代入*/
95             param = line + 3;                  /*param にパラメータ
部を代入*/
96             exec_command(cmd, param);
97         }
98         else                                    /*入力がコマンド
ではないとき*/
99         {
100             new_profile(&profile_data_store[profile_data_nitems++] ,line);
101         }
102     }
103
104     void exec_command(char cmd, char *param)
105     {
106         switch (cmd) {
107             case 'Q': cmd_quit(); break;
108             //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て、NULL まで文字列を表示する (デバッグ用) */
109             case 'C': cmd_check(); break;
110             case 'P': cmd_print(param); break;
111             case 'R': cmd_read(); break;
112             case 'W': cmd_write(); break;
113             case 'F': cmd_find(); break;
114             case 'S': cmd_sort(); break;
115             default: fprintf(stderr, "Invalid command %c: ignored.\n", cmd); break; /*エラー
メッセージを表示*/
116         }
117     }
118
119     void cmd_quit()
120     {
121         //char c;
122
123         //while(1)
124         //{
125             //printf("Do you want to quit?(y/n)\n");          /*確認メッセージ*/
126             //c = getchar();
127             //getchar();                                        /*getchar での入力
時に改行文字が残ってしまうため*/
128             //if(c == 'y')
129             //{
130                 //printf("quit success.\n\n");
131                 exit(0);
132             //}
133             //else if(c == 'n')
134             //{
135                 //printf("quit cancelled.\n\n");
136                 //break;
137             //}
138             //}
139     }
140
141     void cmd_check(void)
142     {

```

```

143     printf("%d profile(s)\n", profile_data_nitems);
144 }
145
146 void cmd_print(char *param)
147 {
148     int a = 0;
149     int i = 0;                                /*for ループ用*/
150
151     a = atoi(param);                          /*文字列を int 型の値に変換*/
152
153     /*a の絶対値が profile_data_nitems より大きいときか a=0 のとき*/
154     if(abs(a) >= profile_data_nitems || a == 0) a = profile_data_nitems;
155
156     if(a > 0)                                  /*引数が正の整数のとき 及び例外*/
157     {
158         for(i = 0; i < a; i++)
159         {
160             printf("Id      : %d\n", profile_data_store[i].id);
161             printf("Name    : %s\n", profile_data_store[i].name);
162             printf("Birth   : %04d-%02d-%02d\n", profile_data_store[i].birthday.y, profile_
data_store[i].birthday.m, profile_data_store[i].birthday.d);
163             printf("Addr.   : %s\n", profile_data_store[i].address);
164             printf("Comm.   : %s\n\n", profile_data_store[i].biko);
165         }
166     }
167     else if(a < 0)                             /*引数が負の整数のとき*/
168     {
169         for(i = profile_data_nitems + a; i < profile_data_nitems; i++)
170         {
171             printf("Id      : %d\n", profile_data_store[i].id);
172             printf("Name    : %s\n", profile_data_store[i].name);
173             printf("Birth   : %04d-%02d-%02d\n", profile_data_store[i].birthday.y, profile_
data_store[i].birthday.m, profile_data_store[i].birthday.d);
174             printf("Addr.   : %s\n", profile_data_store[i].address);
175             printf("Comm.   : %s\n\n", profile_data_store[i].biko);
176         }
177     }
178 }
179
180 void cmd_read()
181 {
182     fprintf(stderr, "Read command is invoked.\n");
183 }
184
185 void cmd_write()
186 {
187     fprintf(stderr, "Write command is invoked.\n");
188 }
189 void cmd_find()
190 {
191     fprintf(stderr, "Find command is invoked.\n");
192 }
193
194 void cmd_sort()
195 {
196     fprintf(stderr, "Sort command is invoked.\n");
197 }
198
199 void new_profile(struct profile *profile_p, char *line)
200 {
201     char *ret[10];
202     char *ret2[4];                                /*誕生日の情報
を分割し、その先頭アドレスを保存*/
203     char sep = ',';                                /*csv ファイル
からの入力を想定しているため、カンマ*/
204     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*/
205     int max = 10;

```

```

206         int max2 = 4;
207         int c, birth_c;
208         int MAX_BIKO = 0;                                /*備考の文字数
カウント用*/
209
210         c = split(line, ret, sep, max);                    /*ID, 名前などの
情報を分割する*/
211         if(c != 4)                                         /*入力形式が合わ
ない場合*/
212         {
213             fprintf(stderr, "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ます. \n 処理を中止しました. \n\n");
214             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*/
215             return;
216         }
217         birth_c = split(ret[2], ret2, sep2, max2);          /*誕生日の年, 月,
日を分割する*/
218         if(birth_c != 2)                                  /*誕生日の年, 月,
日を正常に分割できない場合*/
219         {
220             fprintf(stderr, "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*/
221             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*/
222             return;
223         }
224
225         /*構造体への情報の書き込み処理*/
226         profile_p->id = atoi(ret[0]);                      /*ID の書き込み*/
227         strncpy(profile_p->name, ret[1], 69);              /*名前の書き込み*/
228         (profile_p->birthday).y = atoi(ret2[0]);          /*誕生年の書き込み*/
229         (profile_p->birthday).m = atoi(ret2[1]);          /*誕生月の書き込み*/
230         (profile_p->birthday).d = atoi(ret2[2]);          /*誕生日の書き込み*/
231         strncpy(profile_p->address, ret[3], 69);          /*住所の書き込み*/
232
233         MAX_BIKO = strlen(ret[4]) + 1;                    /*備考情報の文字数
のカウンタ*/
234
235         profile_p->biko = (char *)malloc(sizeof(char) * MAX_BIKO); /*文字数分だけ
メモリ確保*/
236         strncpy(profile_p->biko, ret[4], MAX_BIKO);        /*備考の書き込み*/
237     }
238
239     int main(void)
240     {
241         char LINE[MAX_LINE] = {0};                        /*入力文字列 (1 行分) は
main 関数で管理*/
242
243         while(get_line(LINE))                              /*文字配列 LINE に文字列
を入力する*/
244         {
245             parse_line(LINE);                              /*入力文字列がある場合,
構文解析を行う*/
246         }
247         return 0;
248     }

```

## 8.6 プログラミング演習 1 最終プログラム

プログラミング演習 1 の最後の基本関数のテストで `split()` 関数が NG となったため, ソースコードに修正を加えた.

`split()` 関数に修正を加えたプログラムのソースコード (247 行)

