プログラミング演習 1
期末レポート

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

出題日: 2019 年 04 月 10 日
提出日: 2019 年 05 月 26 日
締切日: 2019 年 05 月 29 日

1 概要

プログラミング演習 1 にて C 言語の実践的なプログラミングの演習を行った．演習内容としては，外部からのデータ入力を処理すること，ポインタや構造体のデータ構造について学習をした．そして，その実践的な内容として，段階的な詳細化を行いながら，名簿管理プログラムの作成を課題としてこなした．本課題を通し，データ構造管理の手法やポインタを利用した構造体の仕組み，さらに入力されたデータの編集処理の手法について新たな知識を得た．また，完成したプログラムの結果を通して，様々なデータ入力を考慮した際のエラー処理についても考察を行った．

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

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

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

   (b) ただし%で始まるコマンドを受け付けて，登録してあるデータを表示したり整列したりするなどの機能を持つ．実装するコマンドを表 1 に示す．
2. 要件
   (a) 名簿データは配列などを用いて少なくとも 10000 件のデータを登録できるようにする．今回のプログラムでは，構造体 `struct profile` の配列 `profile_data_store[10000]` を宣言して，10000 件のデータを格納できるようにする．
   (b) 名簿データは構造体 `struct profile` および構造体 `struct date` を利用して，構造を持ったデータとしてプログラム中に定義して利用する．実装すべきデータ構造は表 2 である．表中の n bytes とは，n バイトの `char` 型配列を意味する．

また，本レポートでは以下の考察課題について考察をおこなった．

表 1: 実装するコマンド

コマンド	解説	パラメータ範囲
%C	メモリ中のデータ件数を表示する	パラメータなし
%P	メモリ中データを，n に応じて表示させる	n:-10000〜10000 (0:全件表示 n>0:前から指定件数正順表示 n<0:後ろから指定件数正順表示)
%Q	システムを終了する	パラメータなし

表 2: 名簿データ

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

1. 追加機能に関する考察.
2. エラー処理に関する考察.
3. 構造体 `struct profile` がメモリ中を占めるバイト数についての確認.

2 プログラムの作成方針

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

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

2.1 宣言部

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

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

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

 struct profile profile_data_store[10000];
```

```
int profile_data_nitems = 0;
```

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

2.2 解析部

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

(a) 標準入力から読むべき行が残っている間，文字の配列 char line[] に 1 行分を読み込む．

(b) line の 1 文字目が，’%’ ならば，2 文字目をコマンド名，3 文字目以降をその引数として，決定されたコマンドを実行する．

(c) さもなくば line を新規データとみなし’,’ を区切りとして 5 つの文字列に分割する．

(d) 分割してできた 5 つの文字列を変換部に渡し構造体に代入する．

(e) 次の行を読み込む

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

2.3 変換部

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

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

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

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

2.4　各種コマンド実現部

“各種コマンド実現部” は，表 1 にある実装コマンドの，実際の処理をおこなう部分である．このレポートでは，具体的には，登録されているデータ件数を表示する機能と，指定形式でデータ内容を表示する機能，また，システムを終了させるための機能の 3 つを実装している．

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

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

3　プログラムおよびその説明

プログラムリストは 8 節に添付している．プログラムは全部で 235 行からなる．以下では，前節の作成方針における分類に基づいて，プログラムの主な構造について説明する．

3.1　汎用的な関数の宣言（62 行目から 112 行目）

まず，汎用的な文字列操作関数として，subst() 関数を 62 から 72 行目で宣言し，split() 関数を 74 から 102 行目で宣言，さらに get_line() 関数を 105 から 111 行目で宣言している．

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

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

get_line() は，標準入力からの入力を受け付けて，引数 input へ格納後，input の内容に応じて処理を行わせている．具体的には，標準入力の入力 NULL だった場合に失敗を意味するように 0 を返し，

3.2　変換部（208 行目から 235 行目）

12〜16 行目は struct date 型の宣言部である．メンバについては，変数 y は設立年，変数 m は設立月，変数 d は設立日にそれぞれ対応させている．

18〜24 行目は struct profile 型の宣言部，208〜235 行目はそれを扱う関数 new_profile である．メンバについては，設立日を入れ子構造にしている．こうすることで，要素を管理しやすくできる．なお，備考に対応する文字列*others は任意長を許すようにしているため，malloc 関数と strlen 関数を用いて文字列を動的に格納できるようにした．文字列から各データ型への変換を担う関数は，struct new_profile とすることで，変換部であることを明確にした．具体的な処理内容としては，受け取った文字列 str を分割し，分割した文字列を ret1[] に格納し，その後要素ごとに対応する構造体メンバにエラー検出のある strncpy 関数を用いて格納している．設立日については，ret2[] を用意し，各メンバに対応するよう格納させている．

3.3　各種コマンド実現部（124 行目から 207 行目）

40〜43 行目の各種コマンド実現に必要な関数群は，cmd_処理名 という名前に統一することで，関数であることを明確にした．コマンド%P は cmd_print()，コマンド%C は cmd_check() にそれぞれ対応している．

124–144 行目は，%P，%C，%Q のコマンドを解釈して適切な関数を呼び出す部分である．

%P に対応する関数 cmd_print() の処理内容としては，154 から 207 行目に記載してある．内容は，表 1 に記載した．

%C，%Q はそれぞれ，146 行目からと 150 行目からに処理内容を記述した．

3.4　解析部（113 行目から 122 行目）

53 から 60 行目は main() 関数で，113 から 122 行目は，parse_line() 関数であり，作成方針で説明した解析部の動作におおよそ相当する．ただし (c) の つの文字列に分割する部分は，解析部の main() 関数では実現せず，処理内容を明確にするために変換部である new_profile() 関数中で split を呼出すことにしている．

4　プログラムの使用法

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

プログラムは， で動作を確認しているが，一般的な UNIX で動作することを意図している．gcc でコンパイルした後，標準入力から入力ファイルを与える．

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

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

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

以下のような出力を得る．

