

# プログラミング演習 1

## 期末レポート

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

出題日: 2020 年 04 月 22 日

提出日: 2020 年 06 月 08 日

締切日: 2020 年 06 月 10 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 6 についての内容を報告する。

課題 1 文字列操作の基礎: `subst` 関数と `split` 関数の実装

課題 2 構造体や配列を用いた名簿データの定義

課題 3 標準入力の取得と構文解析

課題 4 CSV データ登録処理の実装

課題 5 コマンド中継処理の実装

課題 6 コマンドの実装: `%p` コマンド

また、取り組んだ課題のうち、特に以下の課題については、詳細な考察を行った。

課題 1 文字列操作の基礎: `subst` 関数と `split` 関数の実装

課題 3 標準入力の取得と構文解析

課題 4 CSV データ登録処理の実装

## 2 プログラムの作成方針

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

### (1) 基本要件

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

- (c) 名簿データの全数表示, および, 部分表示
  - (d) 名簿データのファイルへの保存, および, ファイルからの復元
  - (e) 名簿データの検索と表示
  - (f) 名簿データの整列
4. 標準入力からのユーザ入力を通して, データ登録やデータ管理等の操作を可能とすること.
  5. 標準出力には, コマンドの実行結果のみを出力すること.

## (2) 基本仕様

1. 名簿データは, コンマ区切りの文字列 (**CSV** 入力と呼ぶ) で表されるものとし, 図??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??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??章に添付している. 最終的なプログラムは全部で 247 行からなる. 以下では, ??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 (??節) に基づく.

## 3.1 文字列操作の基礎: `subst` 関数と `split` 関数の実装

まず, 汎用的な文字列操作関数として, `subst()` 関数を 44–58 行目で宣言し, `split()` 関数を 60–76 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。9–14 行目で、`date` 構造体を定義し、16–23 行目で、`profile` 構造体を定義している。この `profile` 型の変数 1 つが、名簿データ 1 つに相当する。そして、41 行目の `profile_data_store` 変数で、全名簿データを管理し、42 行目の `int` 型変数 `profile_data_nitems` で、名簿データの個数を管理する。`date` 構造体の定義にあたっては、年、月、日に分けて情報を管理できるよう、3 つの `int` 型変数を用意している。`profile` 構造体は、その要素に `date` 構造体を含む。

`profile` 構造体の各要素について説明する。まず、ID を格納するための `int` 型変数 `id` である。これは `int` 型変数の最小値 -2147483648 と最大値 2147483647 の間で整数値を格納できる。次に、氏名を格納するための `char` 型の配列 `name` である。これは NULL 文字を含めて最大 70 文字の文字列を格納できる。そして、`date` 型の変数 `birthday` である。これは前述したように年、月、日の 3 つの `int` 型のメンバからなる変数である。次に、住所を格納するための `char` 型の配列 `address` である。文字列の配列 `name` 同様、NULL 文字を含めて最大 70 文字の文字列を格納できる。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 78–84 行目で宣言している。構文解析のための `parse_line()` 関数は、86–101 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。標準入力が入力された文字列が NULL または、制御文字 ESC を 1 文字目に入力した場合は、呼び出し元に戻り値 0 を返す。それ以外の場合、`subst()` 関数で末尾の改行文字を NULL 文字に置き換えた後、呼び出し元に戻り値 1 を返す。

`parse_line(line)` 関数は、他関数からの文字列配列をポインタ `line` で取得し、入力内容が特定のコマンドとその引数であるか、単なる文字列であるかを判定し、それぞれ `exec_command()` 関数を呼び出すか、`new_profile()` 関数を呼び出す。`get_line()` 関数で入力された入力文字列の 1 文字目が % である場合、2 文字目の文字を文字変数 `cmd` に代入、4 文字目以降の文字列をポインタ `param` で参照できるようにする。そして、`exec_command()` 関数を呼び出す。

### 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、`split()` 関数で 1 行分の入力をカンマを基準に ID、氏名、誕生日、住所、備考に分け、ポインタ配列 `ret[]` にそれぞれの文字列要素の先頭アドレスを格納する。`split()` 関数の戻り値を用いて、要素を ID、氏名、誕生日、住所、備考に分割できていることを確認する。分割できていなかった場合は処理を中止する。そのうち、誕生日は `split()` 関数でハイフンを基準に年、月、日の要素に分け、ポインタ配列 `ret2[]` にそれぞれの要素を格納する。こちらも `split()` 関数の戻り値を用いて、年、月、日分割できていることを確認し、分けられていない場合は処理を中止する。処理を中止した場合、変数 `profile_data_store` への代入処理を行わないため、`data_profile_nitems` のみが、`new_profile()` 関数の呼び出し時にインクリメントされた状態になる。このままだと、要素が入らない状態になってしまうため、`new_profile()` 関数を終了する前に、`profile_data_nitems` をデクリメントする。