```

1         #include <stdio.h>

```

```

2      #include <string.h>                                /*strncpy 関数等用*/
3      #include <stdlib.h>                                /*exit 関数用*/
4
5      #define ESC 27                                     /*文字列 ESC を ESC
の ASCII コードで置換*/
6      #define MAX_LINE 1025                             /*文字配列 LINE の
最大入力数の指定用*/
7
8      /*構造体宣言*/
9      struct date
10     {
11         int y; /*年*/
12         int m; /*月*/
13         int d; /*日*/
14     };
15
16     struct profile
17     {
18         int id; /*ID*/
19         char name[70]; /*名前*/
20         struct date birthday; /*誕生日 (date 構造体)*/
21         char address[70]; /*住所*/
22         char *biko; /*備考*/
23     };
24
25     /*関数プロトタイプ宣言 (煩雑化防止) */
26     int subst(char *str, char c1, char c2);
27     int split(char *str, char *ret[], char sep, int max);
28     int get_line(char *line);
29     void parse_line(char *line);
30     void exec_command(char cmd, char *param);
31     void cmd_quit(void);
32     void cmd_check(void);
33     void cmd_print(char *param);
34     void cmd_read();
35     void cmd_write();
36     void cmd_find();
37     void cmd_sort();
38     void new_profile(struct profile *profile_p, char *line);
39
40     /*グローバル変数宣言*/
41     struct profile profile_data_store[10000];            /*profile 情報を
格納*/
42     int profile_data_nitems = 0;                        /*profile 情報の
保存数を格納*/
43
44     int subst(char *str, char c1, char c2)
45     {
46         int i; /*for ループ用*/
47         int c = 0; /*置き換えた文字
数のカウント用*/
48         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*/
49             {
50                 if(c1 == c2) break;                    /*見た目上文字列
に変化がないとき*/
51                 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[c++] = 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                 ret[c++] = str + (i + 1);    /*ret[c] に NULL
文字の"次の"アドレスを代入*/
73             }
74         }
75         return c;                            /*文字列をいく
つに分割したかを戻り値とする*/
76     }
77
78     int get_line(char *line)
79     {
80         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*/
81         if(*line == ESC) cmd_quit();
82         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*/
83         return 1;                            /*入力文字列が
存在したとき, 1 を戻り値とする*/
84     }
85
86     void parse_line(char *line)
87     {
88         char cmd;                             /*% の次の 1 文字
を格納用*/
89         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*/
90
91         if(*line == '%')                    /*入力文字列
の 1 文字目が % のとき*/
92         {
93             cmd = *(line + 1);               /*cmd に入力
文字列の 2 文字目の値を代入*/
94             param = line + 3;                /*param にパラメータ
部を代入*/
95             exec_command(cmd, param);
96         }
97         else                                /*入力がコマンド
ではないとき*/
98         {
99             new_profile(&profile_data_store[profile_data_nitems++] ,line);
100         }
101     }
102
103     void exec_command(char cmd, char *param)
104     {
105         switch (cmd) {
106             case 'Q': cmd_quit(); break;
107             //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て, NULL まで文字列を表示する (デバッグ用) */
108             case 'C': cmd_check(); break;
109             case 'P': cmd_print(param); break;

```

