云南大学数学与统计学院  
《运筹学通论实验》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 | 专业：信息与计算科学 |
| 上机实践名称：求给定序列的最小值及所有最小值的下标 | 学号：20151910042 | 上机实践日期：2018-07-07 |
| 上机实践编号：1 | 组号： |  |

# 实验目的

完成该实验，为后期的更进一步的实验做准备。

# 实验内容

给定两组数和，求

1. 一组数，其中

2. 求最小值及所有最小值的下标，其中最小值为

# 实验平台

Windows 10 Pro 1703；

Microsoft Visual Studio 2017 Enterprise。

# 算法设计[[1]](#footnote-1)

**Algorithm**: **DIV**, find the minimal value and all its(their) indexes

**Input**: and .

**Output**: list , minimal value and their indexes.

**Begin**

**Step 1**: DEFINE

**Step 2**:

**Step 3**: is a set who contains all the elements in whose

**Step 4**: sort incrementally, let the first element of the sorted

**Step 5**: find all elements in whose and equal to then put their indexes into .

**End**

# 程序代码

## 程序描述

这个解释程序的使用方法是这样的：在shell中通过调用本可执行程序div，输入两个字符串参数，然后程序自动输出与最小值及其所有位置。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：负号直接跳过，最后乘；单个的数字与小数点直接归入队列，与此同时，队列的头号元素，跟随着一个从开始的索引，该索引按照增序排到队列的末尾——遇到的第一个分号。如此之后，可以通过遍历一次，找到小数点所在位置对应的索引，然后利用对称的坐标变换公式，把其他数字符号与小数点的距离转化为的指数，然后通过pow函数算出具体的数值，完成字符到数值的转化。

在整个过程中要注意保护头指针与work指针的归位。一个数字一旦算出，就交给动态数组保存。整个字符串的切割，一直做到\0。这个过程一直中，一直保持着保存操作。当解释程序返回一个浮点数就要存入，返回NULL就结束归入。当遇到\0之后，也就得到了一个存有输入信息的双精度数组。

