

プログラミング 第4回レポート

202111609 仲村和士

2022年7月4日

1 はじめに

今回は連結リストのデータ構造に関する問題である。連結リストは個人的には Java の Collection として実装されているものが一番最初に浮かぶが、実際に実装してみると、オブジェクト指向ではない C 言語では Java に比べて少々扱いにくい印象であった。今回は授業資料のほかに、C++で解説された書籍 [1] で全体的な知識を得た。日本語の書誌情報を含むので pbibtex を推奨する。

2 設問 (1)

2.1 課題内容と方針

双方向連結リストを扱いやすい関数にまとめるほか、末尾から辿って出力する関数を作る問題である。

方針としては、講義ノート (14) の図 9 より図 11 のほうが作成したい体裁に近いことから、図 11 を双方向リストに修正する方向で実装する。修正する部分は図 9 の解説を読めば十分である。

2.2 実装

図 11 に対する変更点を簡単に説明する。まず 8 行目で前の Element への参照を持つように変更している。次に、27 行目では next 同様に初期化している。53 行目では append する際に戻り方向の参照も設定するようにしている。

66 行目からの rprintAllElement 関数では、最後の要素 l->t から逆順に h にたどり着くまで要素の値を出力している。main では最後にこの関数と呼んでいる。

Listing 1: s2111609-1.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
```

```

3
4 #define BUFSIZE 100
5
6 struct Element {
7     int val;
8     struct Element *prev;
9     struct Element *next;
10 };
11
12 struct LIST {
13     struct Element *h;
14     struct Element *t;
15 };
16 /* create new element (value: int n)*/
17 struct Element *getElement(int n){
18     struct Element *p;
19
20     p = (struct Element*)malloc(sizeof(struct Element));
21     if (p == NULL){
22         printf("Memory allocation error\n");
23         exit(EXIT_FAILURE);
24     }
25
26     p->val = n;
27     p->prev = NULL;
28     p->next = NULL;
29     return p;
30 }
31
32 struct LIST *initList(){
33     struct LIST *l;
34
35     l = (struct LIST*)malloc(sizeof(struct LIST));
36     if (l == NULL){
37         printf("Memory Allocation Error\n");
38         exit(EXIT_FAILURE);
39     }
40
41     l->h = getElement(0);
42     l->t = l->h;
43
44     return l;
45 }
46
47 /* add new element */
48 void appendElement(struct LIST *l, int n){
49     struct Element *e;

```

```

50
51     e=getElement(n);
52     l->t->next = e;
53     e->prev = l->t;
54     l->t = e;
55 }
56
57
58 void printAllElements(struct LIST *l){
59     struct Element *e;
60
61     for(e=l->h->next; e != NULL; e = e->next){
62         printf("val = %d\n", e->val);
63     }
64 }
65
66 void rprintAllElements(struct LIST *l){
67     struct Element *e;
68     for(e=l->t; e != l->h; e = e->prev){
69         printf("val = %d\n", e->val);
70     }
71 }
72
73 int getint(){
74     char buf[BUFSIZE];
75     fgets(buf, BUFSIZE, stdin);
76     return atoi(buf);
77 }
78
79 int main(int ac, char* av[]){
80     struct LIST *list;
81     int i;
82
83     list = initList();
84     while(1){
85         printf("input a number (quit when 0): ");
86         i = getint();
87         if (i == 0){
88             break;
89         }
90         appendElement(list, i);
91     }
92
93     printAllElements(list);
94     puts("reverse");
95     rprintAllElements(list);
96

```

```
97  
98 }
```

2.3 確認

この設問の要求は、双方向連結リストの関数化、`rprintAllElements` 関数の実装、標準入力の値をリストに入れて出力することである。もともと単方向のリストが実装されていたので、`rprintAllElements` 関数で正しく逆順に辿れていることが確認できればよい。また、要素数が0のコーナーケース(となりうる状況)も一応確認する。以下の実行例から、正しく実装されていることが確認できる。

実行例