```
3 profile(s)
Id    : 111
Name  : The Bridge
Birth : 1845-11-02
Addr  : Okayama
Com.  : SEN Unit 2.0 Open

Id    : 222
Name  : Bower School
Birth : 1908-01-19
```

```
Addr  : Kagawa
Com.  : 01955 641225 Primary 25 2.6 Open
```

入力中の%C はこれまでの入力データの件数を表示することを示し，%P 2 は入力したデータのうち，先頭から 2 件分のデータを表示することを示している．なお，%Q はシステムを終了することを示す．

5 作成過程における考察

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

5.1 関数 split についての考察

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

5.2 関数 get_line についての考察

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

5.3 関数 new_profile についての考察

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

6 結果に関する考察

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

1. 不足機能についての考察
2. エラー処理についての考察

6.1 不足機能についての考察

不足機能については，以下の内容が考えられる．

1. データの削除についての機能
2. 入力したデータの並び替えに関する機能
3. 指定語句を検索し，その結果に合致するデータの表示
4. なんの機能があるか表示する `help` のような機能

標準入力から受け付けたデータを編集したい場合に備え，誤って入力したデータを削除できる機能が必要だと考えた．指定のデータ番号を削除もしくは，入力した直近のデータの削除の 2 通りが考えられる．

また，並び替えの機能についてだが，仮に実装する場合，名簿データの構成上，ID 順，学校名順，設立日順，住所順に並び替える機能が必要だと考えた．さらに，各要素について，降順と昇順に並び替えることも必要になるだろう．

登録できるデータ件数が 10000 件と仕様にあるため，必然的にデータ数が多くなれば管理も複雑になることが多くなる．そこで，登録したデータを容易に検索できるシステムがあればさらに便利に当システムを利用することが可能になるのではないかと考えた．なお，これらの機能を見れる機能も実装したい．

6.2 エラー処理についての考察

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

6.2.1 CSV データ処理中のエラー処理

CSV データ中に，不正なデータが含まれていた場合の処理について考察する．エラーが含まれていた場合は，以下のような対処が考えられる．

(1) エラーのあった行を指摘して，無視する

1 度の入力で，エラーを多く見つけることができるようになるため，エラー処理を実装する際はこの方法をとることが望ましいと考える．しかし，そのエラー後の入力へどのように復帰させるのかといったことや，エラーのあった入力文字列の扱いについて考える必要があるため，プログラムが複雑になることが懸念される．

(2) エラーのあった行を指摘して，終了する

この方法は，入力中にいずれかの 1 つのエラーを発見することしかできない．しかし，エラーのあった入力をデータを無視してしまうと以降のデータ入力の内容を確認する際の処理にも影響がでるような場合には，この方法を採らざるを得ないこともあると考えられる．

エラーのあった行を指摘せず，終了または無視するという方法も考えられるが，正常終了との区別が付かない上に，どの状態でエラーが発生しているのか確認をとることができないため実用的でないと考えた．

今回は，エラーのあった行を指摘して，無視する方法がよいと考えた．現時点ではエラー処理については未実装であるため，プログラム中に error: で始まるコメントを書いて検討の目印としている．また，エラーのあった内容を指摘するためには，enum という機能を利用してどのエラーなのかユーザが一度見て理解できるように，標準エラー出力を利用してエラーの内容を表示させるのが良いだろう．

6.2.2 split 関数におけるエラー処理

関数 split におけるエラー処理としては，規定分割数に達しない場合もしくは，規定分割数を超えた場合の処理が考えられる．実装方法として例えば，ここでは返り値として変数 count を返しているが，この前に変数 count が規定分割数を超えているか否かという分岐処理を行い，超えていれば -1 を，達していなければ -2 を返し，その値に応じて別関数もしくは同関数内でエラー表示を行わせることが考えられる．

6.2.3 cmd_print 関数におけるエラー処理

関数 cmd_print は，入力された変数 param が profile_data_nitems より多いと全件表示させているが，今後はこれもエラーとしたい．何件入っているかに関わらず処理を行わせると，cmd_check でデータ件数を確認する理由がなくなってしまうと考えているからである．

また，データ件数が 0 件の場合，セグメンテーションフォールトなどは起こらないが，せめてデータが入っていないという表示を行わせるべきであると考える．

6.2.4 new_profile 関数におけるエラー処理

まず，関数 new_profile は，データ形式が正しいか否かに関わらずデータ件数をインクリメントさせているため，データ形式が不正な場合はインクリメントさせないようにするべきである．new_profile への引数として渡している構造体配列の変数を変更し，返り値を返す際に成功していればインクリメントさせる方法が考えられる．

この実装の場合，不正なデータ形式が入力ときは NULL を返す処理を追加しなくてはならない．不正なデータとして考えられるのは，受け取った CSV データが 5 分割でない場合，設立日が-で区切られていない場合，そして，ID が数字でない場合が考えられる．その他はエラーにしなくともよいだろう．

7 感想

課題が与えられた際は，実装方針が全くわからず完成するか不安だった．しかし，いきなりプログラムを組むのではなく，日本語で段階的に流れを組み，徐々に詳細化していき，プログラムをしていくという方法を学んだため，頭でイメージを立てながらプログラムを組むことができた．しかし，メモリ使用量などデータ構造についてはさらに検討の余地があると感じる．また，C 言語は，オブジェクト指向型言語ではないがポインタや構造体を用いることでこれに近い動きをできることに驚いた．しかし，文字列の代入や値の受け渡しについては最近の言語とは異なること

が多いように感じた．今回の課題を通して，ポインタと構造体に関する理解を深められたように思うが，まだまだ足りないため，考察内容を実装する中でさらに理解を深められるようにしたい．

8 作成したプログラム

作成したプログラムを以下に添付する．

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <string.h>
