プログラミング　第5回レポート

202111609　仲村和士

2022年7月8日

1　はじめに

今回は文字列操作が主となる課題である。C言語における文字列操作はすなわちポインタ操作なので、安全性が低く、どちらかというと実装よりもデバッグが大変である。そのため、gdbデバッガをうまく活用することが正しいコードへの近道となろう。

2　設問(1)

2.1　課題内容と方針

指示内容が多いので整理しておこう。

- 文字列を受け取り連結リストを返す関数csv2listの実装。
- csv2listで返されたlistが不要になった際にそのリストに割当てられたすべてのメモリを開放するfreelistの実装
- main関数では実装した関数を用いて各要素をコンマを用いずにわかりやすく表示する

方針も順番に考えよう。前回のレポートの設問(3)で文字列の連結リストを作成したのでそれをベースに始めることにする。まずはcsv2list関数だが、これには大きく2つの動作が必要である。1つ目はコンマを利用して文字列を分割すること、2つ目は分割した部分列を動的メモリにコピーしたうえでそのポインタをリストに入れること、である。後者は前回までの関数を利用すればいいので今回大事なのは前者である。区切り文字による文字列の分割にはいろいろな方法が考えられると思うが、いちばん最初に思いつくstrtok関数は講義資料に載っていないという理由で利用できない。そこで、本設問に適した分割を次の方法で行う。

1. コンマ付きの文字列のポインタ(CSVファイルの1行)を受け取る。

2. 前処理として、改行文字をヌル文字に置換する。

3. 分割した文字列の先頭を示すポインタ head を用意する。(最初は当然受け取った文字列の先頭を指している)

4. 先頭からコンマを探す、見つかったらヌル文字に置き換える。

5. head が示す文字列は分割された文字列の 1 つである。動的なメモリにコピーしてから連結リストに入れる。

6. head をヌル文字に置き換えた文字の 1 つ次の文字を指すようにする。

7. 続きからコンマを探す。以後文字列終了 (ヌル文字の登場) まで繰り返す。

次に freelist 関数を考える。list を与えたときに解放する必要があるのは

- 各ノードが参照する文字列
- 各ノード
- リスト

の 3 つであり、それを解放すればよい。

最後に、main 関数での表示であるが、これは printAllElements 関数を改造することにする。一つのリストにつきブレースで囲い、各要素は改行することで実現する。

2.2 実装

69 行目までは前回のレポートから引っ張ってきた連結リストとその操作関数の定義である。ただし、printAllElements 関数内の 63,67 行目では前の小節で述べたように、表示したときにブレースで囲われるようにだけ改造している。

70 行目からは freelist 関数であり、コメントにある通り、各 Element が参照する文字列、各 Element、連結リスト自体の 3 つを解放するようにした。

90 行目からは前回も利用した改行削除の chomp 関数なので説明は割愛する。

100 行目からは csv2list 関数である。変数について説明すると、i はループ変数、token は分割で得られた要素を保管するための動的に与えられたメモリのポインタが入る。head は探索中の部分文字列の先頭番地が入り、len_str は最初に与えられた文字列の長さ (ヌル文字を含む) を入れる。delim はデリミタであり、今回はカンマが入っている。line は返却するリストである。

110 行目で改行を削除してヌル文字を挿入している。112 行目の for で最初に与えられた文字列のすべての文字 (ヌル文字を含む) に関して探索する。現在の文字がデリミタもしくはヌル文字 (すなわち、与えられた文字列の終端) のとき、現在

の文字をヌル文字に置き換え、head を起点とした文字列として読み取る。読み取られた文字列は、動的なメモリにコピーされ、appendElement 関数でリストにそのポインタ token を入れる。そして、head を次の文字列の開始番地に変更する。これを文字列の終端まで繰り返す。

129 行目からの main 関数ではコマンドライン引数を確認してから、139 行目でファイルを読み込みモードで open している。145 行目から一行ずつ読み込みをしており、例外処理のあとに csv2list 関数で list をつくり、printAllElements 関数で表示し、freelist で解放している。すべての行の読み込みが終了したらファイルを close している。

Listing 1: s2111609-1.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #define BUFSIZE 100
5
6
7  /* define linked list(datatype: char*) */
8
9  struct Element {
10     char* val;
11     struct Element *prev;
12     struct Element *next;
13 };
14
15 struct LIST {
16     struct Element *h;
17     struct Element *t;
18 };
19
20 struct Element *getElement(char* s){
21     struct Element *p;
22
23     p = (struct Element*)malloc(sizeof(struct Element));
24     if (p == NULL){
25         printf("Memory allocation error\n");
26         exit(EXIT_FAILURE);
27     }
28
29     p->val = s;
30     p->prev = NULL;
31     p->next = NULL;
32     return p;
33 }
34
35 struct LIST *initList(){
```