```

110         case 'R': cmd_read();    break;
111         case 'W': cmd_write();   break;
112         case 'F': cmd_find();    break;
113         case 'S': cmd_sort();    break;
114         default: fprintf(stderr, "Invalid command %c: ignored.\n", cmd); break; /*エラー
メッセージを表示*/
115     }
116 }
117
118 void cmd_quit()
119 {
120     //char c;
121
122     //while(1)
123     //{
124     //printf("Do you want to quit?(y/n)\n");           /*確認メッセージ*/
125     //c = getchar();
126     //getchar();                                       /*getchar での入力
時に改行文字が残ってしまうため*/
127     //if(c == 'y')
128     //{
129     //printf("quit success.\n\n");
130     //    exit(0);
131     //}
132     //else if(c == 'n')
133     //{
134     //printf("quit cancelled.\n\n");
135     //break;
136     //}
137     //}
138 }
139
140 void cmd_check(void)
141 {
142     printf("%d profile(s)\n", profile_data_nitems);
143 }
144
145 void cmd_print(char *param)
146 {
147     int a = 0;
148     int i = 0;                                       /*for ループ用*/
149
150     a = atoi(param);                                /*文字列を int 型の値に変換*/
151
152     /*a の絶対値が profile_data_nitems より大きいときか a=0 のとき*/
153     if(abs(a) >= profile_data_nitems || a == 0) a = profile_data_nitems;
154
155     if(a > 0)                                         /*引数が正の整数のとき及び例外*/
156     {
157         for(i = 0; i < a; i++)
158         {
159             printf("Id      : %d\n", profile_data_store[i].id);
160             printf("Name    : %s\n", profile_data_store[i].name);
161             printf("Birth   : %04d-%02d-%02d\n", profile_data_store[i].birthday.y, profile_
data_store[i].birthday.m, profile_data_store[i].birthday.d);
162             printf("Addr.   : %s\n", profile_data_store[i].address);
163             printf("Comm.   : %s\n\n", profile_data_store[i].biko);
164         }
165     }
166     else if(a < 0)                                   /*引数が負の整数のとき*/
167     {
168         for(i = profile_data_nitems + a; i < profile_data_nitems; i++)
169         {
170             printf("Id      : %d\n", profile_data_store[i].id);
171             printf("Name    : %s\n", profile_data_store[i].name);
172             printf("Birth   : %04d-%02d-%02d\n", profile_data_store[i].birthday.y, profile_
data_store[i].birthday.m, profile_data_store[i].birthday.d);
173             printf("Addr.   : %s\n", profile_data_store[i].address);

```



```

174         printf("Comm. : %s\n\n",profile_data_store[i].biko);
175     }
176 }
177 }
178
179 void cmd_read()
180 {
181     fprintf(stderr, "Read command is invoked.\n");
182 }
183
184 void cmd_write()
185 {
186     fprintf(stderr, "Write command is invoked.\n");
187 }
188 void cmd_find()
189 {
190     fprintf(stderr, "Find command is invoked.\n");
191 }
192
193 void cmd_sort()
194 {
195     fprintf(stderr, "Sort command is invoked.\n");
196 }
197
198 void new_profile(struct profile *profile_p, char *line)
199 {
200     char *ret[10];
201     char *ret2[4];
202     char sep = ',';
203     char sep2 = '-';
204     int max = 10;
205     int max2 = 4;
206     int c, birth_c;
207     int MAX_BIKO = 0;
208
209     c = split(line, ret, sep, max);
210     if(c != 5)
211     {
212         fprintf(stderr, "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ます. \n 処理を中止しました. \n\n");
213         profile_data_nitems--;
214         return;
215     }
216     birth_c = split(ret[2], ret2, sep2, max2);
217     if(birth_c != 3)
218     {
219         fprintf(stderr, "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*/
220         profile_data_nitems--;
221         return;
222     }
223
224     /*構造体への情報の書き込み処理*/
225     profile_p->id = atoi(ret[0]);
226     strncpy(profile_p->name, ret[1], 69);
227     (profile_p->birthday).y = atoi(ret2[0]);
228     (profile_p->birthday).m = atoi(ret2[1]);
229     (profile_p->birthday).d = atoi(ret2[2]);

```

/\*誕生日の情報  
 /\*csv ファイル  
 /\*誕生日の入力  
 /\*備考の文字数  
 /\*ID, 名前などの  
 /\*入力形式が合わ  
 /\*処理中止により,  
 /\*誕生日の年, 月,  
 /\*誕生日の年, 月,  
 /\*処理中止により,  
 /\*ID の書き込み\*/  
 /\*名前の書き込み\*/  
 /\*誕生年の書き込み\*/  
 /\*誕生月の書き込み\*/  
 /\*誕生日の書き込み\*/

```

230         strncpy(profile_p->address, ret[3], 69);           /*住所の書き込み*/
231
232         MAX_BIKO = strlen(ret[4]) + 1;                       /*備考情報の文字数
のカウンタ*/
233
234         profile_p->biko = (char *)malloc(sizeof(char) * MAX_BIKO); /*文字数分だけ
メモリ確保*/
235         strncpy(profile_p->biko, ret[4], MAX_BIKO);           /*備考の書き込み*/
236     }
237
238     int main(void)
239     {
240         char LINE[MAX_LINE] = {0};                           /*入力文字列（1行分）は
main関数で管理*/
241
242         while(get_line(LINE))                                 /*文字配列 LINE に文字列
を入力する*/
243         {
244             parse_line(LINE);                                 /*入力文字列がある場合,
構文解析を行う*/
245         }
246         return 0;
247     }

```

## 8.7 プログラミング演習 2 最終プログラム

プログラムのソースコード (627 行)

```

1     #include <stdio.h>
2     #include <string.h>                                     /*strncpy 関数等用*/
3     #include <stdlib.h>                                     /*exit 関数用*/
4
5     #define ESC 27                                          /*文字列 ESC を ESC の ASCII コードで置換
*/
6     #define TAB 9                                          /*文字列 TAB を TAB の ASCII コードで置換
*/
7     #define MAX_LINE 1025                                 /*文字配列 LINE の最大入力数の指定用*
/
8
9     /*構造体宣言*/
10    struct date
11    {
12        int y; /*年*/
13        int m; /*月*/
14        int d; /*日*/
15    };
16
17    struct profile
18    {
19        int id; /*ID*/
20        char name[70]; /*名前*/
21        struct date birthday; /*誕生日 (date 構造体)*/
22        char address[70]; /*住所*/
23        char *biko; /*備考*/
24    };
25
26    /*関数プロトタイプ宣言 (煩雑化防止) */
27    int subst(char *str, char c1, char c2);
28    int split(char *str, char *ret[], char sep, int max);
29    int get_line(FILE *F, char *line);
30    void parse_line(char *line);
31    void exec_command(char *cmd, char *param);
32    void cmd_quit(char *param);
33    void cmd_check(void);
34    void cmd_print(char *param);
35    void cmd_read(char *param);

```