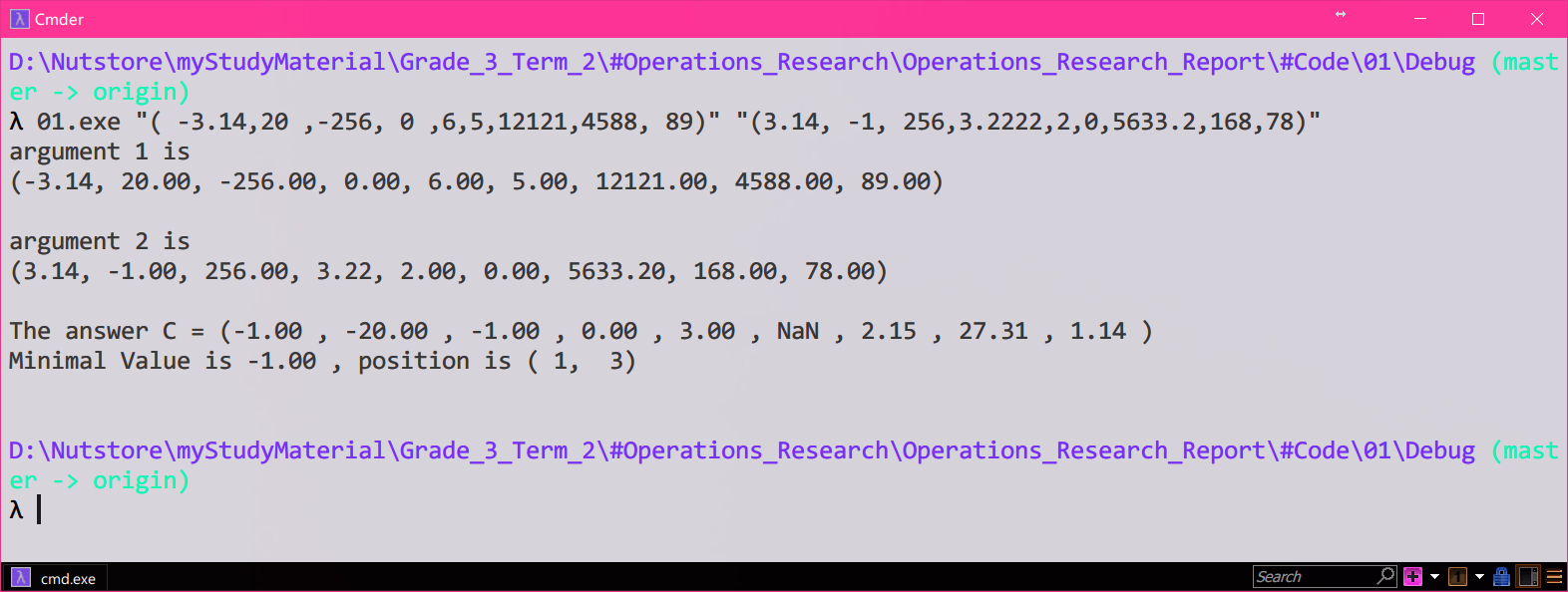
拿到了两个动态数组之后，就可以做除法、排序与查找了。

## 程序代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456  457  458  459  460  461  462  463  464  465  466  467  468  469  470  471  472  473  474  475  476  477  478  479  480  481  482  483  484  485  486  487  488  489  490  491  492  493  494  495  496  497  498  499  500  501  502  503  504  505  506  507  508 | /\*  \* Copyright (c) 2018, Liu Peng, School of Mathematics and Statistics, YNU  \* Apache License.  \*  \* 文件名称：Source.cpp  \* 文件标识：见配置管理计划书  \* 摘 要：Prim算法  \*  \* 当前版本：1.0  \* 作 者：刘鹏  \* 创建日期：2018年3月14日  \* 完成日期：2018年6月25日  \*  \* 取代版本：  \* 原作者 ：刘鹏  \* 完成日期：  \*/  /\*  \* A function like division which can execuate with some conditions.  \*/  #include<stdio.h>  #include<stdlib.h>  #include<string.h>  #include<math.h>  // The following type is a container for creating a stack.  **typedef** struct char\_LinkedList **{**  char\_LinkedList **\***head**;**  char elements**;** // partition must be integer less than 10  int times**;** // container  char\_LinkedList **\***next**;**  **}**char\_LinkedList**;**  **typedef** struct Dynamic\_Array **{**  double **\***A**;** // low-level array  int capacity**;** // the capacity  int n**;** // used room  **}**Dynamic\_Array**;**  **typedef** struct Div **{**  double up**;**  double down**;**  double value**;**  char state**[**10**];** // NaN or Negative or Normal  **}**Div**;**  **typedef** struct Div\_Dynamic\_Array **{**  Div **\***A**;** // 底层结构体数组的头指针，不能动！  int capacity**;** // 底层结构体数组的容量  int n**;** // 底层数组的占用量  **}**Div\_Dynamic\_Array**;**  void Div\_Resize**(**Div\_Dynamic\_Array **\***D**)** **{**  int i **=** 0**;**  Div **\***tmp **=** **(**Div **\*)**calloc**(**2 **\*** D**->**capacity**,** **sizeof(**Div**));**  **if** **(**tmp **==** **NULL)** **{**  printf**(**"Cannot get memory, crash!\n"**);**  **return;**  **}**  **for** **(**i **=** 0**;** i **<** D**->**capacity**;** i**++)** **{**  **(**tmp **+** i**)->**up **=** **(**D**->**A **+** i**)->**up**;**  **(**tmp **+** i**)->**down **=** **(**D**->**A **+** i**)->**down**;**  **(**tmp **+** i**)->**value **=** **(**D**->**A **+** i**)->**value**;**  strcpy**((**tmp **+** i**)->**state**,** **(**D**->**A **+** i**)->**state**);**  **}**  free**(**D**->**A**);**  D**->**A **=** tmp**;**  tmp **=** **NULL;**  D**->**capacity **\*=** 2**;**  **}**  void Div\_Append**(**Div\_Dynamic\_Array **\***D**,** Div e**)** **{**  **if** **(**D**->**n **==** D**->**capacity**)** **{**  Div\_Resize**(**D**);**  **}**  **(**D**->**A **+** D**->**n**)->**up **=** e**.**up**;**  **(**D**->**A **+** D**->**n**)->**down **=** e**.**down**;**  **(**D**->**A **+** D**->**n**)->**value **=** e**.**value**;**  strcpy**((**D**->**A **+** D**->**n**)->**state**,** e**.**state**);**  D**->**n **+=** 1**;**  //int i;  //for (i = 0; i <= D->n; i++) {  // printf("%s\t", (D->A + i)->state);  //}  //printf("\n");  **}**  void Div\_print**(**Div\_Dynamic\_Array **\***d**)** **{**  int i**;**  printf**(**"The answer C = ("**);**  **for** **(**i **=** 0**;** i **<** d**->**n**;** i**++)** **{**  **if** **(!