 4
 5  #define LIMIT 1024
 6  #define maxsplit 5//最大分割数
 7  #define luck -1
 8  #define over -2
 9  #define endp NULL//strtol 用ポインタ
10  #define base1 10//10 進数
11
12  struct date {
13      int y;//year
14      int m;//month
15      int d;//day
16  };
17
18  struct profile{
19      int id;//id
20      char name[70];//schoolname
21      struct date found;
22      char add[70];//address
23      char *others;//備考
24  };
25
26  /*subst*/
27  int subst(char *str,char c1,char c2);
28
29  /*split*/
30  int split(char *str,char *ret[],char sep,int max);
31  void error_split(int check);
32
33  /*get_line*/
34  int get_line(char *input);
35
36  /*parse_line*/
37  void parse_line(char *line);
38
39  /*cmd*/
40  void cmd_quit();
41  void cmd_check();
42  void cmd_print(struct profile *pro,int param);
43  void exec_command(char cmd, char *param);
44
45  /*profile*/
46  struct profile *new_profile(struct profile *pro,char *str);
47
48  /*GLOBAL*/
49  struct profile profile_data_store[10000];
50  int profile_data_nitems = 0;
51
52  /*MAIN*/
53  int main(void){
```

```
54
55      char line[LIMIT + 1];
56      while (get_line(line)) {//null でない限り続ける
57          parse_line(line);
58      }
59      return 0;
60  }
61
62  int subst(char *str,char c1,char c2){
63      int count = 0;
64      while(*str != '\0'){
65          if(*str == c1){//指定文字なら変換
66              *str = c2;
67              count++;
68          }
69          str++;
70      }
71      return count;
72  }
73
74  int split (char *str,char *ret[],char sep,int max){
75      int count = 0;//分割数
76
77      while (1) {
78          if(*str == '\0') {
79              break;//からもじなら抜ける
80          }
81
82          if(count>max)break;
83          ret[count++] = str;
84          //str をいじれば ret も変わるように分割後の文字列にはポインタを入れる
85
86          while( (*str != '\0') && (*str != sep) ){
87              //区切り文字が見つかるまでポインタすすめる
88              str++;
89          }
90
91          if(*str == '\0') {
92              break;
93              //区切り文字がなかったら抜ける＝文字列はそのまま
94          }
95
96          *str = '\0';
97          //必ず区切り文字のはずだからくぎる
98              str++;
99          //インクリメントさせる
100     }
101     //error:カウント数が大きいもしくは、規定カウント数以下の場合エラー
102     return count;
103 }
104
105 int get_line(char *input){
106     if (fgets(input, LIMIT + 1, stdin) == NULL){
107         return 0; /* 失敗 EOF */
108     }
109     subst(input, '\n', '\0');
110     return 1; /*成功*/
111 }
112
113 void parse_line(char *line){
114     if(line[0]=='%'){
115         exec_command(line[1], &line[3]);
```

```
116          //[0]->% [1]->P [2]->空白 [3]->2 など
117      }
118      else{
119          new_profile(&profile_data_store[profile_data_nitems++],line);
120          //error:データが不正でもインクリメントされている
121      }
122  }
123
124  void exec_command(char cmd, char *param)
125   {
126      switch(cmd){
127          case 'Q':
128              cmd_quit();
129          break;
130
131          case 'C':
132              cmd_check();
133          break;
134
135          case 'P':
136              cmd_print(&profile_data_store[0],strtol(param,endp,base1));
137          break;
138
139          default:
140              fprintf(stderr,"Invalid command %c: ignored.\n", cmd);
141              //定義されていないコマンド
142          break;
