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：2018-03-20
上机实践编号：No.01	组号：	上机实践时间：0:16

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 $c_i = \dfrac{a_i}{b_i}$，$i = 1,\ 2,\ \cdots,\ n$.

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

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：（这里隐藏了 PowerShell 的工作目录，仅用 PS > 作为提示符）

```
PS > .\div.exe "( -3.14,20 ,-256, 0 ,6,5,12121,4588, 89)" "(3.14, -1, 256,3.2222,2,0,5633.2,168,78)"
argument 1 is
(-3.14, 20.00, -256.00, 0.00, 6.00, 5.00, 12121.00, 4588.00, 89.00)

argument 2 is
(3.14, -1.00, 256.00, 3.22, 2.00, 0.00, 5633.20, 168.00, 78.00)

The answer C = (-1.00 , -20.00 , -1.00 , 0.00 , 3.00 , NaN , 2.15 , 27.31 , 1.14 )
Minimal Value is -1.00 , position is (1.00, 3.00)

PS >
```

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

double 数值跟着一个逗号。（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，直到切光。头部已经是处理好的了，所以一直切到遇到的第一个逗号，这个过程把逗号之前的字符，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，利用对称的坐标变换公式，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\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，就可以做除法、排序与查找了，有数字，有大小，C 很容易就做出来了。