```

36 void cmd_write(char *param, char sep);
37 void cmd_find(char *param);
38 void cmd_sort(char *param);
39 void data_move(struct profile *sp1, struct profile *sp2);
40 void new_profile(struct profile *profile_p, char *line);
41 int int_value_check(char *str);
42 char input_format_check(char *str);
43 int day_format_check(int y, int m, int d);
44
45 /*グローバル変数宣言*/
46 struct profile profile_data_store[10000]; /*profile 情報を格納*/
47 int profile_data_nitems = 0; /*profile 情報の保存数を格納*/
48
49 int subst(char *str, char c1, char c2)
50 {
51     int i; /*for ループ用*/
52     int c = 0; /*置き換えた文字数のカウント用*/
53     for(i = 0; *(str + i) != '\0'; i++) /*入力文字列の終端に辿り着くまでル
一プ*/
54     {
55         if(c1 == c2) break; /*見た目上文字列に変化がないとき*/
56         if(*(str + i) == c1) /*(str + i) の文字が c1 の文字と同じと
き*/
57         {
58             *(str + i) = c2; /*(str + i) の文字を c2 の文字に置き換
える*/
59             c++; /*置き換えた文字を数える*/
60         }
61     }
62     return c; /*置き換えた文字数を戻り値とする. *
/
63 }
64
65 int split(char *str, char *ret[], char sep, int max)
66 {
67     int i; /*for ループ用*/
68     int c = 0; /*ポインタの配列の指定用*/
69     ret[c++] = str; /*ret[0] に str の先頭アドレスを代入*/
70     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*/
71     {
72         if(*(str + i) == sep) /*(str + i) が sep のとき*/
73         {
74             *(str + i) = '\0'; /*(str + i) に NULL を代入*/
75             ret[c++] = str + (i + 1); /*ret[c] に NULL 文字の"次の"アドレス
を代入*/
76         }
77     }
78     return c; /*文字列をいくつに分割したかを戻り
値とする*/
79 }
80
81 int get_line(FILE *F, char *line)
82 {
83     if(fgets(line, MAX_LINE, F) == 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*/
84     if(*line == ESC) cmd_quit("r");
85     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*/
86     return 1; /*入力文字列が存在したとき, 1 を戻り
値とする*/
87 }
88
89 void parse_line(char *line)

```

```

94     {
95         static int i = 1;                                /*登録中止数カウント用*/
96         char *ret[2] = {NULL, NULL};                    /*コマンド文字列のポインタ, 引数文
字列のポインタ用*/
97
98         if(*line == '%')                                /*入力文字列の1文字目が%のとき*/
99         {
100             line++;
101             split(line, ret, ' ', 2);
102             exec_command(ret[0], ret[1]);
103         }
104         else if(profile_data_nitems < 10000)              /*入力コマンドではなく、登録数が1
万件以下のとき*/
105         {
106             new_profile(&profile_data_store[profile_data_nitems++] ,line);
107         }
108         else
109         {
110             fprintf(stderr, "Warning: 10000件を超える名簿データは読み込めません。計 %d 件
の登録が中止されました。 \n", i);
111             i++;
112         }
113     }
114
115     void exec_command(char *cmd, char *param)
116     {
117
118         switch (*cmd) {
119             case 'Q':
120                 if(param == NULL)
121                 {
122                     if(strcmp(cmd, "Q") == 0) cmd_quit("A");
123                     fprintf(stderr, "%sQ rと入力することで、確認メッセージなしで終了できます。
\n\n");
124                     cmd_quit(0);
125                 }
126                 else
127                 {
128                     if(strcmp(param, "r") == 0) cmd_quit(param);
129                     else {
130                         fprintf(stderr, "%sQ rと入力することで、確認メッセージなしで終了できます
. \n\n");
131                         cmd_quit(0);
132                     }
133                 }
134                 break;
135
136             case 'C': cmd_check(); break;
137             case 'P':
138                 if(strcmp(cmd, "P") == 0) cmd_print(param);
139                 else fprintf(stderr, "コマンドの入力体裁が間違っています。処理を中止しました。
\nもしかして: \"%s\" %s\n\n", *cmd, (cmd + 1));
140                 break;
141
142             case 'R':
143                 if(strcmp(cmd, "R") == 0) cmd_read(param);
144                 else fprintf(stderr, "コマンドの入力体裁が間違っています。処理を中止しました。
\nもしかして: \"%s\" %s\n\n", *cmd, (cmd + 1));
145                 break;
146
147             case 'W':
148                 if(strcmp(cmd, "W") == 0 || strcmp(cmd, "WC") == 0) cmd_write(param, ',');
149                 else if(strcmp(cmd, "WS") == 0) cmd_write(param, ';');
150                 else if(strcmp(cmd, "WT") == 0) cmd_write(param, TAB);
151                 else fprintf(stderr, "コマンドの入力体裁が間違っています。処理を中止しました。
\nもしかして: \"%s\" %s\nまたは \"%s\" %s\n\n", *cmd, *(cmd + 1), *(cmd + 2), (cmd
d + 3), *cmd, *(cmd + 1), *(cmd + 2), (cmd + 3));
152                 break;

```

