云南大学数学与统计学院
上机实践报告

课程名称： 运筹学实验	**年级：** 2015 级	**上机实践成绩：**
指导教师： 李建平	**姓名：** 刘鹏	
上机实践名称： 前期程序准备	**学号：** 20151910042	**上机实践日期：** 2018-03-14
上机实践编号： No.01	**组号：**	**上机实践时间：** 23:51

一、实验目的

复习编程平台和编程资源，回顾 C 语言的相关知识；
完成该实验，为后期的更进一步的实验做准备。

二、实验内容

1. 用 C 语言编制程序，解决所给出的问题；
2. 保留.c 程序与可执行文件，以便提交。

三、实验平台

Windows 10 1703 Enterprise（编程与编辑文稿）；
Microsoft© *Visual Studio* 2017 Enterprise （IDE）；
Ubuntu 17.10 x86-64（辅助编程）
Xshell 5 Build 1339。

四、实验记录与实验结果分析

1 题

给定两组数$\boldsymbol{a}=(a_1,\ a_2,\ \cdots,\ a_n)$和$\boldsymbol{b}=\{b_1,\ b_2,\ \cdots,\ b_n\}$，求

（1） 一组数$\boldsymbol{c}=(c_1,\ c_2,\ \cdots,\ c_n)$，其中$c_i=\frac{a_i}{b_i},\ i=1,\ 2,\ \cdots,\ n.$

（2） 求最小值及所有最小值的下标，其中最小值为$\min\left\{\frac{a_i}{b_i}\middle|\ b_i>0,\ i=1,\ 2,\ \cdots,\ n\right\}.$

Solution:

对于这个解释程序，我界面是这样的：在 shell 中通过调用可执行程序 div，输入两个字符串参数，然后程序自动输出$\boldsymbol{c}$与最小值及其位置。如下所示：

```
>> $ div ( -3.14,-2 ,256, 0 ,6,5,12121,4588,-89) (3.14, -1, 10333,3.2222,2,0,5633.2,168,78)
argument 1 is
(-3.14, -2.00, 256.00, 0.00, 6.00, 5.00, 12121.00, 4588.00, -89.00)
argument 2 is
(3.14, -1.00, 10333.00, 3.22, 2.00, 0.00, 5633.20, 168.00, 78.00)

The answer C = (-1.00 , 2.00 , 0.02 , 0.00 , 3.00 , NaN , 2.15 , 27.31 , -1.14 )
Minimal value is -1.14, position is 9

>> $
```

因为并没有 shell 接口，所以基本上是自己写一个 shell 来做这个与机器的交互。首先是清洗，把两个字符串进行 clean 重整，去除可能的空格之后，第一步是跳过第一个圆括号，同时把最后的圆括号变为逗号。这样一来就好多了，一个数跟

着一个逗号。（这里都是对一个字符串来说的，毕竟解释得了一个就能解释两个。）第二步就是分割，把这个字符串当作一块长条豆腐，每次从头部切一部分下来，治到切光。头部已经是好的了，所以一直切到遇到的第一个逗号，这个过程把逗号之前的东西，即可能出现的负号与小数点进行分类处理，符号的话直接跳过最后乘-1 即可，其余的数字符号与小数点就直接归入队列（其实是链表），与此同时，队列的头号元素，跟着一个索引 1，一直到队列的末尾，即遇到的第一个分号。这样的话，遍历一遍找到小数点的索引，利用坐标变换公式，把数字与小数点的距离转化为 10 的指数，就可以通过 pow 函数做出这个具体的数值。同时在整个过程中要注意保护头指针与 work 指针的归位。这个数一旦算出来，就交给动态数组保存，一直读到反斜杠 0，就算是读完了。这个过程一直保存，解释函数自己判断。当解释程序返回一个浮点数就归入，返回 NULL 就结束归入。

拿到了两个动态数组之后，就随心所欲了，这个程序简直不要太简单，有数字，有大小，C 很容易就做出来了。这里最好可以用一个指针数组来表示分母为 0 的 NaN 情况，毕竟 C 不能很简单地用一般浮点数组能表示的了。这个程序我还要想一想。