程序代码

```c
1   // filename: div.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 C = (");
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 print(int n, Dynamic_Array *d) {    // 输出一个动态的双精度数组
141     printf(/* "argument %d is \n*/"(");
142     int i;
143     for (i = 0; i < d->n - 1; i++) {
144         printf("%2.2f, ", *(d->A + i));
145     }
146     printf("%2.2f", *(d->A + i));
147     printf(")\n\n");
148 }
149
150 void Resize(Dynamic_Array *D) {
151     int i = 0;
152     double *tmp = (double *)calloc(2 * D->capacity, sizeof(double));
153     if (tmp == NULL) {
154         printf("Cannot get memory, crash!\n");
155         return;
156     }
157     for (i = 0; i < D->capacity; i++) {
158         *(tmp + i) = *(D->A + i);
159     }
160     D->A = tmp;
161     D->capacity *= 2;
162 }
163
164 void Append(Dynamic_Array *D, double e) {
165     if (D->n == D->capacity) {
166         Resize(D);
167     }
168     *(D->A + D->n) = e;
169     D->n += 1;
170 }
171
172 Dynamic_Array *Quick_sort(Dynamic_Array *a) {
173
174     Dynamic_Array *less = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
175     less->A = (double *)calloc(1, sizeof(double));
176     if (!less) {
177         printf("Can't get memory!");
178         return NULL;
179     }
180     less->capacity = 1;
181     less->n = 0;
182
183     Dynamic_Array *more = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
184     more->A = (double *)calloc(1, sizeof(double));
```

```
185        if (!more) {
186            printf("Can't get memory!");
187            return NULL;
188        }
189        more->capacity = 1;
190        more->n = 0;
191
192        Dynamic_Array *eq = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
193        eq->A = (double *)calloc(1, sizeof(double));
194        if (!eq) {
195            printf("Can't get memory!");
196            return NULL;
197        }
198        eq->capacity = 1;
199        eq->n = 0;
200
201        int i;
202        if (a->n <= 1) {
203            return a;
204        }
205        else {
206            /*double pivot = 1 / 3. * (*(a->A) + ;*/
207
208            for (i = 0; i < a->n; i++) {
209                double pivot = *(a->A);
210                if (*(a->A + i) > pivot) {
211                    Append(more, *(a->A + i));
212                }
213                else {
214                    if (*(a->A + i) < pivot) {
215                        Append(less, *(a->A + i));
216                    }
217                    else {
218                        Append(eq, *(a->A + i));
219                    }
220                }
221            }
222        }
223        less = Quick_sort(less);
224        more = Quick_sort(more);
225        for (i = 0; i < eq->n; i++) {
226            Append(less, *(eq->A + i));
227        }
228        for (i = 0; i < more->n; i++) {
229            Append(less, *(more->A + i));
230        }
231        return less;
232    }
233
```

```
234  void find(Div_Dynamic_Array *a) {
235
236      Dynamic_Array *c = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
237      Dynamic_Array *d = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
238      c->A = (double *)calloc(a->n, sizeof(double));
239      if (c == NULL || d == NULL || c->A == NULL) {
240          printf("Can't get memory!\n");
241          return;
242      }
243      c->capacity = a->n;
244      c->n = 0;
245
246      int i = 0;
247      for (i = 0; i < a->n; i++) {
248          if (!strcmp((a->A + i)->state, "Normal")) {     // 分母合法的就 append
249              Append(c, (a->A + i)->value);
250          }
251      }
252      d = Quick_sort(c);      // 排序一下
253      //print(d->n, d);
254
255      double pivot = *(d->A + 0);
256      printf("Minimal Value is %2.2f , position is ", pivot);
257      Dynamic_Array *tmp = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
258      tmp->A = (double *)calloc(1, sizeof(double));
259      if (tmp == NULL || tmp->A == NULL) {
260          printf("Can't get memory!\n");
261          return;
262      }
263      tmp->capacity = 1;
264      tmp->n = 0;
265      for (i = 0; i < a->n; i++) {
266          if (!strcmp((a->A + i)->state, "Normal") && (a->A + i)->value == pivot) {
267              Append(tmp, ++i);
268          }
269      }
270      print(tmp->n, tmp);
271  }
272
273  char *clean(char *string) {     // 已经后期优化，减去了字符串中所有的空格
274      char *head = string;
275      int count_space = 0;
276      while (*string == ' ' && *string != '\0') {
277          count_space += 1;
278          string += 1;
279      }
280      string = head;
281
282      int len = 1;    // 有'\0'，所以要+1
```

```c
283        while (*string != '\0') {
284            len += 1;
285            string++;
286        }
287        string = head;
288
289        char *ans = (char *)calloc(len - count_space, sizeof(char));
290        if (ans == NULL) {
291            printf("Can't get memory!\n");
292            return NULL;
293        }
294        char *ans_head = ans;
295
296        while (*string != '\0') {
297            if (*string != ' ') {
298                *ans = *string;
299                ans++;
300            }
301            string++;
302        }
303        *ans = *string;
304        ans = ans_head;
305        string = head;
306
307        ans = ans + 1;
308        char *tmp;
309        for (tmp = ans; *tmp != '\0'; tmp++) {
310            if (*(tmp + 1) == '\0') {
311                *tmp = ',';
312            }
313        }
314        return ans;
315    }
316
317    char *cut(char *string) {
318        while (*string != ',') {