```
$ gcc s2111609-1.c
$ ./a.out
input a number (quit when 0): 1
input a number (quit when 0): 4
input a number (quit when 0): 8
input a number (quit when 0): 5
input a number (quit when 0): 1
input a number (quit when 0): 67
input a number (quit when 0): 356
input a number (quit when 0): 0
val = 1
val = 4
val = 8
val = 5
val = 1
val = 67
val = 356
reverse
val = 356
val = 67
val = 1
val = 5
val = 8
val = 4
val = 1
$ ./a.out
input a number (quit when 0): 0
reverse
```

2.4 難しかった点など

LinkedList 自体がそれなりに難しいと思うが、事前にしっかり理解してから注意深く実装したので問題なかった。

3 設問 (2)

3.1 課題内容と方針

受け取った入力を昇順になるように挿入する insertElement 関数を作る課題である。

方針としては、関数を分割する。すなわち、特定の Element のあとに Element を挿入する (ポインタを繋ぎ変える) 関数を別に作っておき、挿入する場所を探して挿入する。挿入する場所を探す際、それまでのノードはすでに昇順になっているという条件を用いれば、前から順番にリストの要素と挿入する要素の値を比較すればよい。

3.2 実装

前問との変更点は 57 行目からの insert 関数と insertElement 関数を追加した点である。insert 関数は引数に 2 つの Element のポインタ p, v をとり、p のあとに v を挿入する関数である。ここでは、p が末尾の要素の場合に p->next が NULL になることに注意する。60 行目の if では NULL 値の場合に 61 行目が問題になるのでそれを場合分けしている。67 行目の if では p が末尾の要素であったときに v を末尾に更新する処理である。

insertElement 関数では、新しい要素を挿入すべき場所をループで探して insert 関数を用いて挿入する。h を起点として順番に探して行く構造である。79 行目からの判定について説明すると、最初の if では、今見ている要素 seek の次の要素が NULL であるときには seek は末尾の要素であるから、最後に挿入する。else if では次の要素の値が、挿入する値よりも大きければ、現在見ている seek 要素の次に挿入する。これらの挿入処理は append する関数と insert 関数で分けることもできるが、t の更新処理まで行っている insert が万能なので結局同じように書ける。¹ それ以外のときにはまだ挿入する場所が見つからないので seek を次の要素にする。これを繰り返して正しい場所に挿入する。

main 関数ではそのまま append していく list1 と昇順になっている list2 を用意して比較できるようにした。

Listing 2: s2111609-2.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define BUFSIZE 100
5
6 struct Element {
```

¹or で繋いで一つの if で書くこともできるが、内容は同じなので好みの問題だろう。

```

7     int val;
8     struct Element *prev;
9     struct Element *next;
10 };
11
12 struct LIST {
13     struct Element *h;
14     struct Element *t;
15 };
16 /* create new element (value: int n)*/
17 struct Element *getElement(int n){
18     struct Element *p;
19
20     p = (struct Element*)malloc(sizeof(struct Element));
21     if (p == NULL){
22         printf("Memory allocation error\n");
23         exit(EXIT_FAILURE);
24     }
25
26     p->val = n;
27     p->prev = NULL;
28     p->next = NULL;
29     return p;
30 }
31
32 struct LIST *initList(){
33     struct LIST *l;
34
35     l = (struct LIST*)malloc(sizeof(struct LIST));
36     if (l == NULL){
37         printf("Memory Allocation Error\n");
38         exit(EXIT_FAILURE);
39     }
40
41     l->h = getElement(0);
42     l->t = l->h;
43
44     return l;
45 }
46
47 /* add new element */
48 void appendElement(struct LIST *l, int n){
49     struct Element *e;
50
51     e=getElement(n);
52     l->t->next = e;
53     e->prev = l->t;

```

```

54     l->t = e;
55 }
56
57 /* insert node v after node p */
58 void insert(struct LIST *l, struct Element *v, struct Element
    *p){
59     /* when p->next is NULL, NULL->prev causes Segmentation
        Error! */
60     if(p->next != NULL){
61         p->next->prev = v;
62     }
63     v->next = p->next;
64     p->next = v;
65     v->prev = p;
66     /* update tail element */
67     if(l->t == p){
68         l->t = v;
69     }
70 }
71
72 void insertElement(struct LIST *l, int n){
73     struct Element *e;
74     struct Element *seek;
75     seek = l->h;
76     e = getElement(n);
77     while(1){
78         /* when seek is the tail element */
79         if(seek->next == NULL){
80             insert(l, e, seek);
81             break;
82         /* when you detect the correct place to insert */
83         }else if(seek->next->val > n){
84             insert(l, e, seek);
85             break;
86         }else{
87             seek = seek->next;
88         }
89     }
90 }
91
92 void printAllElements(struct LIST *l){
93     struct Element *e;
94
95     for(e=l->h->next; e != NULL; e = e->next){
96         printf("val = %d\n", e->val);
97     }
98 }

```



```

99
100 void rprintAllElements(struct LIST *l){
101     struct Element *e;
102     for(e=l->t; e != l->h; e = e->prev){
103         printf("val = %d\n", e->val);
104     }
105 }
106
107 int getint(){
108     char buf[BUFSIZE];
109     fgets(buf, BUFSIZE, stdin);
110     return atoi(buf);
111 }
112
113 int main(int ac, char* av[]){
114     struct LIST *list1, *list2;
115     int i;
116
117     list1 = initList();
118     list2 = initList();
119     while(1){
120         printf("input a number (quit when 0): ");
121         i = getint();
122         if (i == 0){
123             break;
124         }
125         appendElement(list1, i);
126         insertElement(list2, i);
127     }
128
129     printAllElements(list1);
130     puts("sorted");
131     printAllElements(list2);
132
133
134 }