```
36      struct LIST *l;
37
38      l = (struct LIST*)malloc(sizeof(struct LIST));
39      if (l == NULL){
40          printf("Memory Allocation Error\n");
41          exit(EXIT_FAILURE);
42      }
43
44      l->h = getElement(NULL);
45      l->t = l->h;
46
47      return l;
48  }
49
50  void appendElement(struct LIST *l, char* s){
51      struct Element *e;
52
53      e=getElement(s);
54      l->t->next = e;
55      e->prev = l->t;
56      l->t = e;
57  }
58
59
60
61  void printAllElements(struct LIST *l){
62      struct Element *e;
63      puts("{");
64      for(e=l->h->next; e != NULL; e = e->next){
65          printf("%s\n", e->val);
66      }
67      puts("}");
68  }
69
70  void freelist(struct LIST *l){
71      struct Element *now, *next;
72      now = l->h;
73      while(1){
74          /* free string */
75          if(now->val != NULL){
76              free(now->val);
77          }
78          /* free Element */
79          next = now->next;
80          free(now);
81          now = next;
82          if(now == NULL){
```

```
            break;
        }
    }
    /* free list */
    free(l);
}

/* string control */

char* chomp(char* s){
    int l = strlen(s);
    if(l>0 && s[l-1]=='\n'){
        s[l-1] = '\0';
    }
}


/* convert csv formatted string into linked list */

struct LIST* csv2list(char* str){
    int i;
    char* token;
    char* head = str;
    int len_str;
    char delim = ',';
    struct LIST* line = initList();
    chomp(str);
    len_str = strlen(str)+1;

    for(i=0; i<len_str; i++){
        if(str[i]== delim || str[i]=='\0'){
            str[i] = '\0';
            token = (char*) malloc(strlen(head)+1);
            if(token == NULL){
                puts("memory allocation error!");
                exit(EXIT_FAILURE);
            }
            strcpy(token, head);
            appendElement(line, token);
            head = &str[i+1];
        }
    }
    return line;
}


int main(int ac, char* av[]){
```

```
130     FILE *fp;
131     char buf[BUFSIZE];
132     struct LIST* line_list;
133 
134     if(ac != 2){
135         puts("invalid args!");
136         exit(EXIT_FAILURE);
137     }
138 
139     fp = fopen(av[1], "r");
140     if(fp==NULL){
141         perror("fopen");
142         exit(EXIT_FAILURE);
143     }
144 
145     while(1){
146         if(fgets(buf, BUFSIZE, fp) == NULL){
147             if(ferror(fp) != 0){
148                 perror("fgets");
149                 exit(EXIT_FAILURE);
150             }else{
151                 break;
152             }
153         }
154 
155         line_list = csv2list(buf);
156         printAllElements(line_list);
157         freelist(line_list);
158     }
159     fclose(fp);
160 }
```

2.3 確認

設問条件の (1-1)(1-2)(1-3) は 2.1 節から確認しつつすべて盛り込んできた。また、文字列は csv2list 関数内で動的なメモリに保管している。細やかな部分はデバッガで確認した。2 つの大きさの異なる CSV ファイルを与えて期待する実行結果が出ていることを確認しよう。2 つの CSV ファイルを以下のように定義した。

test1.csv

```
name,birthday,ID,belong
Yamada,0402,1,mast
Tanaka,0801,2,mast
Suzuki,1013,3,klis
Sato,1111,4,coins
```

test2.csv

```
name,birthday,ID,belong,HP,MP
Yamada,0402,1,mast,100,10
Tanaka,0801,2,mast,50,30
Suzuki,1013,3,klis,150,3
Sato,1111,4,coins,30,80
Lucy,0725,5,esys,40,60
Steven,0819,6,med,90,80
```

実行例 1

```
$ gcc s2111609-1.c -std=c89
$ ./a.out test1.csv
  {
  name
  birthday
  ID
  belong
  }
  {
  Yamada
  0402
  1
  mast
  }
  {
  Tanaka
  0801
  2
  mast
  }
  {
  Suzuki
  1013
  3
  klis
  }
  {
  Sato
  1111
  4
  coins
  }
```

実行例 2

