プログラミング演習 1
第 2 回レポート

氏名: 池田 海斗 (IKEDA, Kaito)
学生番号: 09501502

出題日: 2020 年 05 月 13 日
提出日: 2020 年 05 月 13 日
締切日: 2020 年 05 月 20 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

2　プログラムの作成方針

※執筆上の注意：追加仕様については，書かなくても良い．書く場合は，次節の説明などでも，その仕様を参照しながら書くとよい．

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，・・・・
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，・・・

（a）プログラムの正常な終了

（b）登録された・・・

4. 標準入力からのユーザ入力を通して,,,

(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,・・・

※執筆上の注意：繰り返すが，本書は空想上の課題に対するレポートの執筆例である．書くべき内容としては，課題書等を読んだうえで適切に執筆すること

3　プログラムの説明

プログラムリストは 7 章に添付している．プログラムは全部で 27 行からなる．以下では，1 節の課題ごとに，プログラムの主な構造について説明する．

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 4-17 行目で宣言し， main() 関数を 39-52 行目に記述している． また，これらの関数で利用するために，stdio.h，stdlib.h，string.h のヘッダファイルを読み込んでいる．

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() 関数の中で，文字\n，，を終端文字\0 置き換えるために，この関数を呼び出している．この関数では，文字 C1 を文字 C2 に置き換えた回数を返り値とする．

split(STR，RET，SEP，MAX) 関数では，STR が指す文字列を，最大 MAX 個まで文字 SEP の箇所で区切り，配列 RET に格納していく．プログラム中では，get_line() 関数で取得した 1 行を，カンマで区切りで配列に格納する際に呼び出される．この関数では，配列に分割された個数を返り値とする．

```
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)	
%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
*****	（サンプルのため　省略）	

3.2 構造体や配列を用いた名簿データの定義

3.3 標準入力の取得と構文解析

get_line(LINE) 関数は，標準入力で読み込まれた名簿データを 1 行ずつ取り出し，LINE に代入する．読み込む行がない場合，文字列が 1024 文字を超えてバッファオーバーランをする場合，ユーザが改行コードを入力した場合に返り値 0 を返す．

parse_line(LINE) 関数では，get_line(LINE) 関数で読み込まれた 1 行がコマンドか入力データであるかで条件分岐を行う．コマンドの場合，コマンド名と引数に分けてその値を exec_command(CMD, PARAM) 関数に引き渡し，入力データであった場合は new_profile(LINE) 関数で表示を行う．

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，MacOS Catalina 10.15.4 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，MacOS Catalina 10.15.4 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 とは警告オプションを全て有効にするためのオプションであり，-o とは実行ファイルの名前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，コードの視認性を高めたり無駄なコードを省くことができ，他のソースコードの実行ファイルとの識別が容易である．

```
% gcc -Wall -o eop_03_09501502 eop_03_09501502.c
```

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データの入力を模擬するため，CSV ファイルを標準入力により与えることで，実行する例を示している．通常の利用においては，%R file によりデータを読み込む．

```
$ ./eop_03_09501502.out < csvdata.csv
```

プログラムの出力結果として，CSV データの各項目が読みやすい形式で出力される．例えば，下記の cvsdata.csv に対して，

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

以下のような出力が得られる．

```
 >ret[0] = '5100046'
 >ret[1] = 'The Bridge'
 >ret[2] = '1845-11-2'
 >ret[3] = '14 Seafield Road Longman Inverness'
 >ret[4] = 'SEN Unit 2.0 Open'
%P command is invoked with arg: 'contoso'
 >ret[0] = '5100224'
 >ret[1] = 'Canisbay Primary School'
 >ret[2] = '1928-7-5'
```

```
 >ret[3] = 'Canisbay Wick'
 >ret[4] = '01955 611337 Primary 56 3.5 Open'
%C command is invoked with no arg
```

まず，入力データについて説明する．コマンドは %P, %C, %Q の 3 つを記述している．また，この CSV ファイルには 2 件のデータを保存しており，それぞれの項目はカンマで区切られている．

また出力結果について，ret[*] は配列の番号とその中に格納している文字列を表している．

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 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. 標準入力の取得と構文解析

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

5.1.1 「subst 関数」に関する考察

ここでは subst 関数について考察を行う．練習問題の解答例を見てみると，main 関数内で複数の文字列を変換できるようにループさせており，また配列に保存した複数の文字を置換した結果も出力できるようにしてある．このことから，最終的に完成するプログラムでは，配列に保存された文字列を複数個 subst 関数にかけるのではないかと考察できる．

また，for 文で囲まれた部分に着目すると，他の言語での foreach 文に似たような動作をしていることがわかった．

5.1.2 「split 関数」に関する考察

次に，split 関数についての考察を行う．今回工夫したところは，カンマを全て終端文字に置き換えて split 関数に渡すことで，文字列の先頭にポインタを合わせることで一気に文字列をコピーできることである．その後 23 行目で，ポインタを文字数 +1(終端文字) 移動させることで，次の文字列の文頭にポインタを移動している．

5.2 「標準入力の取得と構文解析」に関する考察

5.2.1 「get_line 関数」に関する考察

get_line 関数についての考察を行う．今回重要なポイントとなってくるところは，fgets の最大文字数を 1024 ではなく 1026 に設定したところである．まず改行文字が格納されるので，1024 文字に +1 してある．また 1024 文字以上であることを確認するために，文字を 1 文字多く取得しそこに文字があるかで返り値を変えている．しかし，strlen 関数は終端文字をカウントしないので，strlen 関数の値が 1025 文字以上を含まないようにしている．

5.2.2 「parse_line 関数」に関する考察

parse_line 関数についての考察を行う．まずコマンドか否かの判別は，文字頭の値が % で始まるかどうかで行っている．これは，データ入力の場合文字列の始めは [0-9] で始まるので，上記の条件に含まれることはないからである．96 行目では，ポインタの 4 文字目以降からヌル終端文字までを配列に格納している．

6 感想

(※サンプルのため省略)

7 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．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．