```

3.3 確認

重複要素や負数を含んだ入力に対して正しく対応できればよい。また、処理冒頭で要素数が0のときが必ずあるのであえて分けて検証する必要はあまりない。

実行例

```
$ gcc s2111609-2.c -std=c89
$ ./a.out
input a number (quit when 0): -5
input a number (quit when 0): -100
input a number (quit when 0): -10
input a number (quit when 0): -5
input a number (quit when 0): 7
input a number (quit when 0): -3
input a number (quit when 0): 100
input a number (quit when 0): 7
input a number (quit when 0): 0
val = -5
val = -100
val = -10
val = -5
val = 7
val = -3
val = 100
val = 7
sorted
val = -100
val = -10
val = -5
val = -5
val = -3
val = 7
val = 7
val = 100
```

3.4 難しかった点など

当初、NULL 値の場合分けに気づかず、Segmentation fault を起こしたが、割とすぐに発見できた。

4 設問 (3)

4.1 課題内容の方針

リストを文字列に対応させて入出力 (ただし、改行文字は削除する) を行う課題である。

基本的に指示通りの実装をすればよいが、文字列の受け取りは前回の課題を参考にする。

4.2 実装

まず、前問までの実装で本問には必要のない関数は削除した。また、int 型を入力していた部分を char* 型に変更した。printAllElements 関数では、%d を %s に変更して文字列として表示するように変更した。69 行目からは改行文字を除外するのに必要な chomp 関数を講義ノート (14) をもとに記述した。main 関数では buf で文字列を受け取り、その後に動的なメモリにコピーし、そのポインタをリストで管理している。

Listing 3: s2111609-3.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #define BUFSIZE 100
5
6 struct Element {
7     char* val;
8     struct Element *prev;
9     struct Element *next;
10 };
11
12 struct LIST {
13     struct Element *h;
14     struct Element *t;
15 };
16 /* create new element (value: char* s)*/
17 struct Element *getElement(char* s){
18     struct Element *p;
19
20     p = (struct Element*)malloc(sizeof(struct Element));
21     if (p == NULL){
22         printf("Memory allocation error\n");
23         exit(EXIT_FAILURE);
24     }
25 }
```

```

26     p->val = s;
27     p->prev = NULL;
28     p->next = NULL;
29     return p;
30 }
31
32 struct LIST *initList(){
33     struct LIST *l;
34
35     l = (struct LIST*)malloc(sizeof(struct LIST));
36     if (l == NULL){
37         printf("Memory Allocation Error\n");
38         exit(EXIT_FAILURE);
39     }
40
41     l->h = getElement("");
42     l->t = l->h;
43
44     return l;
45 }
46
47 /* add new element */
48 void appendElement(struct LIST *l, char* s){
49     struct Element *e;
50
51     e=getElement(s);
52     l->t->next = e;
53     e->prev = l->t;
54     l->t = e;
55 }
56
57
58
59 void printAllElements(struct LIST *l){
60     struct Element *e;
61
62     for(e=l->h->next; e != NULL; e = e->next){
63         printf("%s\n", e->val);
64     }
65 }
66
67 /* string control */
68
69 char* chomp(char* s){
70     int l = strlen(s);
71     if(l>0 && s[l-1]=='\n'){
72         s[l-1] = '\0';

```

```

73     }
74 }
75
76 int main(int ac, char* av[]){
77     struct LIST *list;
78     int i;
79     char buf[BUFSIZE];
80     char* line;
81
82     list = initList();
83     while(1){
84         if(fgets(buf, BUFSIZE, stdin) == NULL){
85             break;
86         }else{
87            .chomp(buf);
88         }
89         line = (char*) malloc(strlen(buf)+1);
90         if(line == NULL){
91             puts("memory allocation failure!");
92             exit(EXIT_FAILURE);
93         }
94         strcpy(line, buf);
95         appendElement(list, line);
96     }
97     puts("-----list content-----");
98     printAllElements(list);
99
100 }

```

4.3 確認

動的なメモリ割り当てと、改行文字の削除は実装している。実行例では入力内容が1行ずつ表示されることを確認すれば要件を満たす。

実行例

```
$ gcc s2111609-3.c -std=c89
$ ./a.out
abcd
1234
a
12334556
welcome
-----list content-----
abcd
1234
a
12334556
welcome
```

4.4 難しかった点など

特になかった。

5 感想

これは副次的なものではあるが、LinkedList の実装を通して、new キーワードの振る舞いや Java の ArrayList と LinkedList の違いを正しく理解することができた。また、設問 (3) では、C89 では使えないが generics の威力が実感できた。

参考文献

[1] 大槻兼資. 問題解決力を鍛える！ アルゴリズムとデータ構造. 講談社, 2020.