```
$./a.out test2.csv
  {
  name
  birthday
  ID
  belong
  HP
  MP
  }
  {
  Yamada
  0402
  1
  mast
  100
  10
  }
  {
  Tanaka
  0801
  2
  mast
  50
  30
  }
  {
  Suzuki
  1013
  3
  klis
  150
  3
  }
-------- 以下略 --------
```

2.4　難しかった点など

細やかなデバッグが大変であった。デバッガがなければだいぶ厳しかったかもしれない。

3　設問 (2)

3.1　課題内容と方針

csv を HTML のテーブルに変換して、最低限の体裁の整った HTML ファイルに出力する課題である。

方針は、とにかく fprintf 関数で HTML を出力することになるが、main 関数が見やすいようにほかの関数にまとめていくことにする。とくに重要なのはリストを受け取って HTML のレコードをつくる関数である。1 行を tr タグで囲み、各要素を td タグで囲むというまとまりを出力すればよい。

3.2　実装

前問との変更点を説明する。124 行目から HTML の体裁を出力する関数群を定義している。これらの関数は最低限書き込むファイルのファイルポインタを受け取る。print_html_head 関数は title の文字列を受け取り、head 部分までの HTML を出力する。print_html_body_table 関数は見出しの文字列を受け取り、table タグ (開始) までを出力する。次の print_table_record 関数は printAllElements をもとにして、要素をたどる前と後に tr タグを構成している部分と、printf を fprintf にして、各要素を td で挟むようにフォーマットする部分を変更した。print_html_close 関数では table のレコードが出力された後に記述するべき終了タグを出力する。

main 関数ではコマンドライン引数を確認したあと、CSV ファイルを読み込みモード、html ファイルを書き込みモードで開き、先程定義した関数 print_html_head と print_html_body_table により table のレコードを記述する直前までの HTML を出力している。180 行目からは CSV の各行を読み込み、リストを作成し、一応 printAllElements 関数で標準出力をしたあと、print_table_record 関数により、レコードを出力している。その後、必要なくなったリストは解放している。この処理を繰り返し行っている。すべての行を出力したあとには print_html_close 関数により、終了タグを出力して、開いていた 2 つのファイルを close している。

Listing 2: s2111609-2.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #define BUFSIZE 100
```

```
5
6
7  /* define linked list(datatype: char*) */
8
9  struct Element {
10     char* val;
11     struct Element *prev;
12     struct Element *next;
13 };
14
15 struct LIST {
16     struct Element *h;
17     struct Element *t;
18 };
19
20 struct Element *getElement(char* s){
21     struct Element *p;
22
23     p = (struct Element*)malloc(sizeof(struct Element));
24     if (p == NULL){
25         printf("Memory allocation error\n");
26         exit(EXIT_FAILURE);
27     }
28
29     p->val = s;
30     p->prev = NULL;
31     p->next = NULL;
32     return p;
33 }
34
35 struct LIST *initList(){
36     struct LIST *l;
37
38     l = (struct LIST*)malloc(sizeof(struct LIST));
39     if (l == NULL){
40         printf("Memory Allocation Error\n");
41         exit(EXIT_FAILURE);
42     }
43
44     l->h = getElement(NULL);
45     l->t = l->h;
46
47     return l;
48 }
49
50 void appendElement(struct LIST *l, char* s){
51     struct Element *e;
```

```
52
53      e=getElement(s);
54      l->t->next = e;
55      e->prev = l->t;
56      l->t = e;
57  }
58
59
60
61  void printAllElements(struct LIST *l){
62      struct Element *e;
63      puts("{");
64      for(e=l->h->next; e != NULL; e = e->next){
65          printf("%s\n", e->val);
66      }
67      puts("}");
68  }
69
70  void free_list(struct LIST *l){
71      struct Element *now, *next;
72      now = l->h;
73      while(1){
74          /* free string */
75          if(now->val != NULL){
76              free(now->val);
77          }
78          /* free Element */
79          next = now->next;
80          free(now);
81          now = next;
82          if(now == NULL){
83              break;
84          }
85      }
86      /* free list */
87      free(l);
88  }
89
90  /* string control */
91
92  char* chomp(char* s){
93      int l = strlen(s);
94      if(l>0 && s[l-1]=='\n'){
95          s[l-1] = '\0';
96      }
97  }
98
```

```
99
100 /* convert csv formatted string into linked list */
101
102 struct LIST* csv2list(char* str){
103     int i;
104     char* token;
105     char* head = str;
106     int len_str;
107     char delim = ',';
108     struct LIST* line = initList();
109     chomp(str);
110     len_str = strlen(str)+1;
111
112     for(i=0; i<len_str; i++){
113         if(str[i]== delim || str[i]=='\0'){
114             str[i] = '\0';
115             token = (char*) malloc(strlen(head)+1);
116             strcpy(token, head);
117             appendElement(line, token);
118             head = &str[i+1];
119         }
120     }
121     return line;
122 }
123
124 /* HTML control */
125 void print_html_head(FILE* fp, char* title){
126     fprintf(fp, "<!DOCTYPE html>\n");
127     fprintf(fp, "<html>\n");
128     fprintf(fp,"<head>\n");
129     fprintf(fp, "<title>%s</title>\n", title);
130     fprintf(fp, "</head>\n");
131 }
132 void print_html_body_table(FILE* fp, char* heading){
133     fprintf(fp, "<body>\n");
134     fprintf(fp, "<h1>%s</h1>\n", heading);
135     fprintf(fp, "<table border=\"1\">\n");
136 }
137 void print_table_record(FILE* fp, struct LIST* l){
138     struct Element* e;
139     fprintf(fp, "<tr> ");
140     for(e=l->h->next; e != NULL; e = e->next){
141         fprintf(fp, "<td>%s</td> ", e->val);
142     }
143     fprintf(fp, "</tr>\n");
144 }
145 void print_html_close(FILE* fp){
```

```
146     fprintf(fp, "</table>\n");
147     fprintf(fp, "</body>\n");
148     fprintf(fp, "</html>\n");
149 }
150 
151 int main(int ac, char* av[]){
152     FILE *fp_csv, *fp_html;
153     char buf[BUFSIZE];
154     struct LIST* line_list;
155 
156     /* you need input & output file name */
157     if(ac != 3){
158         puts("invalid args!");
159         exit(EXIT_FAILURE);
160     }
161     /* open csv */
162     fp_csv = fopen(av[1], "r");
163     if(fp_csv==NULL){
164         perror("fopen_csv");
165         exit(EXIT_FAILURE);
166     }
167     /* open html as write mode */
168     fp_html = fopen(av[2], "w");
169     if(fp_html == NULL){
170         perror("fopen_html");
171         exit(EXIT_FAILURE);
172     }
173 
174     /* write html head */
175     print_html_head(fp_html, "csv2table");
176 
177     /* write html body/table(start) */
178     print_html_body_table(fp_html, "csv2table");
179 
180     /* convert csv to html table */
181     while(1){
182         if(fgets(buf, BUFSIZE, fp_csv) == NULL){
183             if(ferror(fp_csv) != 0){
184                 perror("fgets");
185                 exit(EXIT_FAILURE);
186             }else{
187                 break;
188             }
189         }
190         line_list = csv2list(buf);
191         printAllElements(line_list);
192         print_table_record(fp_html, line_list);
```

```
193             free_list(line_list);
194         }
195         /*close html tags*/
196         print_html_close(fp_html);
197         /*close file stream*/
198         fclose(fp_csv);
199         fclose(fp_html);
200     }
```