処理が中止されなかった場合、代入処理を行う。まず、ID の代入を行う。ただし、この ID に対応する変数 `profile_data_store` のメンバは `profile` 構造体で、`int` 型の値として宣言されている。もともとの入力は文字列であるため、これをそのまま代入することはできない。例えば、入力された ID 情報が 437 だったとしても、それは文字 '4'、'3'、'7' のことであり、整数値 437 のことではない。この文字列 437 を `int` 型の変数に代入するため、`atoi()` 関数を用いる。`atoi()` 関数は、文字列で表現された数値を `int` 型の整数値に変換するものである。変換不能な文字列の場合、結果は 0 となる [?]. 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 [?]. 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ので、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 [?]. 次に、`malloc()` 関数にて必要分のメモリを確保する [?]. `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 [?]. 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関数 `exec_command(cmd, param)` 関数を、115-128 行で宣言している。なお、使用可能なコマンドは表??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 文字の変数 `cmd` とコマンドの引数とする文字列を指すアドレス `param` を受け取る。`switch` 文で `cmd` の値によって、呼び出す関数を選択する。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。

%Q コマンドは、`exec_command()` 関数の `switch` 文中で `cmd_quit()` 関数を呼び出す。`cmd_quit()` 関数に引数はなく、`exit()` 関数でプログラムを終了する役割を持つ。ここでは正常終了を示す 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。

```
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。

3.6 コマンドの実装：%p コマンド

%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。

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、??節で説明した。

プログラムは、CentOS 7.6.1810(Core) で動作を確認しているが、一般的な UNIX で動作することを意図している。なお、以降の実行例における、行頭の\$記号は、CentOS 7.6.1810(Core) におけるターミナルのプロンプトである。

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

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

次に、プログラムを実行する。以下の実行例は、プログラム実行中の動作例を模擬するため、任意の `csv` ファイルを標準入力のリダイレクションにより与えることで、実行する例を示している [?]. `csv` ファイルの内容は、図??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。??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
```

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

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. ??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. ??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。引数が無い場合や 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。

## 5 考察

??章のプログラムの説明、および、??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、??章で述べた基本要件と基本仕様のいずれも満たしていることを示した。ここでは、個別の課題のうち、以下の 3 つの項目について、考察を述べる。

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

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

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、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（??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、入力文字列の途中に `NULL` 文字が出現することは標準入力では起こらないため、文字置き換えのループ処理の継続条件に `NULL` 文字を用いることで、確実に入力文字列の終端である `NULL` 文字手前まで文字の置き換えを行えるようにした。`c1` と `c2` に同じ文字が入力された場合、文字の置き換え処理は実質行われずに等しい。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 81 行目に戻り値の条件を追加した（??節のソースコード。??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[?]。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 [?]. `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。

## 6 感想

毎回の講義でプログラムのソースコードを追記していくとき、どの関数を上から順に書けばいいかを考えていたが、煩雑化してしまうため、関数プロトタイプ宣言をすることで見やすくなり作業がしやすくなった [?]. 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 [?, ?]. `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らない。 `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だ。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、“.” でつなぐことができるのかもしれない。 ?? 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 ?? 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.

## 7 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, ??章に示した課題については, ??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.

### 7.1 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     }
```

### 7.2 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;
9         int c = 0;
10        for(i = 0; *(str + i) != '\0'; i++)
11        {
12            if(c1 == c2) break;
13            if(*(str + i) == c1)
14            {
15                *(str + i) = c2;
16                c++;
17            }
18        }
19        return c;
20    }
21
22    int split(char *str, char *ret[], char sep, int max)
23    {
24        int i;
25        int c = 0;
26
27        ret[0] = str;
28
29        for(i = 0; *(str + i) != '\0' && c < max; i++)
30        {
31            if(*(str + i) == sep)
32            {
33                *(str + i) = '\0';
34                c++;
35                ret[c] = str + (i + 1);
36            }
37        }
38        return c;
39    }
40
41    int get_line(char *line)
42    {
43        if(fgets(line, MAX_LINE, stdin) == NULL) return 0;
44        if(*line == ESC) return 0;
45        subst(line, '\n', '\0');
46        return 1;
47    }
48
49    int main(void)
50    {
51        char line[MAX_LINE] = {0};
52        char *ret[10];
53        char sep = ',';
54        int max = 10;
55        int c, i, a = 1;
56
57        while(get_line(line))