```

153
154     case 'F':
155         if(strcmp(cmd, "F") == 0) cmd_find(param);
156         else fprintf(stderr, "コマンドの入力体裁が間違っています。処理を中止しました。
\n もしかして: \"%%c %s\"\\n\\n", *cmd, (cmd + 1));
157         break;
158
159     case 'S':
160         if(strcmp(cmd, "S") == 0) cmd_sort(param);
161         else fprintf(stderr, "コマンドの入力体裁が間違っています。処理を中止しました。
\n もしかして: \"%%c %s\"\\n\\n", *cmd, (cmd + 1));
162         break;
163
164     default: fprintf(stderr, "不明なコマンド\"%%s\"です。処理を中止しました。\\n\\n",
cmd); break; /*エラーメッセージを表示*/
165     }
166 }
167
168 void cmd_quit(char *param)
169 {
170     char c = 65;
171
172     if(param != NULL) /*param 部が存在している場合*/
173         if(*param == 'r') /*r オプションで確認メッセージなしで
終了*/
174         {
175             printf("正常終了。\\n\\n");
176             exit(0);
177         }
178
179     while(1)
180     {
181         printf("終了しますか?(y/n)\\n"); /*確認メッセージ*/
182         scanf("%c", &c); /*いきなり改行文字を入力した場合、この下の処理は実行
しない*/
183
184         if(c != '\\n') getchar(); /*scanf での入力時に改行文字が残ってしまうため*/
185         if(c == 'y')
186         {
187             printf("正常終了。\\n\\n");
188             exit(0);
189         }
190         else if(c == 'n')
191         {
192             printf("処理を中止しました。\\n\\n");
193             break;
194         }
195     }
196
197 void cmd_check(void) /*名簿件数表示*/
198 {
199     printf("%d profile(s)\\n", profile_data_nitems);
200 }
201
202 void cmd_print(char *param)
203 {
204     int a = 0;
205     int i = 0; /*for ループ用*/
206
207     /*atoi 関数で正常に文字列を int 値に変換できるかの確認を実施*/
208     if(param == NULL) a = 0;
209     if(param != NULL)
210     {
211         if(int_value_check(param))
212         {
213             fprintf(stderr, "引数は数値である必要があります。処理を中止しました。\\n\\n
");
214             return;

```

```

215         }
216         a = atoi(param);                                /*文字列を int 型の値に変換*/
217     }
218
219     /*a の絶対値が profile_data_nitems より大きいときか a=0 のとき*/
220     if(abs(a) >= profile_data_nitems || a == 0) a = profile_data_nitems;
221
222     if(a > 0)                                            /*引数が正の整数のとき及び例外*/
223     {
224         for(i = 0; i < a; i++)
225         {
226             printf("Id      : %d\n", profile_data_store[i].id);
227             printf("Name    : %s\n", profile_data_store[i].name);
228             printf("Birth   : %04d-%02d-%02d\n", profile_data_store[i].birthday.y, profile_data_store[i].birthday.m, profile_data_store[i].birthday.d);
229             printf("Addr.   : %s\n", profile_data_store[i].address);
230             printf("Comm.   : %s\n\n", profile_data_store[i].biko);
231         }
232     }
233     else if(a < 0)                                      /*引数が負の整数のとき*/
234     {
235         for(i = profile_data_nitems + a; i < profile_data_nitems; i++)
236         {
237             printf("Id      : %d\n", profile_data_store[i].id);
238             printf("Name    : %s\n", profile_data_store[i].name);
239             printf("Birth   : %04d-%02d-%02d\n", profile_data_store[i].birthday.y, profile_data_store[i].birthday.m, profile_data_store[i].birthday.d);
240             printf("Addr.   : %s\n", profile_data_store[i].address);
241             printf("Comm.   : %s\n\n", profile_data_store[i].biko);
242         }
243     }
244 }
245
246 void cmd_read(char *param)
247 {
248     char LINE[MAX_LINE] = {0};
249     FILE *fp;
250
251     if(param == NULL)
252     {
253         fprintf(stderr, "実行には引数が必要です。処理を中止しました。 \n\n");
254         return;
255     }
256
257     if((fp = fopen(param, "r")) == NULL)                /*指定されたファイル名が存在しない
場合*/
258     {
259         fprintf(stderr, "\"%s\"を読み込めません。カレントディレクトリにファイルが存在しないか、読み取り許可がない可能性があります。処理を中止しました。 \n\n", param);
260         return;
261     }
262
263     while(get_line(fp, LINE))                            /*文字配列 LINE に文字列を入力する*/
264     {
265         parse_line(LINE);                                /*入力文字列がある場合、構文解析を
行う*/
266     }
267
268     fclose(fp);
269 }
270
271 void cmd_write(char *param, char sep)
272 {
273     int i;                                                /*for ループ用*/
274     FILE *fp;
275
276     if(param == NULL)
277     {

```