```
 1: #include <stdio.h>
 2: #include <string.h>
 3:
 4: #define MAX_LINE_LEN 1024
 5:
 6: void cmd_quit(void) {
 7:     exit(0);
 8: }
 9:
10: void cmd_check(char cmd) {
11:     fprintf(stderr, "%%%c command is invoked with no arg\n", cmd);
12: }
13:
14: void cmd_print(char cmd, char *param) {
15:     fprintf(stderr, "%%%c command is invoked with arg: '%s'\n", cmd, param);
16: }
17:
18: void cmd_read(char cmd, char *param) {
19:     fprintf(stderr, "%%%c command is invoked with arg: '%s'\n", cmd, param);
20: }
21:
22: void cmd_write(char cmd, char *param) {
23:     fprintf(stderr, "%%%c command is invoked with arg: '%s'\n", cmd, param);
24: }
25:
26: void cmd_find(char cmd, char *param) {
27:     fprintf(stderr, "%%%c command is invoked with arg: '%s'\n", cmd, param);
28: }
29:
30: void cmd_sort(char cmd, char *param) {
31:     fprintf(stderr, "%%%c command is invoked with arg: '%s'\n", cmd, param);
32: }
33:
34: void exec_command(char cmd, char *param) {
35:     switch (cmd) {
36:         case 'Q': cmd_quit();   break;
37:         case 'C': cmd_check(cmd);  break;
38:         case 'P': cmd_print(cmd, param);  break;
39:         case 'R': cmd_read(cmd, param);   break;
40:         case 'W': cmd_write(cmd, param);  break;
41:         case 'F': cmd_find(cmd, param);   break;
42:         case 'S': cmd_sort(cmd, param);   break;
43:         default:
44:         printf("Unregistered Command Is Entered.\n");
45:     }
46: }
47:
48: int subst(char *str, char c1, char c2) {
49:     int diff = 0;
50:     char *p;
51:
52:     p = str;
53:     while (*p != '\0') {
54:         if (*p == c1) {
55:             *p = c2;
```

```
56:             diff++;
57:         }
58:         p++;
59:     }
60:     return diff;
61: }
62:
63: int split(char *str, char *ret[], char sep, int max) {
64:     int i, count = 0;
65:     subst(str, sep, '\0');
66:     for (i=0; i<max; i++) {
67:         ret[i] = str;
68:         str += strlen(str)+1;
69:         count++;
70:     }
71:     return count;
72: }
73:
74: int get_line(char *line) {
75:     if (fgets(line, MAX_LINE_LEN + 1, stdin) == NULL || strlen(line) > MAX_LINE_LEN || *line == '\n') {
76:         return 0;
77:     } else {
78:         subst(line, '\n', '\0');
79:         return 1;
80:     }
81: }
82:
83: void new_profile(char *line) {
84:     int i, max = 5;
85:     char *ret[80] = {0}, sep = ',';
86:     split(line, ret, sep, max);
87:     for (i = 0; i < max; i++) {
88:         printf(" >ret[%d] = '%s'\n", i, ret[i]);
89:     }
90: }
91:
92: void parse_line(char *line) {
93:     char cmd, param[80] = {0};
94:     if (*line == '%') {
95:         cmd = *(line+1);
96:         strcpy(param, line+3);
97:         exec_command(cmd, param);
98:     } else {
99:         new_profile(line);
100:    }
101: }
102:
103: int main(void) {
104:     char line[MAX_LINE_LEN + 1];
105:     while (get_line(line)) {
106:         parse_line(line);
107:     }
108:     return 0;
109: }
110:
```

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] 著者名，書名，出版社，発行年.

[3] WWW ページタイトル，URL，アクセス日.

※執筆上の注意：これらは書き方の例である．実際に，参考にした書籍等を見て書くこと．