```

/\*for ループ用\*/  
 /\*置き換えた文字数の  
 カウント用\*/  
 /\*入力文字列の終端に辿り  
 着くまでループ\*/  
 /\*見た目上文字列に変化が  
 ないとき\*/  
 /\*(str + i) の文字が c1 の  
 文字と同じとき\*/  
 /\*(str + i) の文字を c2 の  
 文字に置き換える\*/  
 /\*置き換えた文字を数える\*/  
 /\*置き換えた文字数を戻り値  
 とする。\*/  
 /\*for ループ用\*/  
 /\*ポインタの配列の指定用\*/  
 /\*ret[0] に str の先頭アドレス  
 を代入\*/  
 /\*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\*/  
 /\*(str + i) が sep のとき\*/  
 /\*(str + i) に NULL を代入\*/  
 /\*ret[c] に NULL 文字の"次の"  
 アドレスを代入\*/  
 /\*文字列をいくつに分割したか  
 を戻り値とする\*/  
 /\*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\*/  
 /\*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\*/  
 /\*subst 関数により、入力の  
 改行文字を終端文字に置き換える\*/  
 /\*入力文字列が存在したとき、  
 1 を戻り値とする\*/  
 /\*入力文字列は最大 1024 文字\*/  
 /\*csv ファイルからの入力を  
 想定しているため、カンマ\*/  
 /\*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 }

```

### 7.3 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>                           /*exit 関数用*/
3
4      #define ESC 27                                /*文字列 ESC を ESC の ASCII
コードで置換*/
5      #define MAX_LINE 1025                         /*文字配列 LINE の最大入力数
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;                                     /*for ループ用*/
25         int c = 0;                                /*置き換えた文字数のカウ
ント用*/
26         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*/
27             {
28                 if(c1 == c2) break;                /*見た目上文字列に変化が
ないとき*/
29                 if(*(str + i) == c1)               /*(str + i) の文字が c1 の
文字と同じとき*/
30                     {
31                         *(str + i) = c2;            /*(str + i) の文字を c2 の
文字に置き換える*/
32                         c++;                          /*置き換えた文字を数える*/
33                     }
34             }
35         return c;                                  /*置き換えた文字数を戻り値
とする。*/
36     }
37
38     int split(char *str, char *ret[], char sep, int max)
39     {

```

```

40         int i;                                /*for ループ用*/
41         int c = 0;                             /*ポインタの配列の指定用*/
42
43         ret[0] = str;                          /*ret[0] に str の先頭アドレ
スを入力*/
44
45         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*/
46         {
47             if(*(str + i) == sep)               /*(str + i) が sep のとき*/
48             {
49                 *(str + i) = '\0';             /*(str + i) に NULL を代入*/
50                 c++;
51                 ret[c] = str + (i + 1);         /*ret[c] に NULL 文字の"次の"
アドレスを代入*/
52             }
53         }
54         return c;                              /*文字列をいくつに分割したか
を戻り値とする*/
55     }
56
57     int get_line(char *line)
58     {
59         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*/
60         if(*line == ESC) return 0;             /*ESC を 1 文字目に入力すること
により 0 を戻り値とする (デバッグ用) */
61         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*/
62         return 1;                              /*入力文字列が存在したとき,
1 を戻り値とする*/
63     }
64
65     void parse_line(char *line)
66     {
67         char cmd;                                /*% の次の 1 文字を格納用*/
68         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*/
69         char *buffer = "(Null Parameter)";      /*例外処理用*/
70
71         if(*line == '%')                        /*入力文字列の 1 文字目が % のとき*/
72         {
73             cmd = *(line + 1);                  /*cmd に入力文字列の 2 文字目
の値を代入*/
74             if(*(line + 3) != '\0') 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*/
75             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*/
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; /*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) */
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);
break; /* エラーメッセージを表示 */
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");
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     printf("line number:%d\n", a);
149     printf("input: \"%s\"\n", line);
150     c = split(line, ret, sep, max);
151     for (i = 0; i <= c; i++)
152     {
153         printf("split[%d]: \"%s\"\n", i, ret[i]);
154     }
155     printf("\n");
156     a++;
157 }

```

/\*確認メッセージ\*/

/\*getchar での入力時に改行文字が残ってしまうため\*/

/\*csv ファイルからの入力を想定しているため、カンマ\*/

/\*値を main 関数終了時まで保持するため、static int 型\*/

/\*split 関数を呼び出す\*/

/\*見やすさのために改行\*/

```

159
160     int main(void)
161     {
162         char LINE[MAX_LINE] = {0};
管理*/
163
164         while(get_line(LINE))
(get_line 関数)*/
165         {
166             parse_line(LINE);
を行う (parse_line 関数)*/
167         }
168         return 0;
169     }