程序代码：

```
1   // filename: main.c
2
3   /* -*- coding: utf-8 -*-
4
5   Created on Wed Mar 14 19 : 10 : 28 2018
6
7   @author: LiuPeng
8   */
9
10  #include<stdio.h>
11  #include<stdlib.h>
12  #include<string.h>
13  #include<math.h>
14
15  // The following type is a container for creating a stack.
16  typedef struct char_LinkedList {
17      char_LinkedList *head;
18      char elements;          // partition must be integer less than 10
19      int times;              // 这是一个容器，放置一个数组，用指针作为头
20      char_LinkedList *next;
21  }char_LinkedList;
22
23  typedef struct Dynamic_Array {
24      double *A;              // 底层数组
25      int capacity;           // 底层数组的容量
26      int n;                 // 底层数组的占用量
27  }Dynamic_Array;
28
29  typedef struct Div {
30      double up;
31      double down;
32      double value;
33      char state[10];         // NaN or Negative，长度不定
34                             // 这个 state 必须是 malloc 而来的，坚决不能直接用
35  }Div;
36
```

```
37  typedef struct Div_Dynamic_Array {
38      Div *A;             // 底层结构体数组的头指针，不能动!
39      int capacity;       // 底层结构体数组的容量
40      int n;              // 底层数组的占用量
41  }Div_Dynamic_Array;
42
43  void Div_Resize(Div_Dynamic_Array *D) {
44      int i = 0;
45      Div *tmp = (Div *)calloc(2 * D->capacity, sizeof(Div));
46      if (tmp == NULL) {
47          printf("Cannot get memory, crash!\n");
48          return;
49      }
50      for (i = 0; i < D->capacity; i++) {
51          (tmp + i)->up = (D->A + i)->up;
52          (tmp + i)->down = (D->A + i)->down;
53          (tmp + i)->value = (D->A + i)->value;
54          strcpy((tmp + i)->state, (D->A + i)->state);           //不能简单复制，否则会内存出错
55      }
56      free(D->A);
57      D->A = tmp;
58      tmp = NULL;         // 避免野指针
59
60      D->capacity *= 2;
61  }
62
63  void Div_Append(Div_Dynamic_Array *D, Div e) {
64      if (D->n == D->capacity) {
65          Div_Resize(D);
66      }
67      (D->A + D->n)->up = e.up;
68      (D->A + D->n)->down = e.down;
69      (D->A + D->n)->value = e.value;
70      strcpy((D->A + D->n)->state, e.state);
71      D->n += 1;
72      //int i;
73      //for (i = 0; i <= D->n; i++) {
74      //  printf("%s\t", (D->A + i)->state);
75      //}
76      //printf("\n");
77  }
78
79  void Div_print(Div_Dynamic_Array *d) {
80      int i;
81      printf("The answer:\nC = (");
82      for (i = 0; i < d->n; i++) {
83          if (!strcmp((d->A + i)->state, "NaN")) {
84              printf("%s ", "NaN");
85          }
```

```
86          else {
87              double value = (d->A + i)->value;
88              printf("%2.2f ", value);
89          }
90          if (i == d->n - 1) {
91              printf("");
92          }
93          else {
94              printf(", ");
95          }
96      }
97      printf(")\n");
98  }
99
100 void Div_onArray(Dynamic_Array *a, Dynamic_Array *b, Div_Dynamic_Array *ans) {
101     if (a->n != b->n) {
102         printf("length should be the same.");
103         return;
104     }
105
106     int i;
107     for (i = 0; i < a->n; i++) {
108         if (*(b->A + i) == 0) {
109             Div tmp;
110             tmp.up = NULL;
111             tmp.down = NULL;
112             tmp.value = NULL;
113             char c[] = "NaN";
114             strcpy(tmp.state, c);
115             Div_Append(ans, tmp);
116         }
117         else {
118             if (*(b->A + i) < 0.) {
119                 Div tmp;
120                 tmp.up = *(a->A + i);
121                 tmp.down = *(b->A + i);
122                 tmp.value = tmp.up / tmp.down;
123                 char c[] = "Negative";
124                 strcpy(tmp.state, c);
125                 Div_Append(ans, tmp);
126             }
127             else {
128                 Div tmp;
129                 tmp.up = *(a->A + i);
130                 tmp.down = *(b->A + i);
131                 tmp.value = tmp.up / tmp.down;
132                 char c[] = "Normal";
133                 strcpy(tmp.state, c);
134                 Div_Append(ans, tmp);
```

```
135             }
136         }
137     }
138 }
139
140 void Resize(Dynamic_Array *D) {
141     int i = 0;
142     double *tmp = (double *)calloc(2 * D->capacity, sizeof(double));
143     if (tmp == NULL) {
144         printf("Cannot get memory, crash!\n");
145         return;
146     }
147     for (i = 0; i < D->capacity; i++) {
148         *(tmp + i) = *(D->A + i);
149     }
150     D->A = tmp;
151     D->capacity *= 2;
152 }
153