**strcmp**((**d**->**A **+** i**)->**state**,** "NaN"**))** **{**  printf**(**"%s "**,** "NaN"**);**  **}**  **else** **{**  double value **=** **(**d**->**A **+** i**)->**value**;**  printf**(**"%2.2f "**,** value**);**  **}**  **if** **(**i **==** d**->**n **-** 1**)** **{**  printf**(**""**);**  **}**  **else** **{**  printf**(**", "**);**  **}**  **}**  printf**(**")\n"**);**  **}**  void Div\_onArray**(**Dynamic\_Array **\***a**,** Dynamic\_Array **\***b**,** Div\_Dynamic\_Array **\***ans**)** **{**  **if** **(**a**->**n **!=** b**->**n**)** **{**  printf**(**"length should be the same."**);**  **return;**  **}**  int i**;**  **for** **(**i **=** 0**;** i **<** a**->**n**;** i**++)** **{**  **if** **(\*(**b**->**A **+** i**)** **==** 0**)** **{**  Div tmp**;**  tmp**.**up **=** **NULL;**  tmp**.**down **=** **NULL;**  tmp**.**value **=** **NULL;**  char c**[]** **=** "NaN"**;**  strcpy**(**tmp**.**state**,** c**);**  Div\_Append**(**ans**,** tmp**);**  **}**  **else** **{**  **if** **(\*(**b**->**A **+** i**)** **<** 0.**)** **{**  Div tmp**;**  tmp**.**up **=** **\*(**a**->**A **+** i**);**  tmp**.**down **=** **\*(**b**->**A **+** i**);**  tmp**.**value **=** tmp**.**up **/** tmp**.**down**;**  char c**[]** **=** "Negative"**;**  strcpy**(**tmp**.**state**,** c**);**  Div\_Append**(**ans**,** tmp**);**  **}**  **else** **{**  Div tmp**;**  tmp**.**up **=** **\*(**a**->**A **+** i**);**  tmp**.**down **=** **\*(**b**->**A **+** i**);**  tmp**.**value **=** tmp**.**up **/** tmp**.**down**;**  char c**[]** **=** "Normal"**;**  strcpy**(**tmp**.**state**,** c**);**  Div\_Append**(**ans**,** tmp**);**  **}**  **}**  **}**  **}**  // output a Double array  void print**(**int n**,** Dynamic\_Array **\***d**)** **{**  printf**(**/\* "argument %d is \n\*/"("**);**  int i**;**  **for** **(**i **=** 0**;** i **<** d**->**n **-** 1**;** i**++)** **{**  printf**(**"%2.2f, "**,** **\*(**d**->**A **+** i**));**  **}**  printf**(**"%2.2f"**,** **\*(**d**->**A **+** i**));**  printf**(**")\n\n"**);**  **}**  // Output a double-array with integer format  void print\_int**(**int n**,** Dynamic\_Array **\***d**)** **{**  printf**(**/\* "argument %d is \n\*/"("**);**  int i**;**  **for** **(**i **=** 0**;** i **<** d**->**n **-** 1**;** i**++)** **{**  printf**(**"%2.0f, "**,** **\*(**d**->**A **+** i**));**  **}**  printf**(**"%2.0f"**,** **\*(**d**->**A **+** i**));**  printf**(**")\n\n"**);**  **}**  void Resize**(**Dynamic\_Array **\***D**)** **{**  int i **=** 0**;**  double **\***tmp **=** **(**double **\*)**calloc**(**2 **\*** D**->**capacity**,** **sizeof(**double**));**  **if** **(**tmp **==** **NULL)** **{**  printf**(**"Cannot get memory, crash!\n"**);**  **return;**  **}**  **for** **(**i **=** 0**;** i **<** D**->**capacity**;** i**++)** **{**  **\*(**tmp **+** i**)** **=** **\*(**D**->**A **+** i**);**  **}**  D**->**A **=** tmp**;**  D**->**capacity **\*=** 2**;**  **}**  void Append**(**Dynamic\_Array **\***D**,** double e**)** **{**  **if** **(**D**->**n **==** D**->**capacity**)** **{**  Resize**(**D**);**  **}**  **\*(**D**->**A **+** D**->**n**)** **=** e**;**  D**->**n **+=** 1**;**  **}**  Dynamic\_Array **\***Quick\_sort**(**Dynamic\_Array **\***a**)** **{**  Dynamic\_Array **\***less **=** **(**Dynamic\_Array **\*)**calloc**(**1**,** **sizeof(**Dynamic\_Array**));**  less**->**A **=** **(**double **\*)**calloc**(**1**,** **sizeof(**double**));**  **if** **(!