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: 20xx 年 xx 月 xx 日
提出日: 20xx 年 xx 月 xx 日
締切日: 20xx 年 xx 月 xx 日

1 概要

本演習では，外部からの入力データを計算機で扱える内部形式に変換して格納し，それらを操作する方法について学習する．具体的には，標準入力から与えられる名簿の CSV データを C 言語の構造体の配列に格納し，それらをソートして表示するプログラムを作成する．

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. 基本仕様
   (a) 標準入力から「コード，氏名，生年月日，性別，出身，身長，体重」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．

```
0,Takahashi Kazuyuki,1977-04-27,3,Saitama,184,78
10,Honma Mitsuru,1972-08-25,2,Hokkaidou,180,78
11,Ogura Shinsuke,1976-07-23,0,Kanagawa,177,72
13,Shibata Kazuya,1968-03-16,0,Hyougo,176,75
:
```

   (b) 標準入力から%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機能を持つ．実装するコマンドを表??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 100 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 person` の配列 `person_array[]` を宣言して，100 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 person` 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?? 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
%Sn	CSV の n 番目の項目でソート	n: 1–7
%Pn	CSV の n 番目を抜き出して表示	n: 0–99 (0 は全て)

表 2: 名簿データ

コード	氏名	生年月日	性別	出身	身長	体重
32 bit 整数	20 bytes	struct date	char	10 bytes	32 bit 整数	32 bit 整数

1. 不足機能についての考察をおこなった．特に，…（サンプルのため省略）
2. エラー処理についての考察をおこなった．例えば，…（サンプルのため省略）
3. 構造体 struct profile がメモリ中を占めるバイト数について確認をおこなった．
   具体的には，sizeof 演算子を使用して …（サンプルのため中略）… 確認をおこなった．

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（**??**節）
2. 標準入力から得た CSV データの解析部（ … 節）
3. 構文解析したデータの内部形式への変換部（ … 節）
4. 各種コマンド実現部（ … 節）

2.1 宣言部

“宣言部” は … をする部分である．このレポートでは概要で示した表**??**に基づいて，以下のように宣言する．

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

struct person {
  int code;
  char name[20];
  struct date bday;
  char type;
  char home[10];
  int height;
  int weight;
};

struct person person_array[100];
```

ここで，… については … としている．これによって，… とすることができる．
（※サンプルのため省略）

2.2 解析部

“解析部” は … をおこなう箇所である．しかし，このままでは，… であるため，… 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い，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 CSV とみなし’,’ を区切りとして 7 つの文字列に分割する．

(d) 分割してできた 7 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

ここで，… は … と詳細化することもできるが，今回は … とする．また，… で扱う文字列は … として処理するため，解析部に続く … では … に注意する必要がある．

（※サンプルのため省略）

2.3 変換部

“変換部” は分割された CSV 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 `xxxxxx` を用いて代入する．数値の場合，関数 `xxxx` を用いて文字列を … してから代入する．構造体 `struct date` であるメンバー `bday` については … してから代入する．

なお，構造体への代入については，··· 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 … し，… によって … しつつ格納するという処理は，… を … する処理と同じ処理である．従って，区切り文字が CSV の’,’ とは異なる … 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揮発性であることにも注意する．つまり，変換部で文字列を処理する際には … を … するのではなく，関数 … を使って … を行わなければならないことに気をつける必要がある．

（※サンプルのため省略）

2.4 各種コマンド実現部

“各種コマンド実現部” は … 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 … を実装している．

表示（`%Pn`）は `printf` で各項目毎に表示すればよい．ただし，… であることに注意が必要である．また，実装中に … ということがわかったため，… のように実装をすることにしている．この実装に関する方針決定の詳細は後の xxxx 節で説明する．

ソートは，C の標準関数である `qsort()` を使用することにする．構造体の各メンバー毎にソートをするために，7 つの比較関数を用意する．7 つの比較関数へのポインタを要素とする配列を宣言することによって項目毎のソートを見通しよく行えるようにする．

（※サンプルのため省略）

3　プログラムおよびその説明

プログラムリストは … 節に添付している．プログラムは全部で 267 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（11 行目から 44 行目）

まず，汎用的な文字列操作関数として，`subst()` 関数を 11–25 行目で宣言し， split() 関数を 27-44 行目で宣言している．

`subst` は，`sp` 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 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．

（※サンプルのため省略）

3.2　… 部（93 行目から 214 行目）

46–91 行目は struct date データ型の宣言部とそれを扱う関数群である．

93–214 行目は struct person データ型の宣言部とそれを扱う関数群である．文字列から各データ型への変換を担う関数は，名前を new_データ型 とすることで，変換部であることを明確にした．

また，各種コマンド実現部の qsort に必要な比較関数群は，cmp_メンバー名 という名前に統一することで，比較関数であることを明確にした．

（※サンプルのため省略）

3.3　… 部（216 行目から 245 行目）

216–245 行目は，%P, %S 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．

（※サンプルのため省略）

3.4　… 部（247 行目から 267 行目）

247 行目以降は，`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 7 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_person()` 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．

（※サンプルのため省略）

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，概要の節を参照のこと．

プログラムは，Red Hat Linux 3.2.2-5 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 < csvdata.csv
```

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 cvsdata.csv に対して，

```
0,Takahashi Kazuyuki,1977-04-27,3,Saitama,184,78
10,Honma Mitsuru,1972-08-25,2,Hokkaidou,180,78
%S3
%P0
```

以下のような出力を得る．

```
code:   10
name:   Honma Mitsuru
bday:   1972/08/25
type:   2
home:   Hokkaidou
height: 180
weight: 78

code:   0
name:   Takahashi Kazuyuki
bday:   1977/04/27
type:   3
home:   Saitama
height: 184
weight: 78
```

入力中の%S3 は，これまでの入力データを 3 番目の項目 (生年月日) でソートすることを示し，%P0 は 入力した項目の全ての項目 (1–7) を表示することを示している．

（※サンプルのため省略）

5　作成過程における考察

… 節で述べた実装方針に基づいて，… 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　… についての考察

… については方針通りに実装することができたが，… については … となった．これは … に原因があると考えている．

（※サンプルのため省略）

5.2　… 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　… 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

(※サンプルのため省略)

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

(※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  }
```