**大作业说明文档**

计95班 熊天翼 2019011303

计95班 徐栋 2019011305

计96班 李伟楷 2019011327

1. **基本说明**
   1. 关于画图：位图的长度和高度要求必须为能够被4整除的正整数，最大图像支持5000\*5000（支持4k图像）。在本解释器中，x坐标轴沿屏幕向右， y坐标轴沿屏幕向下，原点位于屏幕左上角。
   2. 关于logo文件的输入：本解释器要求使用者手动键入logo文件的地址相对路径，logo文件放在input\_file文件夹中。输入样例： ..\\input\_file\\test.logo
   3. 关于bmp文件的输出：本解释器要求使用者手动键入bmp文件的相对地址。输入样例： ..\\output\_file\\test.logo
   4. 关于logo文件的格式：本解释器对于logo文件的空行，空格缩进均无要求。但要求每一个语句能且仅能占据logo文件中的一行。对于文件输入格式的具体要求，请参见基本功能部分及debug 部分。而最重要的是，本解释器要求logo文件中所有字符均为英文字符，对于中文的括号，逗号，冒号均不支持。
2. **小组分工**
   1. 熊天翼（组长）：负责数据结构的设计以及整体代码框架的实现， 包括logo文件的输入，命令字符的读取，基本语句的完成（COLOR，CLOAK，MOVE，TURN， DELETE，FUNC, CALL），设计相关字符读取函数（gett\_char，getpara），bmp格式图像的输出以及数据结构的设置；同时实现了涂色这一拓展功能。另外，熊天翼负责整体小组进度的安排与监督，并协助调试了递归拓展功能。
   2. 徐栋：完成了划线函数这一重要部分的实现，共改良了6版划线函数；针对划线函数，添加了logo语言中的SIZE语句以更改画笔粗细；同时，他主要负责完成伪logo代码的调试与书写。
   3. 李伟楷：负责debug拓展功能以及函数递归功能的实现，并针对拓展功能实现了pro，if拓展语句，且通过建造gett\_float函数改良程序使其支持小数输入。另外，李伟楷同学展示了设计了tree图像等一系列复杂递归图形。

在小组中，李伟楷和徐栋两位同学主要负责视频的制作，熊天翼在其中起到辅助作用。本说明文档由三人共同书写（每个人主要负责自己的部分）。

1. **基本功能完成情况**
   1. 结构体以及变量命名说明
      1. 结构体
         1. cursor：画笔，包括画笔的横纵坐标、朝向、粗细、是否隐身以及颜色
         2. BITMAPFILEHEADER， BITMAPINFOHEADER：bmp文件头和信息头，用来bmp记录基础信息
         3. color：用来记录颜色，其中包含rgb分量
         4. paralist：用来记录参数。
         5. recfunct：用来记录函数。包括函数名（name），函数参数数量（fpanum），函数的起始行（fbeg）和终止行（fen）。
         6. variab：用于记录变量，包含变量名（name）和当前的值（value）
      2. 一些变量的命名说明

针对于本作业中可能不太容易理解且处于程序核心位置的变量，我们在下表中给予具体说明。，

|  |  |  |  |
| --- | --- | --- | --- |
| 变量名 | 类型 | 作用 | 主要相关函数 |
| Dra | color[][] | 记录图像每个色块的颜色,画图用 | backcolor, move |
| Rec | color[] | 记录颜色的一维数组，输出用（用conver函数实现二维数组到一维数组的映射） | output，conver |
| commands | string[] | 记录@以下的所有信息 | 全部 |
| tmp\_get | char[] | 用以读取命令中的字符 | get\_para，readcom，gett\_char |
| funcpname | int | 记录所有函数的形式变量名称 | readcom |
| I | int | 记录命令读取到第几行 | readcom，get\_系列 |
| G | int | 记录命令读取到commands的第【i】行的第几个位置 | readcom，get\_系列 |
| Rvalistnum | int | 全局变量个数 |  |
| Q | int | 与valistnum一起使用，见下 |  |
| valistnum‘’ | int |  |  |