319            if (*string == '\0') {
320                return '\0';
321            }
322            string++;
323        }
324        return ++string;
325    }
326
327    // Put an new element into the stack
328    double get_Number(char *string) {
329        // 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。
330        if (*string == '\0') {
331            return NULL;
```

```c
332        }
333        double ans = 0.;
334        if (*string == '\0') {
335            return NULL;
336        }
337        if (*string != '-') {
338            char_LinkedList *work = (char_LinkedList *)malloc(sizeof(char_LinkedList));
339            if (work == NULL) {
340                printf("Can't get memory!\n");
341                return 0;
342            }
343            // container
344
345            char_LinkedList *head = work;
346            int i = 1;
347            while (*string != ',') {
348                work->elements = *string;
349                work->times = i;
350                work->next = (char_LinkedList *)malloc(sizeof(char_LinkedList));    // 申请
351                if (work->next == NULL) {
352                    printf("Can't get memory!\n");
353                    return 0.;
354                }
355                work = work->next;                                                  // 移动
356                work->elements = NULL;
357                work->times = NULL;
358                string++;
359                i++;    // i 在后面还有用
360            }
361
362            work->elements = *string;        // 逗号也要加上
363            work->times = NULL;              // 逗号的指数不能为有意义的
364
365            string++;
366
367            work = head;
368            int dot = 1;
369            int comma = 1;      // 逗号的用处
370            int dot_index = NULL;
371            while (work->elements != ',') {
372                if (work->elements == '.') {
373                    dot_index = dot;
374                    break;
375                }
376                work = work->next;
377                dot++;
378            }
379
380            if (dot_index == NULL) {
```
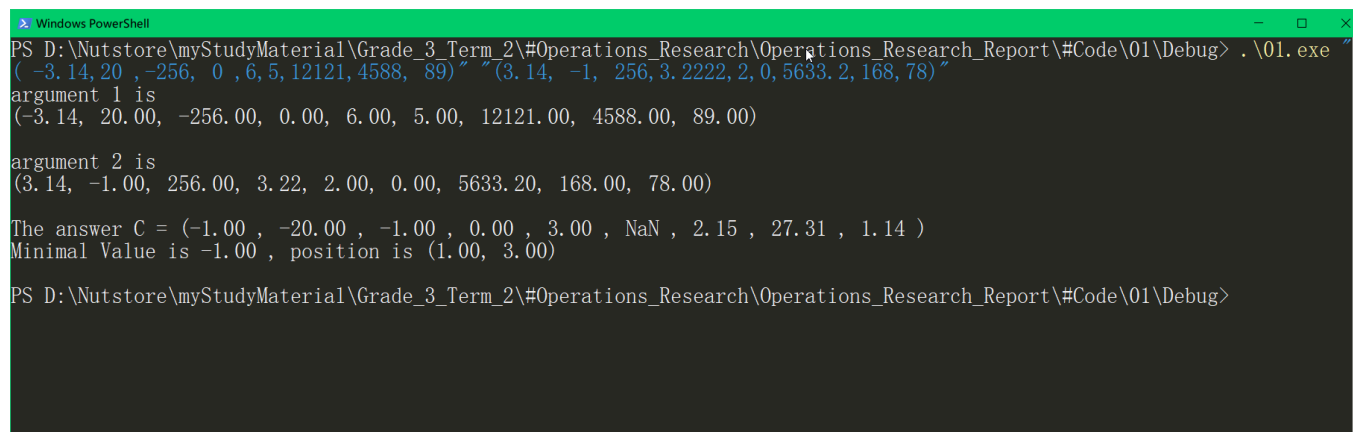
```
381            dot_index = i;
382        }
383
384        work = head;
385
386        while (work->times != NULL) {
387            work->times = -1 * (work->times - dot_index);
388            work = work->next;
389        }
390
391        work = head;
392
393        while (work->elements != ',') {
394            if (work->elements == '.') {
395                work = work->next;
396                continue;
397            }
398            if (work->times > 0) {
399                ans += pow(10, work->times - 1) * double(int(work->elements) - int('0'));
400                work = work->next;
401            }
402            else {
403                ans += pow(10, work->times) * double(int(work->elements) - int('0'));
404                work = work->next;
405            }
406        }
407    }
408    else {
409        string = string + 1;
410        ans = -1 * get_Number(string);
411    }
412    return ans;
413 }
414
415 int main(int argc, char *argv[]) {
416    if (argc != 3) {
417        printf("This function needs and only needs 2 arguments.\n");
418        return 0;
419    }
420
421    char *string_1 = *(argv + 1);
422    char *string_2 = *(argv + 2);
423
424    //char string_1_tmp[] = "( -3.14,20 ,-256, 0 ,6,5,12121,4588, 89)";
425    //char *string_1 = string_1_tmp;
426
427    //char string_2_tmp[] = "(3.14, -1, 256,3.2222,2,0,5633.2,168,78)";
428    //char *string_2 = string_2_tmp;
429
```

```
430        string_1 = clean(string_1);
431        string_2 = clean(string_2);
432
433        Dynamic_Array c_1, c_2;
434        c_1.A = (double *)malloc(sizeof(double));
435        if (c_1.A == NULL) {
436            printf("Can't get memory!\n");
437            return 0;
438        }
439        c_1.capacity = 1;
440        c_1.n = 0;
441
442        c_2.A = (double *)malloc(sizeof(double));
443        if (c_2.A == NULL) {
444            printf("Can't get memory!\n");
445            return 0;
446        }
447        c_2.capacity = 1;
448        c_2.n = 0;
449
450        while (string_1 != '\0') {
451            Append(&c_1, get_Number(string_1));
452            string_1 = cut(string_1);
453        }
454
455        while (string_2 != '\0') {
456            Append(&c_2, get_Number(string_2));
457            string_2 = cut(string_2);
458        }
459        c_1.n -= 1;      // 这也是无奈之举啊，谁让 0.0 ==NULL 呢
460        c_2.n -= 1;
461
462        Div_Dynamic_Array ans;
463        ans.A = (Div *)malloc(sizeof(Div));
464        if (ans.A == NULL) {
465            printf("Can't get memory!\n");
466            return 0;
467        }
468        ans.capacity = 1;
469        ans.n = 0;
470
471        printf("argument 1 is\n");
472        print(1, &c_1);
473        printf("argument 2 is\n");
474        print(2, &c_2);
475        Div_onArray(&c_1, &c_2, &ans);
476        Div_print(&ans);
477        find(&ans);
478        //system("pause");
```

```
479        return 0;
480    }
```

程序代码 1

运行结果

运行结果 1

代码分析

优势在于可以 shell 调用[1]，不再需要修改源代码；其次，数组是动态的，所以可以大容量输入。

劣势在于没有采用并行计算，在进行大规模计算的时候，只能调用一个 CPU 核心，效率较低。

六、实验体会

Shell 的解释程序是最难的，这里用了一个原创的方式，来解释输入的字符串。

指针的操作比较复杂，需要时刻牢记 malloc 与 free 的对应[2]，并且要对堆中申请到的地址进行排查，看是否申请成功。在进行调试的时候，时常遇到内存的读取冲突问题，查找了微软的官方 Visual C++编译器的手册，方才明白这里的局部变量必须要初始化才可以使用，这与 GNU 的 MinGW 编译器稍有区别。

七、参考文献

1. Hahn, H., *Harley Hahn's Guide to Unix and Linux*. 2009, New York: McGraw-Hill.
2. **林锐**, *高质量 C++/C 编程指南*. 1.0 ed. 2001.