```

/\*入力文字列 (1 行分) は main 関数で

/\*文字配列 LINE に文字列を入力する

/\*入力文字列がある場合、構文解析

## 7.4 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>
3     #include <stdlib.h>
4
5     #define ESC 27
の ASCII コードで置換*/
6     #define MAX_LINE 1025
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];
格納*/
42    int profile_data_nitems = 0;
保存数を格納*/
43

```

/\*strncpy 関数等用\*/

/\*exit 関数用\*/

/\*文字列 ESC を ESC

/\*文字配列 LINE の

/\*profile 情報を

/\*profile 情報の

```

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];
187     char sep = ',';
188     char sep2 = '-';
189     int max = 10;
190     int max2 = 4;
191     int c, birth_c;
192     static int a = 1;
193     printf("line number:%d\n", a);
194     //printf("input: \"%s\"\n", line);
195     c = split(line, ret, sep, max);
196     if(c <= 2)
197     {
198         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
199             は必須項目です. \n 処理を中止しました. \n\n");
200         profile_data_nitems--;
201         return;
202     }
203     birth_c = split(ret[2], ret2, sep2, max2);
204     if(birth_c != 2)

```

を分割し、その先頭アドレスを保存\*/  
 からの入力を想定しているため、カンマ\*/  
 文字列にあるハイフンで区切るため、ハイフン\*/  
 終了時まで保持する必要があるため、static int 型\*/  
 バグ用) \*/  
 のまま表示 (デバッグ用) \*/  
 情報を分割する\*/  
 がない場合\*/  
 構造物に情報を書き込まないため\*/  
 日を分割する\*/  
 日を正常に分割できない場合\*/

/\*誕生日の情報  
 /\*csv ファイル  
 /\*誕生日の入力  
 /\*値を main 関数  
 /\*行番号表示 (デ  
 /\*入力文字列をそ  
 /\*ID, 名前などの  
 /\*備考以外の入力  
 /\*処理中止により,  
 /\*誕生日の年, 月,  
 /\*誕生日の年, 月,

```

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", 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 }

```

## 7.5 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:\n%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     }

```

## 7.6 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;
47         int c = 0;
数のカウント用*/
48         for(i = 0; *(str + i) != '\0'; i++)
終端に辿り着くまでループ*/
49         {
50             if(c1 == c2) break;
に変化がないとき*/
51             if(*(str + i) == c1)
文字が c1 の文字と同じとき*/
52             {
53                 *(str + i) = c2;
文字を c2 の文字に置き換える*/
54                 c++;
を数える*/
55             }
56         }
57         return c;
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;
63         int c = 0;
の指定用*/
64
65         ret[c++] = str;
の先頭アドレスを代入*/
66
67         for(i = 0; *(str + i) != '\0' && c < max; i++)
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)
sep のとき*/
70             {
71                 *(str + i) = '\0';
NULL を代入*/
72                 ret[c++] = str + (i + 1);
文字の"次の"アドレスを代入*/
73             }
74         }
75         return c;
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;
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');
より, 入力の改行文字を終端文字に置き換える*/
83         return 1;
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;
を格納用*/
89         char *param;
パラメータとなる文字列へのポインタ用*/
90
91         if(*line == '%')
の 1 文字目が % のとき*/
92         {

```

```

/*for ループ用*/
/*置き換えた文字

```

```

/*入力文字列の

```

```

/*見た目上文字列

```

```

/*(str + i) の

```

```

/*(str + i) の

```

```

/*置き換えた文字

```

```

/*置き換えた文字

```

```

/*for ループ用*/
/*ポインタの配列

```

```

/*ret[0] に str

```

```

/*c が max より小

```

```

/*(str + i) が

```

```

/*(str + i) に

```

```

/*ret[c] に NULL

```

```

/*文字列をいく

```

```

/*入力文字列が

```

```

/*subst 関数に

```

```

/*入力文字列が

```

```

/*% の次の 1 文字

```

```

/*コマンドの

```

```

/*入力文字列

```

```

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
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         c = split(line, ret, sep, max);                    /*ID, 名前などの
情報を分割する*/
210         if(c != 5)                                          /*入力形式が合わ
ない場合*/
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--;                          /*処理中止により,

```

```

構造体に情報を書き込まないため*/
214         return;
215     }
216     birth_c = split(ret[2], ret2, sep2, max2);           /*誕生日の年, 月,
日を分割する*/
217     if(birth_c != 3)                                     /*誕生日の年, 月,
日を正常に分割できない場合*/
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--;                             /*処理中止により,
構造体に情報を書き込まないため*/
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]);                         /*ID の書き込み*/
226     strncpy(profile_p->name, ret[1], 69);                 /*名前の書き込み*/
227     (profile_p->birthday).y = atoi(ret2[0]);              /*誕生年の書き込み*/
228     (profile_p->birthday).m = atoi(ret2[1]);              /*誕生月の書き込み*/
229     (profile_p->birthday).d = atoi(ret2[2]);              /*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 }

```

## 7.7 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] 1 0 - 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