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 xx 月 xx 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．名簿管理プログラムとして，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．また，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．

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．

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．

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

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ーに代入する部分である．メンバー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 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@@@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（@@行目から@@行目）

まず，汎用的な文字列操作関数として，subst() 関数を@@–@@行目で宣言し,split() 関数を@@ーー@@行目で宣言，さらに get_line() 関数を宣言している．

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（@@行目から@@行目）

@@～@@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．

@@～@@行目は struct profile 型の宣言部，@@～@@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．詳しい考察は後述にある．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. 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（@@行目から@@行目）

@@～@@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．

@@–@@行目は，%P，%S，%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，引数として構造体配列の先頭と変数 param に応じて表示内容を変化させる．内容は，表 1 に記載した．

3.4　解析部（@@行目から@@行目）

@@@行目は，main() 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，変換部である new_profile() 関数中で split を呼出すことにしている．この理由については，考察にて後述する．

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　標準入力からの入力についての考察

… の作成方針として … としたため，… となっていることには注意が必要である．なぜなら，….しかし，… であるため，… である．例えば，… としたいのであれば，…. とすればよい．

（※サンプルのため省略）

5.3　関数 new_profile についての考察

… については … という方針にしたが，… という方針にすることも考えられる．今回は … ということを考えたため，… とすることにした．ただし，もし … であるならば，… は … よりも … であるから，… という実装方針とするほうがよいだろう．

（※サンプルのため省略）

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
3. 構造体 struct person のサイズに関する考察

6.1　不足機能についての考察

指定データの削除なんのコマンドがあるかを示す help コマンドの実装

6.2　エラー処理についての考察

(※サンプルのため中略)

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
この方法は，一回の入力で，できるだけ多くのエラーを発見できるため，通常はこの方法が好ましい．しかし，エラーのあった状態からの復帰を行う必要があるためプログラムが複雑になる．

(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 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正当性チェック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．

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ため実用的でない．

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が，プログラム中でエラーチェックすべき部分に `ToDo:`で始まるコメントを入れて改版時の目印となるようにしている．

また，エラーのあった行を指摘するためには，…

(※サンプルのため省略)

6.2.2　… 関数におけるエラー処理

meibopro.c のエラー処理の内容を書けばいい

(※サンプルのため省略)

6.2.3　…

(※サンプルのため省略)

6.3 struct person のサイズ

以下のプログラムを gcc でコンパイルして実行して，sizeof(struct person) = 88 という結果を得た．

(※サンプルのため省略)

```
 1 my_person_data:
 2         .long 999          ← 4 バイト code
 3         .string "taro"     ← 5 バイト name (末尾のヌルを入れて)
 4         .zero   25         ← name が 30 になるように 0 を埋める
 5         .zero   2          ← **** 2 バイトの調整用 padding
 6         .long 2000         ← 4 バイト date.y = 2000
 7         .long 1            ← 4 バイト date.m = 1
 8         .long 1            ← 4 バイト date.d = 1
 9         .byte 0            ← 1 バイト type
10         .string "Kagawa"   ← 7 バイト home (末尾のヌルを入れて)
11         .zero   23         ← home が 30 になるように 0 を埋める
12         .zero   1          ← **** 1 バイトの調整用 padding
13         .long 180          ← 4 バイト height = 180
14         .long 75           ← 4 バイト weight = 75
```

これを見ると，5 行目と 12 行目に合計で 3 バイトの padding が入っていることが分かる．よく見ると .long つまり int の前に必ず入って 4 バイト境界に整数が跨がらないようになっている．しかし，10 行目の home のように，文字列の場合は，もともと 1 バイト単位で処理することが前提のためか，特に 4 バイト境界に納めるような padding は行われていない．(つまり 9 行目と 10 行目の間に .zero 3 が入らない)

家電の中に組込まれているマイコンなどのメモリが少ないマシンにおいては，実行効率よりもこのメモリの無駄使いが問題になることがある．そのような場合に対応するため，gcc はこの padding を止めるオプションを提供している．同じコードを -fpack-struct オプションを付けてコンパイルすると，メモリを無駄に使用しない，以下のコードを生成する．

```
 1 my_person_data:
 2         .long 999
 3         .string "taro"
 4         .zero   25
 5         .long 2000
 6         .long 1
 7         .long 1
 8         .byte 0
 9         .string "Kagawa"
10         .zero   23
11         .long 180
12         .long 75
```

(※サンプルのため省略)

7 感想

(※サンプルのため省略)

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．与えられた課題については，… 節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．

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <string.h>
 4
 5  #define MAX_LINE_LEN 256
 6
 7  /*****************************************************************/
 8  /* string fuctions                                               */
 9  /*****************************************************************/
10
11  /* substitute C1 to C2 in string SP.   */
12  /* return value: number of replacement */
13  int subst(char *sp, char c1, char c2)
14  {
15    int n = 0;
16
17    while (*sp){
18      if (*sp == c1){
19        *sp = c2;
20        n++;
21      }
22      sp++;
23    }
24    return n;
25  }
26
27  /* split STR using delimiter char C        */
28  /* return value: number of split elements  */
29  /* ret[] points each split element         */
30  int split(char *str, char *ret[], char c, int max)
31  {
32    int cnt = 0;
33
34    ret[cnt++] = str;
35
36    while (*str && cnt < max){
37      if (*str == c){
38        *str = '\0';
39        ret[cnt++] = str + 1;
40      }
41      str++;
42    }
43    return cnt; /* not more than MAX */
44  }
45
```

(※サンプルのため中略)

```
264      }
265    }
266    return 0;
267  }
```