154 void Append(Dynamic_Array *D, double e) {
155     if (D->n == D->capacity) {
156         Resize(D);
157     }
158     *(D->A + D->n) = e;
159     D->n += 1;
160 }
161
162 Dynamic_Array *Quick_sort(Dynamic_Array *a) {
163
164     Dynamic_Array *less = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
165     less->A = (double *)calloc(1, sizeof(double));
166     if (!less) {
167         printf("Can't get memory!");
168         return NULL;
169     }
170     less->capacity = 1;
171     less->n = 0;
172
173     Dynamic_Array *more = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
174     more->A = (double *)calloc(1, sizeof(double));
175     if (!more) {
176         printf("Can't get memory!");
177         return NULL;
178     }
179     more->capacity = 1;
180     more->n = 0;
181
182     Dynamic_Array *eq = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
183     eq->A = (double *)calloc(1, sizeof(double));
```

```
184     if (!eq) {
185         printf("Can't get memory!");
186         return NULL;
187     }
188     eq->capacity = 1;
189     eq->n = 0;
190 
191     int i;
192     if (a->n <= 1) {
193         return a;
194     }
195     else {
196         /*double pivot = 1 / 3. * (*(a->A) + ;*/
197 
198         for (i = 0; i < a->n; i++) {
199             double pivot = *(a->A);
200             if (*(a->A + i) > pivot) {
201                 Append(more, *(a->A + i));
202             }
203             else {
204                 if (*(a->A + i) < pivot) {
205                     Append(less, *(a->A + i));
206                 }
207                 else {
208                     Append(eq, *(a->A + i));
209                 }
210             }
211         }
212     }
213     less = Quick_sort(less);
214     more = Quick_sort(more);
215     for (i = 0; i < eq->n; i++) {
216         Append(less, *(eq->A + i));
217     }
218     for (i = 0; i < more->n; i++) {
219         Append(less, *(more->A + i));
220     }
221     return less;
222 }
223 
224 void find(Div_Dynamic_Array *a) {
225 
226     Dynamic_Array *c = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
227     Dynamic_Array *d = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
228     c->A = (double *)calloc(a->n, sizeof(double));
229     c->capacity = a->n;
230     c->n = 0;
231 
232     int i = 0;
```

```
233     for (i = 0; i < a->n; i++) {
234         if (!strcmp((a->A + i)->state, "Normal")) {
235             Append(c, (a->A + i)->value);
236         }
237     }
238     d = Quick_sort(c);
239     double pivot = *(c->A + 0);
240     printf("min = %2.2f\n", pivot);
241     for (i = 0; i < a->n; i++) {
242         if (!strcmp((a->A + i)->state, "Normal") && a->A->value == pivot) {
243             printf("%d", i);
244         }
245     }
246     printf("\n");
247 }
248 
249 char *clean(char *string) {     // 已经后期优化，减去了字符串中所有的空格
250     char *head = string;
251     int count_space = 0;
252     while (*string == ' ' && *string != '\0') {
253         count_space += 1;
254         string += 1;
255     }
256     string = head;
257 
258     int len = 1;    // 有'\0'，所以要＋1
259     while (*string != '\0') {
260         len += 1;
261         string++;
262     }
263     string = head;
264 
265     char *ans = (char *)calloc(len - count_space, sizeof(char));
266     if (ans == NULL) {
267         printf("Can't get memory!\n");
268         return NULL;
269     }
270     char *ans_head = ans;
271 
272     while (*string != '\0') {
273         if (*string != ' ') {
274             *ans = *string;
275             ans++;
276         }
277         string++;
278     }
279     *ans = *string;
280     ans = ans_head;
281     string = head;
```

```
282
283     ans = ans + 1;
284     char *tmp;
285     for (tmp = ans; *tmp != '\0'; tmp++) {
286         if (*(tmp + 1) == '\0') {
287             *tmp = ',';
288         }
289     }
290     return ans;
291 }
292
293 char *cut(char *string) {
294     while (*string != ',') {
295         if (*string == '\0') {
296             return '\0';
297         }
298         string++;
299     }
300     return ++string;
301 }
302
303 // Put an new element into the stack
304 double get_Number(char *string) {
305     // 传递一个完整的 clean 过的字符串进来，按需切割头部，剩下的头作为新的头。
306     if (*string == '\0') {
307         return NULL;
308     }
309     double ans = 0.;
310     if (*string == '\0') {
311         return NULL;
312     }
313     if (*string != '-') {
314         char_LinkedList *work = (char_LinkedList *)malloc(sizeof(char_LinkedList));