关于函数调用与递归中使用的一些变量的说明：

rvalistnum：用于记录全局变量的数量。全局变量存在variab数组的第0-第（rvalistnum-1）个位置。

valistnum：用于记录局部变量的数量，即函数中定义的变量的数量，在函数调用和递归中发挥作用。这部分变量存在variab数组的第ravalistnum-第（ravalistnum+valistnum-1）个位置，当函数层数funcl大于0时定义的变量被存在这部分，被定义的变量包含在函数名后面的括号内出现的变量和函数内部定义的变量。

q：用于记录在非仅局限于本作用域，且当前可使用的变量的数量，在函数调用和递归中发挥作用，即为当前层函数之前的函数中所定义的变量及全局变量数量。存在variab数组的第q-第（ravalistnum+valistnum-1）个位置。

* 1. 存储、执行命令的方法

本解释器对于输入logo文件的读取采用先读取基本参数（图像大小、背景色、笔的初始位置），之后将所有命令忽略空行以string类型直接存储（commands【600】），之后按行读取。我们的最大读取能力是600行（不包括含“@”的基本参数）

在本作业中，我们主要使用的是readcom，gett\_float, get\_char，getpara函数来完成每行命令的读取。我们首先通过readcom函数读取每行的第一个命令（大写字母单词）并判断命令所属类型，之后按情况进行具体讨论。

未知类型的数据将使用gett\_char函数进行读取，而确定数据类型的数据将使用get\_para（读取变量名）或者gett\_float（读取具体数值）来进行读取。

gett\_char函数（返回值int）负责读取所有未知类型的数据（字符串，变量名，标点符号以及具体数值），按类型给予不同的返回值，指引下一步的操作。对于如果该数据是字符串，则gett\_char直接读取并存入全局变量tmp\_get，方便后续使用。如果是其他类型的数据则需要其他函数完成具体读取。本解释器中变量名仅支持小写字母字符串，所有数值可支持整数或浮点数（正负均可）。

gett\_float函数（返回值float）负责读取logo语言中的具体数值，包含整数和小数，正负均可，可以被get\_para调用，或者被从readcom（）函数直接调用（在确认该位置数据是具体数值情况下，如DEF x后面）。它接收当前需要读取的数所在的行数和在该行的位置（该位置在需要读取的数之前），跳过该位置和数之间的空格、‘）’和逗号，读取该数并返回该数。

get\_para函数（返回值float）负责读取参数，包含变量和具体数值。它接收需要读取的变量或数的位置，通过调用gett\_char函数判断该数据的类型。如果是具体数值则调用gett\_float函数进行输入；如果是变量则搜索目前已存储的变量list，找到该变量并返回其当前值。这一步中会先在全局变量范围内搜索，之后再在当前函数（如果在函数中）中的局部变量范围内搜索。如果该变量名字同时在函数局部变量范围内和全局变量范围内重复出现，则在局部变量范围内获得的变量值会覆盖之前搜索到的全局变量函数值。如果该处不存在变量或数，则会通过返回值来报错。

* 1. 关于具体语句的实现说明
     1. MOVE：继续向下使用getpara函数调取步长，之后根据具体情境不同利用不同划线函数进行划线（若未隐身）
     2. TURN：继续用getpara读取旋转角度，并旋转笔的方向
     3. COLOR：继续使用getpara读取，若未读取到数据，则笔保持原来的颜色，否则改变笔的颜色
     4. DEF：若定义全局变量，在名为variab的存储变量的数组中写入定义的变量名与具体数值，之后全局变量数目（rvalistnum）加一。若在函数中定义变量，则写入操作相同，之后本层函数中的参数数目（valistnum-q）加一。注意，定义变量的数值不能使用变量的数值！注意，函数中定义形参的名称时其名称不得与全局变量相冲突。以及，全局变量必须在任何一个函数被调用之前定义。另外，不允许在循环结构中定义变量，同时不允许对于变量进行重复定义。（要求：整个程序中全局变量数+最多递归层数\*（函数中定义的变量数+函数形参数量 <= 100000）
     5. ADD：getpara函数会在已有的列表中找出变量名相同的变量并对其进行加减操作。