```

278         fprintf(stderr, "実行には引数が必要です。処理を中止しました。 \n\n");
279         return;
280     }
281
282     if((fp = fopen(param, "w")) == NULL)          /*指定されたファイル名が存在しない
場合*/
283     {
284         fprintf(stderr, "\"%s\"に書き込めません。書き込み許可がない可能性があります
. 処理を中止しました。 \n\n", param);
285         return;
286     }
287
288     /*CSV,SCSV,TSV 形式で出力*/
289     for(i = 0; i < profile_data_nitems; i++)
290     {
291         fprintf(fp, "%d%c", profile_data_store[i].id, sep);
292         fprintf(fp, "%s%c", profile_data_store[i].name, sep);
293         fprintf(fp, "%d-%d-%d%c", profile_data_store[i].birthday.y, profile_data_store[i].birthday.m, profile_data_store[i].birthday.d, sep);
294         fprintf(fp, "%s%c", profile_data_store[i].address, sep);
295         fprintf(fp, "%s\n", profile_data_store[i].biko);
296     }
297
298     fclose(fp);
299 }
300
301 void cmd_find(char *param)
302 {
303     int i = 0;                                /*for ループ用*/
304     char num1[12];                            /*int 値を文字列に変換する際に使用*/
305     char num2[36];                            /*int 値を文字列に変換する際に使用*/
306     char num3[36];                            /*誕生日の0 埋めなし用*/
307     struct profile *p;
308
309     if(param == NULL)
310     {
311         fprintf(stderr, "実行には引数が必要です。処理を中止しました。 \n\n");
312         return;
313     }
314
315     for(i = 0; i < profile_data_nitems; i++)
316     {
317         p = &profile_data_store[i];
318
319         /*int 値を文字列に変換して代入*/
320         sprintf(num1, "%d", p->id);
321         sprintf(num2, "%04d-%02d-%02d", (p->birthday).y, (p->birthday).m, (p->birthday).d);
322         sprintf(num3, "%d-%d-%d", (p->birthday).y, (p->birthday).m, (p->birthday).d);
323
324         if(strcmp(param, num1) == 0 ||          /*ID 比較*/
325            strcmp(param, p->name) == 0 ||      /*name 比較*/
326            strcmp(param, num2) == 0 ||          /*birthday 比較 (0 埋め)*/
327            strcmp(param, num3) == 0 ||          /*birthday 比較 (0 無視)*/
328            strcmp(param, p->address) == 0 ||    /*address 比較*/
329            strcmp(param, p->biko) == 0)         /*biko 比較*/
330         {                                       /*該当名簿情報表示*/
331             printf("Id      : %d\n", p->id);
332             printf("Name   : %s\n", p->name);
333             printf("Birth  : %04d-%02d-%02d\n", (p->birthday).y, (p->birthday).m, (p->birthday).d);
334             printf("Addr.  : %s\n", p->address);
335             printf("Comm.  : %s\n\n", p->biko);
336         }
337     }
338 }
339
340 void cmd_sort(char *param)

```

```

341     {
342         int a = 0;
343         int a_buff = 0;
344         int i1 = 0;
345         int i2 = 0;
346         char num1[36];
347         char num2[36];
348         struct profile *sp1;
349         struct profile *sp2;
350         char *cp1;
351         char *cp2;
352
353         /*atoi 関数で正常に文字列を int 値に変換できるかの確認を実施*/
354         if(param == NULL)
355         {
356             fprintf(stderr, "実行には引数が必要です。処理を中止しました。 \n\n");
357             return;
358         }
359         if(int_value_check(param))
360         {
361             fprintf(stderr, "引数は整数値である必要があります。処理を中止しました。 \n\n");
362             return;
363         }
364         a_buff = atoi(param);
365
366         if(a_buff < 1 || a_buff > 5) /*引数のチェック*/
367         {
368             fprintf(stderr, "引数は 1 から 5 のいずれかの数値である必要があります。処理を中止しました。 \n\n");
369             return;
370         }
371
372         for(i1 = 0; i1 < profile_data_nitems; i1++)
373         {
374             for(i2 = 0; i2 < profile_data_nitems - i1 - 1; i2++)
375             {
376                 a = a_buff;
377                 sp1 = &profile_data_store[i2];
378                 sp2 = &profile_data_store[i2 + 1];
379
380                 if(a == 2) /*氏名で並び換え*/
381                 {
382                     cp1 = sp1->name;
383                     cp2 = sp2->name;
384                     if(strcmp(cp1, cp2) == 0) a = 1; /*項目が一致していた場合は、ID の昇
385                     順になるように並び変える*/
386                 }
387
388                 if(a == 3) /*誕生日で並び換え*/
389                 {
390                     sprintf(num1, "%010d%010d%010d", (sp1->birthday).y, (sp1->birthday).
391                     m, (sp1->birthday).d);
392                     sprintf(num2, "%010d%010d%010d", (sp2->birthday).y, (sp2->birthday).
393                     m, (sp2->birthday).d);
394                     cp1 = num1;
395                     cp2 = num2;
396                     if(strcmp(cp1, cp2) == 0) a = 1; /*項目が一致していた場合は、ID の昇
397                     順になるように並び変える*/
398                 }
399
400                 if(a == 4) /*住所で並び換え*/
401                 {
402                     cp1 = sp1->address;
403                     cp2 = sp2->address;
404                     if(strcmp(cp1, cp2) == 0) a = 1; /*項目が一致していた場合は、ID の昇
405                     順になるように並び変える*/

```