**less**)** **{**  printf**(**"Can't get memory!"**);**  **return** **NULL;**  **}**  less**->**capacity **=** 1**;**  less**->**n **=** 0**;**  Dynamic\_Array **\***more **=** **(**Dynamic\_Array **\*)**calloc**(**1**,** **sizeof(**Dynamic\_Array**));**  more**->**A **=** **(**double **\*)**calloc**(**1**,** **sizeof(**double**));**  **if** **(!**more**)** **{**  printf**(**"Can't get memory!"**);**  **return** **NULL;**  **}**  more**->**capacity **=** 1**;**  more**->**n **=** 0**;**  Dynamic\_Array **\***eq **=** **(**Dynamic\_Array **\*)**calloc**(**1**,** **sizeof(**Dynamic\_Array**));**  eq**->**A **=** **(**double **\*)**calloc**(**1**,** **sizeof(**double**));**  **if** **(!**eq**)** **{**  printf**(**"Can't get memory!"**);**  **return** **NULL;**  **}**  eq**->**capacity **=** 1**;**  eq**->**n **=** 0**;**  int i**;**  **if** **(**a**->**n **<=** 1**)** **{**  **return** a**;**  **}**  **else** **{**  **for** **(**i **=** 0**;** i **<** a**->**n**;** i**++)** **{**  double pivot **=** **\*(**a**->**A**);**  **if** **(\*(**a**->**A **+** i**)** **>** pivot**)** **{**  Append**(**more**,** **\*(**a**->**A **+** i**));**  **}**  **else** **{**  **if** **(\*(**a**->**A **+** i**)** **<** pivot**)** **{**  Append**(**less**,** **\*(**a**->**A **+** i**));**  **}**  **else** **{**  Append**(**eq**,** **\*(**a**->**A **+** i**));**  **}**  **}**  **}**  **}**  less **=** Quick\_sort**(**less**);**  more **=** Quick\_sort**(**more**);**  **for** **(**i **=** 0**;** i **<** eq**->**n**;** i**++)** **{**  Append**(**less**,** **\*(**eq**->**A **+** i**));**  **}**  **for** **(**i **=** 0**;** i **<** more**->**n**;** i**++)** **{**  Append**(**less**,** **\*(**more**->**A **+** i**));**  **}**  **return** less**;**  **}**  void find**(**Div\_Dynamic\_Array **\***a**)** **{**  Dynamic\_Array **\***c **=** **(**Dynamic\_Array **\*)**calloc**(**1**,** **sizeof(**Dynamic\_Array**));**  Dynamic\_Array **\***d **=** **(**Dynamic\_Array **\*)**calloc**(**1**,** **sizeof(**Dynamic\_Array**));**  c**->**A **=** **(**double **\*)**calloc**(**a**->**n**,** **sizeof(**double**));**  **if** **(**c **==** **NULL** **||** d **==** **NULL** **||** c**->**A **==** **NULL)** **{**  printf**(**"Can't get memory!\n"**);**  **return;**  **}**  c**->**capacity **=** a**->**n**;**  c**->**n **=** 0**;**  int i **=** 0**;**  **for** **(**i **=** 0**;** i **<** a**->**n**;** i**++)** **{**  // denominator is legal  **if** **(!**strcmp**((**a**->**A **+** i**)->**state**,** "Normal"**))** **{**  Append**(**c**,** **(**a**->**A **+** i**)->**value**);**  **}**  **}**  d **=** Quick\_sort**(**c**);**  double pivot **=** **\*(**d**->**A **+** 0**);**  Dynamic\_Array **\***tmp **=** **(**Dynamic\_Array **\*)**calloc**(**1**,** **sizeof(**Dynamic\_Array**));**  tmp**->**A **=** **(**double **\*)**calloc**(**1**,** **sizeof(**double**));**  **if** **(**tmp **==** **NULL** **||** tmp**->**A **==** **NULL)** **{**  printf**(**"Can't get memory!\n"**);**  **return;**  **}**  tmp**->**capacity **=** 1**;**  tmp**->**n **=** 0**;**  **for** **(**i **=** 0**;** i **<** a**->**n**;** i**++)** **{**  **if** **(!**strcmp**((**a**->**A **+** i**)->**state**,** "Normal"**)** **&&** **(**a**->**A **+** i**)->**value **==** pivot**)** **{**  Append**(**tmp**,** **++**i**);**  **}**  **}**  **if** **(**tmp**->**n **==** 0**)** **{**  printf**(**"Sorry, no minimal value.