以上这些操作，完成之后程序会读取下一行的命令

* + 1. LOOP：实现到loop后，我们首先用getpara函数读取循环的次数n。此后，我们会按行找到本循环（不是内嵌循环）对应的END LOOP，并记录循环的初始和终止位置。之后，程序会从循环的头部（LOOP下一行）读到尾部（END LOOP前一行），反复n次。之后，程序会跳到END LOOP的下一行。注意，LOOP中不允许定义变量或函数。亲测循环500万次需要一分钟时间，建议总LOOP次数不超过10万次。
    2. FUNC：在读取到这一操作后，先使用gett\_char函数读取函数与参数的名称，要求参数被括在英文括号内，中间以应为逗号隔开。之后，程序会向下读取直到END FUNC语句，从而记录下函数的开始和终止位置。这些信息都会被记录recfunct结构的名为funct的数组之中。结束之后，程序跳到END FUNC的下一行读取。
    3. CALL ：先使用gett\_char读取函数名，找到需调用的函数，之后用getpara读取函数参数的具体数值，并将这些参数复制到variab中，使函数可以调用自己定义的变量与全局变量。此后，程序会从函数的头部（FUNC的下一行）开始逐行读取。结束后，程序读取CALL 的下一行。

拓展功能语句

* + 1. 画线函数：该函数的主要目的是实现伪LOGO语言的不同粗细的线的划线功能。划线功能是整个LOGO语言的核心功能，故该函数是将输入文档转化为输出位图文件的关键步骤，是实现用LOGO语言的指令画图的基础。SIZE用来改变笔的粗细，在后面可以输入数值或者变量名，代表笔的相对粗细。初始默认笔的粗细为0。笔的实际粗细的换算公式约为2\*size+1
    2. FILL：这一拓展功能为涂色。涂色函数后面需要三个变量或数值，分别代表所涂颜色的rgb分量。涂色函数的停止情况有两种：1、涂到画布边界；2、所涂的格颜色与选取涂色区域之前的颜色不同（这是边界的标志）。涂色完成后，程序读取下一行。
    3. DELETE：这一拓展功能用来删除全局变量，在它之后需输入变量名。程序会找到变量名对应的变量将其从列表中剔除，并将之后的变量前移一位。注意，不允许删除列表中没有的变量，且本功能不支持删除函数中定义的变量。
    4. IF：这个拓展功能进行分支结构的实现，具体包含IF STOP语句和IF THEN语句，分别用于在满足IF后的条件时退出一层循环/函数，读取该循环/函数之后的语句；或执行THEN与配套的END IF之间的语句。IF STOP语句仅支持在循环或函数内使用。IF THEN语句不支持多层嵌套。
    5. PRO：利用get\_para语句读取变量或具体数值并进行乘法运算。此功能可服务于递归、循环等。
    6. Debug功能：针对一些常见的logo语句的bug设计了debug功能，如果logo文件有错误可以输出其错误所在行数和错误原因。

1. **拓展功能的具体实现方式**
   1. 涂色方式：

在收到涂色指令后，程序会先提取画笔（pen）所在位置的颜色作为参考色来选取边界。之后，我们建立两个int型动态数组lef，rig来记录之后左右分别要开始涂色的位置，并用乐福num，rignum记录数组中元素个数。同时用iflef，ifrig我们从pen所在的位置开始向下走，将走过的每个位置rec【y-i】【x】中的颜色均改为FILL指令后规定的颜色（参见3.x）。我们以左侧为例来说明涂色规则。最初iflef为false，表示左侧不需要涂色。当行走到某一格的时候，若iflef为false而且这一格左面的格子没有涂色，那么将iflef改为true，并在记录下这个位置的纵坐标【y-i】，lefnum加一。如果iflef为true（表示前一个格需要涂左边的一格）而这一格的左侧已经到达边界，那么则将iflef改为false。当行走到最低端后，返回至【y+1】【x】，将iflef改为false，之后向上走，按照此规则走到顶端。此时这一列就完成了涂色。之后进行递归，涂lefnum所记录的格。右边同理。这样就能尽可能不重复的遍历区域内所有需要涂色的位置进行涂色。以下是一个图形解释