3.3 確認

本問ではほぼ fprintf を用いた HTML の出力の関数しかしていないのでまずは正しく HTML が出力されることを以下の実行例から確認しよう。要求されている体裁が整っており、閉じるべきすべてのタグが閉じられていて、すべての要素がテーブルになっていることが確認できる。一部枠からはみ出してしまうのでレポート内で折り返しているが、実際には tr の開始から終了までが 1 行で表示される。

また、実装では、open や fgets などの例外処理も正しくなされている。

実行例

```
$ gcc s2111609-2.c -srd=c89
$ ./a.out test1.csv test1.html
$ ./a.out test2.csv test2.html
```

test1.html

```
<!DOCTYPE html>
<html>
<head>
<title>csv2table</title>
</head>
<body>
<h1>csv2table</h1>
<table border="1">
<tr> <td>name</td> <td>birthday</td> <td>ID</td> <td>belong</td>
  </tr>
<tr> <td>Yamada</td> <td>0402</td> <td>1</td> <td>mast</td> </tr>
<tr> <td>Tanaka</td> <td>0801</td> <td>2</td> <td>mast</td> </tr>
<tr> <td>Suzuki</td> <td>1013</td> <td>3</td> <td>klis</td> </tr>
<tr> <td>Sato</td> <td>1111</td> <td>4</td> <td>coins</td> </tr>
</table>
</body>
</html>
```

test2.html

(略)(枠に収まりきらないため。添付しているのでそちらを参照)

3.4 難しかった点など

こちらは特になかった。

4 感想

今回の問題は考え方自体はそこまで複雑ではないが、非常にデバッグが大変であった。デバッガが使える環境を構築できたのは今後C/C++を書くときに大きなメリットとなるだろう。特に調べないといけない内容はなかったので参考文献はないが、VSCodeとgdbデバッガの出力は非常に参考になったのでそれをもって参考文献の代わりとする。