315         if (work == NULL) {
316             printf("Can't get memory!\n");
317             return 0;
318         }
319         // container
320
321         char_LinkedList *head = work;
322         int i = 1;
323         while (*string != ',') {
324             work->elements = *string;
325             work->times = i;
326             work->next = (char_LinkedList *)malloc(sizeof(char_LinkedList));    // 申请
327             if (work->next == NULL) {
328                 printf("Can't get memory!\n");
329                 return 0.;
330             }
```

```
331             work = work->next;                                              // 移动
332             work->elements = NULL;
333             work->times = NULL;
334             string++;
335             i++;    // i 在后面还有用
336         }
337 
338         work->elements = *string;       // 逗号也要加上
339         work->times = NULL;             // 逗号的指数不能为有意义的
340 
341         string++;
342 
343         work = head;
344         int dot = 1;
345         int comma = 1;      // 逗号的用处
346         int dot_index = NULL;
347         while (work->elements != ',') {
348             if (work->elements == '.') {
349                 dot_index = dot;
350                 break;
351             }
352             work = work->next;
353             dot++;
354         }
355 
356         if (dot_index == NULL) {
357             dot_index = i;
358         }
359 
360         work = head;
361 
362         while (work->times != NULL) {
363             work->times = -1 * (work->times - dot_index);
364             work = work->next;
365         }
366 
367         work = head;
368 
369         while (work->elements != ',') {
370             if (work->elements == '.') {
371                 work = work->next;
372                 continue;
373             }
374             if (work->times > 0) {
375                 ans += pow(10, work->times - 1) * double(int(work->elements) - int('0'));
376                 work = work->next;
377             }
378             else {
379                 ans += pow(10, work->times) * double(int(work->elements) - int('0'));
```

```
380                 work = work->next;
381             }
382         }
383     }
384     else {
385         string = string + 1;
386         ans = -1 * get_Number(string);
387     }
388     return ans;
389 }
390 
391 void print(int n, Dynamic_Array *d) {
392     printf("argument %d is \n(", n);
393     int i;
394     for (i = 0; i < d->n - 1; i++) {
395         printf("%2.2f, ", *(d->A + i));
396     }
397     printf("%2.2f", *(d->A + i));
398     printf(")\n\n");
399 }
400 
401 int main(int argc, char *argv[]) {
402     //if (argc != 3) {
403     //    printf("This function needs and only needs 2 arguments.\n");
404     //    return 0;
405     //}
406     //
407     //char *string_1 = *(argv + 1);
408     //char *string_2 = *(argv + 2);
409 
410     char string_1_tmp[] = "( -3.14,-2 ,256, 0 ,6,5,12121,4588,-89)";
411     char *string_1 = string_1_tmp;
412 
413     char string_2_tmp[] = "(3.14, -1, 10333,3.2222,2,0,5633.2,168,78)";
414     char *string_2 = string_2_tmp;
415 
416     string_1 = clean(string_1);
417     string_2 = clean(string_2);
418 
419     Dynamic_Array c_1, c_2;
420     c_1.A = (double *)malloc(sizeof(double));
421     if (c_1.A == NULL) {
422         printf("Can't get memory!\n");
423         return 0;
424     }
425     c_1.capacity = 1;
426     c_1.n = 0;
427 
428     c_2.A = (double *)malloc(sizeof(double));
```

```
429     if (c_2.A == NULL) {
430         printf("Can't get memory!\n");
431         return 0;
432     }
433     c_2.capacity = 1;
434     c_2.n = 0;
435 
436     while (string_1 != '\0') {
437         Append(&c_1, get_Number(string_1));
438         string_1 = cut(string_1);
439     }
440 
441     while (string_2 != '\0') {
442         Append(&c_2, get_Number(string_2));
443         string_2 = cut(string_2);
444     }
445     c_1.n -= 1;     // 这也是无奈之举啊，谁让 0.0 ==NULL 呢
446     c_2.n -= 1;
447 
448     Div_Dynamic_Array ans;
449     ans.A = (Div *)malloc(sizeof(Div));
450     if (ans.A == NULL) {
451         printf("Can't get memory!\n");
452         return 0;
453     }
454     ans.capacity = 1;
455     ans.n = 0;
456 
457     print(1, &c_1);
458     print(2, &c_2);
459     Div_onArray(&c_1, &c_2, &ans);
460     Div_print(&ans);
461     find(&ans);
462     system("pause");
463     return 0;
464 }
```

程序代码 1

运行结果

安装过程分析：

六、实验体会

Shell 的解释程序是最难的。

指针的操作比较复杂。

七、参考文献