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x |  |  |  | 4 | 0 |  |  |  |  |  |  |  | x |  |
|  | x | x | x | x | 0 |  |  |  |  |  |  | x |  |  |
|  |  |  |  | x | 0 |  |  |  |  |  | x |  |  |  |
|  |  |  |  | x | 0 | 3 |  |  |  | x |  |  |  |  |
|  |  |  | x | 3 | 0 | x | x | x | x |  |  |  |  |  |
|  |  |  |  | x | 0 | x |  |  |  |  |  |  |  |  |
|  |  |  | x | x | 0 | x |  |  |  |  |  |  |  |  |
|  |  | x |  | 1 | 0 | x | x |  |  |  |  |  |  |  |
|  | x |  |  |  | 0 | **1** |  | x |  |  |  |  |  |  |
|  |  | x | x | x | 0 |  | x | x |  |  |  |  |  |  |
|  |  |  |  | x | 0 | x |  |  |  |  |  |  |  |  |
|  |  |  | x | 2 | 0 | x | x |  | x |  |  |  |  |  |
|  |  |  | x |  | 0 | **2** |  | x |  | x |  |  |  |  |
|  |  |  |  | x | x | x |  |  |  |  | x |  |  |  |

其中，0代表开始涂的格子，0代表在这一层函数中涂的函数，x表示边界，i表示左边被记录的格以及其在数组中的顺序，**i**表示右边记录的格及在数组中的顺序。下一层递归需要涂的就是i与**i。**

* 1. 函数的递归：

函数的递归是在函数的调用及较优化的数据结构的基础上实现的。调用函数时，函数层数funcl加1（funcl用于判断新定义的变量存储在局部变量还是全局变量内）。使用gett\_char读取函数名，找到需调用的函数，之后用gett\_char读取并记录函数形参的数量(funcpanum)及名字，并存入variab list。之后用get\_para读取CALL函数之后括号内的实参并将其存入variab list里面形参的value。让q等于之前的valistnum值，然后将valistnum加上funcpanum。这样就可以区分开这个函数自身作用域的变量，逐层调用到它的那些函数定义的变量及全局变量，对于有重复名字的变量会优先从最小作用域内取值。程序会从函数的头部（FUNC的下一行）开始逐行读取。结束后，q与valistnum值回到调用此函数前的值，相应的variab list部分清空，funcl减1。之后程序读取CALL 的下一行。

在函数调用的基础上增加IF STOP设定好结束条件，就可以实现函数递归了。

* 1. IF语句：

IF STOP功能和IF END功能分别要求的格式为“IF 变量名 MORETHAN/LESSTHAN 变量名/具体数值 STOP”和“IF 变量名 MORETHAN/LESSTHAN 变量名/具体数值 THEN （换行） ………… END IF”。其中MORETHAN表示其左边的数大于右边，LESSTHAN反之。

读到IF语句后先读取其后的判断条件，通过gett\_float函数读取两个待判断的数据，并根据其关系决定判断条件成立与否。然后再读取其后的语句。如果是STOP，如果判断条件成立，则让行数增加600，超过当前所处的LOOP或FUNC的区域，使得在LOOP或FUNC那里循环或函数读取中止，继续读取这一层END LOOP或END FUNC之后的语句；不成立则继续读取下一行。如果是THEN，则执行其与END IF之间的语句；如果不成立则跳到END IF之后。

* 1. Debug：

Debug功能的实现内嵌于语句的读取与执行过程中，由gett\_char返回值，vairab/function list中检索变量/函数名的结果，画笔位置，语句格式等依据进行debug。bug包含错误和待提醒的问题两种。错误会导致程序无法运行，报出具体错误和其所在行并exit退出。待提醒的问题为不影响程序整体运行的小问题，如颜色取值越界。有数组用于记录某行是否已被检测过，以防重复进行“提醒”的debug（如循环中重复执行某语句）浪费时间。具体情况如下表：