143      }
144   }
145
146  void cmd_quit(){
147      exit(0);
148      return;
149  }
150  void cmd_check(){
151      printf("%d profile(s)\n",profile_data_nitems);
152      return;
153  }
154  void cmd_print(struct profile *pro,int param){
155      if(profile_data_nitems == 0){
156          //error:データがないことを示す
157          return ;
158      }
159      int i;
160
161      if(param == 0){//０のとき
162          for(i=0;i<profile_data_nitems;i++){
163              printf("Id    : %d\n",(pro+i)->id);
164              printf("Name  : %s\n",(pro+i)->name);
165              printf("Birth : %04d-%02d-%02d\n",(pro+i)->found.y,(pro+i)->
found.m,(pro+i)->found.d);
166              printf("Addr  : %s\n",(pro+i)->add);
167              printf("Com.  : %s\n",(pro+i)->others);
168              printf("\n");
169          }
170      }
171
172      else if(param > 0){//正のとき
173
174          if( param > profile_data_nitems ){
175              param=profile_data_nitems;
176              /*error:すでに配列に入ってる要素よりも大きい場合は全件表示するよりも、
```

```
   177              データ数オーバーでエラーにするほうが良いのでは？
   178              負の時も同様*/
   179          }
   180          for(i = 0;i<param;i++){
   181              printf("Id    : %d\n",(pro+i)->id);
   182              printf("Name  : %s\n",(pro+i)->name);
   183              printf("Birth : %04d-%02d-%02d\n",(pro+i)->found.y,(pro+i)->
found.m,(pro+i)->found.d);
   184              printf("Addr  : %s\n",(pro+i)->add);
   185              printf("Com.  : %s\n",(pro+i)->others);
   186              printf("\n");
   187          }
   188      }
   189
   190      else if(param < 0){//負の時
   191
   192          param *= -1;
   193          if( param > profile_data_nitems ){
   194              param=profile_data_nitems;
   195          }
   196          pro += profile_data_nitems-param;
   197          for(i=0 ;i<param;i++){
   198              printf("Id    : %d\n",(pro+i)->id);
   199              printf("Name  : %s\n",(pro+i)->name);
   200              printf("Birth : %04d-%02d-%02d\n",(pro+i)->found.y,(pro+i)->
found.m,(pro+i)->found.d);
   201              printf("Addr  : %s\n",(pro+i)->add);
   202              printf("Com.  : %s\n",(pro+i)->others);
   203              printf("\n");
   204          }
   205      }
   206      return;
   207  }
   208  struct profile *new_profile(struct profile *pro,char *str){
   209      char *ret1[maxsplit],*ret2[maxsplit-2];
   210
   211      if(split(str,ret1,',',maxsplit)!=maxsplit){
   212          return NULL;
   213      }//文字列用
   214
   215      //error:id が数字ではないときは形式が違うからエラーにしたい
   216      pro->id = strtol(ret1[0],endp,base1);
   217
   218      strncpy(pro->name, ret1[1],70);//名前のコピー
   219      pro->name[70]='\0';
   220
   221      if(split(ret1[2],ret2,'-',maxsplit-2)!=maxsplit-2){
   222          return NULL;
   223      }//設立日
   224      pro->found.y = strtol(ret2[0],endp,base1);
   225      pro->found.m = strtol(ret2[1],endp,base1);
   226      pro->found.d = strtol(ret2[2],endp,base1);
   227
   228      strncpy(pro->add, ret1[3],70);//住所
   229      pro->add[70]='\0';
   230
   231      pro->others = (char *)malloc(sizeof(char)*(strlen(ret1[4])+1));
   232      strcpy(pro->others, ret1[4]);//備考
   233
   234      return pro;//構造体 pro を返す
   235  }
```