\n"**);**  **return;**  **}**  printf**(**"Minimal Value is %2.2f , position is "**,** pivot**);**  print\_int**(**tmp**->**n**,** tmp**);**  **}**  // Get rid of the useless blank characters.  char **\***clean**(**char **\***string**)** **{**  char **\***head **=** string**;**  int count\_space **=** 0**;**  **while** **(\***string **==** ' ' **&&** **\***string **!=** '\0'**)** **{**  count\_space **+=** 1**;**  string **+=** 1**;**  **}**  string **=** head**;**  int len **=** 1**;**  **while** **(\***string **!=** '\0'**)** **{**  len **+=** 1**;**  string**++;**  **}**  string **=** head**;**  char **\***ans **=** **(**char **\*)**calloc**(**len **-** count\_space**,** **sizeof(**char**));**  **if** **(**ans **==** **NULL)** **{**  printf**(**"Can't get memory!\n"**);**  **return** **NULL;**  **}**  char **\***ans\_head **=** ans**;**  **while** **(\***string **!=** '\0'**)** **{**  **if** **(\***string **!=** ' '**)** **{**  **\***ans **=** **\***string**;**  ans**++;**  **}**  string**++;**  **}**  **\***ans **=** **\***string**;**  ans **=** ans\_head**;**  string **=** head**;**  ans **=** ans **+** 1**;**  char **\***tmp**;**  **for** **(**tmp **=** ans**;** **\***tmp **!=** '\0'**;** tmp**++)** **{**  **if** **(\*(**tmp **+** 1**)** **==** '\0'**)** **{**  **\***tmp **=** ','**;**  **}**  **}**  **return** ans**;**  **}**  char **\***cut**(**char **\***string**)** **{**  **while** **(\***string **!=** ','**)** **{**  **if** **(\***string **==** '\0'**)** **{**  **return** '\0'**;**  **}**  string**++;**  **}**  **return** **++**string**;**  **}**  // Put an new element into the stack  double get\_Number**(**char **\***string**)** **{**  **if** **(\***string **==** '\0'**)** **{**  **return** **NULL;**  **}**  double ans **=** 0.**;**  **if** **(\***string **==** '\0'**)** **{**  **return** **NULL;**  **}**  **if** **(\***string **!=** '-'**)** **{**  char\_LinkedList **\***work **=** **(**char\_LinkedList **\*)**malloc**(sizeof(**char\_LinkedList**));**  **if** **(**work **==** **NULL)** **{**  printf**(**"Can't get memory!\n"**);**  **return** 0**;**  **}**  // container  char\_LinkedList **\***head **=** work**;**  int i **=** 1**;**  **while** **(\***string **!=** ','**)** **{**  work**->**elements **=** **\***string**;**  work**->**times **=** i**;**  work**->**next **=** **(**char\_LinkedList **\*)**malloc**(sizeof(**char\_LinkedList**));** // malloc  **if** **(**work**->**next **==** **NULL)** **{**  printf**(**"Can't get memory!\n"**);**  **return** 0.**;**  **}**  work **=** work**->**next**;** // move  work**->**elements **=** **NULL;**  work**->**times **=** **NULL;**  string**++;**  i**++;**  **}**  work**->**elements **=** **\***string**;**  work**->**times **=** **NULL;**  string**++;**  work **=** head**;**  int dot **=** 1**;**  int comma **=** 1**;**  int dot\_index **=** **NULL;**  **while** **(**work**->**elements **!=** ','**)** **{**  **if** **(**work**->**elements **==** '.'**)** **{**  dot\_index **=** dot**;**  **break;**  **}**  work **=** work**->**next**;**  dot**++;**  **}**  **if** **(**dot\_index **==** **NULL)** **{**  dot\_index **=** i**;**  **}**  work **=** head**;**  **while** **(**work**->**times **!