|  |  |  |
| --- | --- | --- |
| 具体bug种类 | 输出 | exit返回值 |
| 待输入的文件不存在 | [error] inputfile cannot be opened correctly | 0 |
| 初始化指令（logo前三行）格式不正确 | [error] mistake exists in inputfile's basic structure | 0 |
| 初始画布大小超出限定范围 | [error][setting]size out of range | 0 |
| 初始颜色超出范围 | [caution][setting]color out of range | 无 |
| 初始位置超出画布 | [error][setting]start position out of range" | 1 |
| 输入的数据类型错误 | [error][line i]type error" | 2 |
| 变量名未定义 | [error][line i]undefined variab | 2 |
| 画笔位置超出画布 | [error][line i]pen out of range | 1 |
| 颜色超出范围 | [caution][line i]color out of range | 无 |
| 定义的变量名中不全是小写字母 | [error][line i]incorrect variab name | 2 |
| 重复定义变量 | [error][line i]redefine of变量名 | 2 |
| 请求的变量未（在有效作用域内）查找到 | [error][line i]this variab doesnot exist | 2 |
| 调用的函数未定义 | [error][line i]undefined function | 4 |
| LOOP缺少与之对应的END LOOP | [error][line i]LOOP without END | 3 |
| FUNC缺少与之对应的END FUNC | [error][line i]FUNC without END | 3 |
| IF缺少与之对应的END IF | [error][line i]IF without END | 3 |
| IF缺少MORETHAN或LESSTHAN | [error][line i]incorrect use of IF | 5 |
| IF 行缺少STOP或THEN | [error][line i]IF without related word | 5 |
| 行首非支持的关键字 | [error][line i]incorrect program | 6 |
| 输出为空 | [error] output error | 7 |

需要注意的是，i表示的是除了前三行基本设定之外，且除去空行的，有字的行中 的第（i+1）行，即i从0开始。

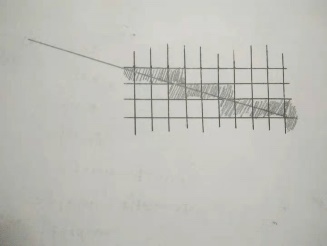
* 1. 有关画线函数：
     1. 画线函数的调用及前期准备：

画线函数的调用当且仅当在“乌龟”的“MOVE”的指令执行的时候，且每执行一次“MOVE”语句，依照LOGO语言的功能要求，在bmp位图上画一段从起点到终点的线段。bmp位图的结构是每一个像素点构成一个结构基元，即“单元格”。画线的实现是通过在bmp位图的部分像素点染成特定的颜色，通过这些像素点的排列来近似表示一条直线。

前期准备工作需要准备好一个二维数组，这个二维数组存放着该像素点的颜色数据，两个维度分别代表x轴和y轴。也就是说这个数组的部分存储单位与图像中的像素点一一对应。画线函数通过改变数组的取值便可以改变对应的像素点的颜色，实现画线。

* + 1. 画线函数的基本原理：

我们利用画线函数从起点至终点一一遍历每一个直线经过的像素，并将其染成“画笔”的颜色（已经设定好）。从最简单的情况（忽略线段的宽度）入手，由起点所在单元格向终点所在单元格沿着x轴的方向依次遍历。对起点x到[x]+1处直线的取值y([x]+1)，遍历纵坐标为[y]到[y([x]+1)]且横坐标为[x]的所有单元格并将其染为画笔的颜色。之后，对于y=y(x)直线的每一个自变量取值x，对横坐标为[x]，纵坐标从[y(x)]到[y(x+1)]的所有单元格遍历并染色。由于y=y(x)是线性方程，故[y(x)，y(x+1)]处的最值为[y(x)]和[y(x+1)]。

终点处的一列像素染法与起点格大致相同。由此，就遍历并染色了每一个线段经过的单元格。

（图1 宽度为0的线段部分经过画线函数的处理效果图）

* + 1. 画线函数的功能拓展及其实现方法探究：

在PS、Windows画图等软件中，均可以调节画笔的宽度，所以我们组也希望能将可调的画笔宽度应用于本次伪LOGO语言代码的指令实现中。我们经历了多次尝试，逐渐减小视觉误差。由于计算机对数据的存储依据有限的位模式。有限的位模式表示某一数量时不可能无限精确，所以就造成了客观上误差的存在。而且在某些情况下这些客观上的误差会集中体现，所以本组至今尚未实现不存在视觉误差的任意长度、宽度、朝向的统一线段绘制算法及其相互组合（例如边很粗的圆形）。