```

401         }
402         if(a == 5) /*備考で並び換え*/
403         {
404             cp1 = sp1->biko;
405             cp2 = sp2->biko;
406             if(strcmp(cp1, cp2) == 0) a = 1; /*項目が一致していた場合は、ID の昇
順になるように並び変える*/
407         }
408
409         if(a == 1) /*ID で並び換え*/
410         {
411             sprintf(num1, "%010d", sp1->id);
412             sprintf(num2, "%010d", sp2->id);
413             cp1 = num1;
414             cp2 = num2;
415         }
416
417         if(strcmp(cp1, cp2) > 0) data_move(sp1, sp2); /*文字列比較*/
418     }
419 }
420
421 void data_move(struct profile *sp1, struct profile *sp2)
422 {
423     struct profile swap_data;
424
425     /*sp1 のデータを swap_data に退避*/
426     swap_data.id = sp1->id;
427     strcpy(swap_data.name, sp1->name);
428     swap_data.birthday.y = (sp1->birthday).y;
429     swap_data.birthday.m = (sp1->birthday).m;
430     swap_data.birthday.d = (sp1->birthday).d;
431     strcpy(swap_data.address, sp1->address);
432     swap_data.biko = sp1->biko;
433
434     /*sp2 のデータを sp1 に移動*/
435     sp1->id = sp2->id;
436     strcpy(sp1->name, sp2->name);
437     (sp1->birthday).y = (sp2->birthday).y;
438     (sp1->birthday).m = (sp2->birthday).m;
439     (sp1->birthday).d = (sp2->birthday).d;
440     strcpy(sp1->address, sp2->address);
441     sp1->biko = sp2->biko;
442
443     /*swap_data を sp2 に移動*/
444     sp2->id = swap_data.id;
445     strcpy(sp2->name, swap_data.name);
446     (sp2->birthday).y = (swap_data.birthday).y;
447     (sp2->birthday).m = (swap_data.birthday).m;
448     (sp2->birthday).d = (swap_data.birthday).d;
449     strcpy(sp2->address, swap_data.address);
450     sp2->biko = swap_data.biko;
451 }
452
453 void new_profile(struct profile *profile_p, char *line)
454 {
455     char *ret[10];
456     char *ret2[4];
457     /*誕生日の情報を分割し、その先頭ア
ドレスを保存*/
458     char sep;
459     /*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*/
460     char sep2 = '-';
461     int max = 5;
462     int max2 = 4;
463     int birth_c = 0;
464     int MAX_BIKO = 0;
465     static int i = 0;
466     /*備考の文字数カウント用*/
467     /*入力項目の行番号監視*/

```

```

465
466         i++;
467         sep = input_format_check(line);
468
469         if(sep == 0)                                     /*カンマ, セミコロン, タブのいずれ
でも区切れない場合*/
470         {
471             fprintf(stderr, "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, カンマ区切り, セミコロン区切り, タブ区切りのいずれかの体裁である必要があります. 処理を
中止しました (項目番号:%d). \n\n", i);
472             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*/
473             return;
474         }
475         split(line, ret, sep, max);                       /*ID, 名前などの情報を分割する*/
476
477         /*atoi 関数で正常に文字列を int 値に変換できるかの確認を実施*/
478         if(int_value_check(ret[0]))
479         {
480             fprintf(stderr, "ID の項目は数値である必要があります. 処理を中止しました (項
目番号:%d). \n\n", i);
481             profile_data_nitems--;
482             return;
483         }
484
485         /*ID が負になる場合は, 処理を中断する*/
486         if(atoi(ret[0]) < 0)
487         {
488             fprintf(stderr, "ID の項目は正の数値である必要があります. 処理を中止しました (
項目番号:%d). \n\n", i);
489             profile_data_nitems--;
490             return;
491         }
492
493         birth_c = split(ret[2], ret2, sep2, max2);        /*誕生日の年, 月, 日を分割する*/
494         if(birth_c != 3)                                   /*誕生日の年, 月, 日を正常に分割で
きない場合*/
495         {
496             fprintf(stderr, "誕生日は\"年-月-日\"の形で入力される必要があります. 処理を
中止しました (項目番号:%d). \n\n", i); /*年, 月, 日に分割できない場合, 処理を停止*/
497             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*/
498             return;
499         }
500
501         /*atoi 関数で正常に文字列を int 値に変換できるかの確認を実施*/
502         if(int_value_check(ret2[0]) ||
503            int_value_check(ret2[1]) ||
504            int_value_check(ret2[2]) )
505         {
506             fprintf(stderr, "誕生年, 月, 日の項目は数値である必要があります. 処理を中止し
ました (項目番号:%d). \n\n", i);
507             profile_data_nitems--;
508             return;
509         }
510
511         /*年月日の体裁チェック*/
512         if(day_format_check(atoi(ret2[0]), atoi(ret2[1]), atoi(ret2[2])))
513         {
514             fprintf(stderr, "その年月日は存在しません. 処理を中止しました (項目番号:%d). \
n\n", i);
515             profile_data_nitems--;
516             return;
517         }
518
519         /*構造体への情報の書き込み処理*/
520         profile_p->id = atoi(ret[0]);                     /*ID の書き込み*/
521         strncpy(profile_p->name, ret[1], 69);             /*名前の書き込み*/

```