=** **NULL)** **{**  work**->**times **=** **-**1 **\*** **(**work**->**times **-** dot\_index**);**  work **=** work**->**next**;**  **}**  work **=** head**;**  **while** **(**work**->**elements **!=** ','**)** **{**  **if** **(**work**->**elements **==** '.'**)** **{**  work **=** work**->**next**;**  **continue;**  **}**  **if** **(**work**->**times **>** 0**)** **{**  ans **+=** pow**(**10**,** work**->**times **-** 1**)** **\*** double**(**int**(**work**->**elements**)** **-** int**(**'0'**));**  work **=** work**->**next**;**  **}**  **else** **{**  ans **+=** pow**(**10**,** work**->**times**)** **\*** double**(**int**(**work**->**elements**)** **-** int**(**'0'**));**  work **=** work**->**next**;**  **}**  **}**  **}**  **else** **{**  string **=** string **+** 1**;**  ans **=** **-**1 **\*** get\_Number**(**string**);**  **}**  **return** ans**;**  **}**  int main**(**int argc**,** char **\***argv**[])** **{**  **if** **(**argc **!=** 3**)** **{**  printf**(**"This function needs and only needs 2 arguments.\n"**);**  **return** 0**;**  **}**  char **\***string\_1 **=** **\*(**argv **+** 1**);**  char **\***string\_2 **=** **\*(**argv **+** 2**);**  //char string\_1\_tmp[] = "( -3.14,20 ,-256, 0 ,6,5,12121,4588, 89)";  //char \*string\_1 = string\_1\_tmp;  //char string\_2\_tmp[] = "(3.14, -1, 256,3.2222,2,0,5633.2,168,78)";  //char \*string\_2 = string\_2\_tmp;  string\_1 **=** clean**(**string\_1**);**  string\_2 **=** clean**(**string\_2**);**  Dynamic\_Array c\_1**,** c\_2**;**  c\_1**.**A **=** **(**double **\*)**malloc**(sizeof(**double**));**  **if** **(**c\_1**.**A **==** **NULL)** **{**  printf**(**"Can't get memory!\n"**);**  **return** 0**;**  **}**  c\_1**.**capacity **=** 1**;**  c\_1**.**n **=** 0**;**  c\_2**.**A **=** **(**double **\*)**malloc**(sizeof(**double**));**  **if** **(**c\_2**.**A **==** **NULL)** **{**  printf**(**"Can't get memory!\n"**);**  **return** 0**;**  **}**  c\_2**.**capacity **=** 1**;**  c\_2**.**n **=** 0**;**  **while** **(**string\_1 **!=** '\0'**)** **{**  Append**(&**c\_1**,** get\_Number**(**string\_1**));**  string\_1 **=** cut**(**string\_1**);**  **}**  **while** **(**string\_2 **!=** '\0'**)** **{**  Append**(&**c\_2**,** get\_Number**(**string\_2**));**  string\_2 **=** cut**(**string\_2**);**  **}**  c\_1**.**n **-=** 1**;**  c\_2**.**n **-=** 1**;**  Div\_Dynamic\_Array ans**;**  ans**.**A **=** **(**Div **\*)**malloc**(sizeof(**Div**));**  **if** **(**ans**.**A **==** **NULL)** **{**  printf**(**"Can't get memory!\n"**);**  **return** 0**;**  **}**  ans**.**capacity **=** 1**;**  ans**.**n **=** 0**;**  printf**(**"argument 1 is\n"**);**  print**(**1**,** **&**c\_1**);**  printf**(**"argument 2 is\n"**);**  print**(**2**,** **&**c\_2**);**  Div\_onArray**(&**c\_1**,** **&**c\_2**,** **&**ans**);**  Div\_print**(&**ans**);**  find**(&**ans**);**  //system("pause");  **return** 0**;**  **}** |

程序代码 1

# 运行结果



运行结果 1 （经过了反相处理）

## 代码分析

优势在于可以shell调用，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。

# 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[1]，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。

# 参考文献

[1] **林锐**. 高质量 C++/C 编程指南 [M]. 1.0 ed., 2001.

1. 此处的伪代码中，矩阵运算符的意义均与MATLAB语言一致，如矩阵的左除、右除和点除等。 [↑](#footnote-ref-1)