```

522     (profile_p->birthday).y = atoi(ret2[0]);        /*誕生年の書き込み*/
523     (profile_p->birthday).m = atoi(ret2[1]);        /*誕生月の書き込み*/
524     (profile_p->birthday).d = atoi(ret2[2]);        /*誕生日の書き込み*/
525     strncpy(profile_p->address, ret[3], 69);        /*住所の書き込み*/
526
527     MAX_BIKO = strlen(ret[4]) + 1;                  /*備考情報の文字数のカウント*/
528
529     profile_p->biko = (char *)malloc(sizeof(char)* MAX_BIKO); /*文字数だけメモリ確
保*/
530     strncpy(profile_p->biko, ret[4], MAX_BIKO);    /*備考の書き込み*/
531 }
532
533 int int_value_check(char *str)
534 {
535     if((*str >= 48 && *str <= 57) || *str == 43 || *str == 45) str++; /*1文字目は'-、'
と'+、'を許容*/
536     else return 1;
537
538     while(*str) /*入力文字列の終端に辿り着くまでループ*/
539     {
540         if(*str >= 48 && *str <= 57) str++; /*確認する文字が0~9の場合、次の文字を確
認*/
541         else return 1; /*確認する文字が0~9で無い場合、戻り値1*/
542     }
543     return 0; /*変換不可能文字列と判定した場合*/
544 }
545
546 char input_format_check(char *str)
547 {
548     int com_c = 0; /*コンマの使用回数のカウント*/
549     int semi_c = 0; /*セミコロンの使用回数のカウント*/
550     int tab_c = 0; /*タブの使用回数のカウント*/
551
552     while(*str) /*入力文字列の終端に辿り着くまでループ*/
553     {
554         if(*str == ',') com_c++; /*カンマ区切り*/
555         if(*str == ';') semi_c++; /*セミコロン区切り*/
556         if(*str == TAB) tab_c++; /*タブ区切り*/
557         str++;
558     }
559
560     /*正しく区切れると見込まれる文字の文字コードを返す*/
561     if(com_c == 4) return 44;
562     if(semi_c == 4) return 59;
563     if(tab_c == 4) return 9;
564
565     return 0; /*どの文字でも区切ることができない場合(
例外)*/
566 }
567
568 int day_format_check(int y, int m, int d) /*存在する年月日の場合、戻り値0、それ以
外は戻り値1*/
569 {
570     switch(m){
571     case 1:
572         if(d >= 1 && d <= 31) return 0;
573         break;
574     case 2:
575         if((y % 4 == 0 && y % 100 != 0) || y % 400 == 0) /*閏年の場合、dを29日まで許可*/
576         {
577             if(d >= 1 && d <= 29) return 0;
578         }
579         else /*そうでない場合は、dは28日まで*/
580         {
581             if(d >= 1 && d <= 28) return 0;
582         }
583         break;

```

```

584     case 3:
585         if(d >= 1 && d <= 31) return 0;
586         break;
587     case 4:
588         if(d >= 1 && d <= 30) return 0;
589         break;
590     case 5:
591         if(d >= 1 && d <= 31) return 0;
592         break;
593     case 6:
594         if(d >= 1 && d <= 30) return 0;
595         break;
596     case 7:
597         if(d >= 1 && d <= 31) return 0;
598         break;
599     case 8:
600         if(d >= 1 && d <= 31) return 0;
601         break;
602     case 9:
603         if(d >= 1 && d <= 30) return 0;
604         break;
605     case 10:
606         if(d >= 1 && d <= 31) return 0;
607         break;
608     case 11:
609         if(d >= 1 && d <= 30) return 0;
610         break;
611     case 12:
612         if(d >= 1 && d <= 31) return 0;
613         break;
614     }
615     return 1;                                /*存在しない年月日である(例外)*/
616 }
617
618 int main(void)
619 {
620     char LINE[MAX_LINE] = {0};              /*標準入力文字列(1行分)はmain関数で管理*/
/
621
622     while(get_line(stdin, LINE))             /*文字配列 LINE に文字列を入力する*/
623     {
624         parse_line(LINE);                   /*入力文字列がある場合、構文解析を行う*/
625     }
626     return 0;
627 }

```

## 8.8 自作の string 型の変数で関数間で文字列を渡すプログラム

プログラムのソースコード (26 行)

```

1      #include <stdio.h>
2
3      struct string
4      {
5          char a[50];
6      };
7
8      struct string func(void)
9      {
10         struct string fstr;
11
12         printf("Input string :fstr\n");
13         scanf("%s", &fstr.a[0]);
14
15         return fstr;
16     }

```

```

17
18     int main(void)
19     {
20         struct string mstr;
21
22         mstr = func();
23         printf("Output string :mstr\n\"%s\"\n",&mstr.a[0]);
24
25         return 0;
26     }

```

## 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](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.
- [10] strlen, <http://www9.plala.or.jp/sgwr-t/lib/strlen.html>, 2020. 06. 04
- [11] メモリ領域の動的確保, <http://rainbow.pc.uec.ac.jp/edu/program/b1/programming-6.htm>, 2020. 06. 04
- [12] malloc, <http://www9.plala.or.jp/sgwr-t/lib/malloc.html>, 2020. 06. 04