本组提出以下几种方案，其实现方法和优缺点各如下所示：

方案一：

置画笔的横截面为(2\*size+1)\*(2\*size+1)的一个正方形，在线段上，每遍历到一个点均将以其为中心的(2\*size+1)\*(2\*size+1)的正方形全部染上相同的颜色，这种方法的优点是代码复杂度较小，操作较为简便，但有两大缺点：

1.当线段斜45度向下时，线段的宽度为垂直于遍历方向的正方形的截线长度，为，在水平竖直方向则为（2\*size+1）；

2.线段的过渡点难以实现整齐的渐变（后面的颜色将前面的覆盖）。

方案二：

记录下线段移动的角度theta，作为函数的参数。仿照线段宽度为0时的情况，依照x从起点到终点进行遍历，并分别向上和向下多染（size/cos(theta)）（取绝对值）个单元格，这样，垂直其行进方向的宽度可以近似视为size。这个方案有一个缺点就是画连续图形的时候高度依赖于x轴的缩进，故在水平和水平接近的时候画线粗细较为准确，而在竖直直线及其附近的情况下竖直距离被严重拉长而水平宽度接近零，所以会变形，故该方法不适用于画连续封闭图形。

方案三：

与方案二同理，不过以y为优先标准进行遍历，这样在水平曲线的情况下误差较大。

方案四：

融合方案二和方案三的两大特点，对每一个函数进行两次遍历，一次以x优先进行遍历，再一次以y优先进行遍历，两次遍历的每处宽度最大值为该直线某处宽度值。该方案在45度时误差最大，当宽度size值在20以内时可以不用肉眼明显分辨出。但该方案有一个漏洞就是不支持多边形的渐变色画线，渐变色画线需要每一个基本单位内颜色统一，然而这种方法同方案1，后侧覆盖会占有前面的原有色块，导致原有色块的垂直行进方向着色不均匀。

方案五：

基于“方案四”遍历次数多且不易实现渐变色而改进。无论如何依照行进方向垂直方向截面大的方案进行遍历，即当行进方向与x轴所成角度在45度以内时以x优先进行遍历，否则以y优先遍历。遍历只有一遍，但当行进方向与x轴成45度角的时候不论以x还是y优先遍历都有难以抵消的误差，且大于方案四。

方案六：

采取分治的方案。求出四个边界点，对中间区段截取尽量大的有两边与坐标轴平行的整点平行四边形，对两边分别求三角形面积和两个端点所在行的染色。这样就可以画出一个标准长方形。这个方法对多边形非常行之有效，但对每一条线段长度均较小的圆，当线段较宽，且该算法构造的所有长方形在圆外侧会产生空隙，造成较大的误差。

综上所述，本组在画线宽度（2\*size+1）和移动距离相比较小时使用方案六，否则使用方案四，这样能使误差达到最小，且画出的图形为一连续图形。

1. 收获与体会
   1. 熊天翼：在这一次作业中，我从最开始面对CSDN上各种资料的惊慌失措到逐渐学懂文件的输入输出再到自己一点点开始实现logo语言中的基本语句，再到在过程中不断赶紧我们作业中的数据结构……作为本作业的开头者，我确实体会到了那种一点点看着自己的作品成长的过程所带来的快意。我发现，有的时候在电脑前干想真的不如就凭着自己的感觉先把代码写出来。船到桥头自然直，这话说的果然是一点不错；与其在问题中恐惧，不如先多尝试几步看看。

当然，也感谢我的的队友在作业完成的过程中所给予我的巨大支持与鼓励。团队的力量，一方面在于思维的碰撞，另一方面更在于一种相互支撑。我们三个一点基础都没有的人凑在一起，相比大佬们来说必定要花更多实现，但我们也在不断讨论的过程中共同对所学知识有了更深入的了解，并且更意识到我们这个团队的重要性。而这，正是大作业对我而言最重要的意义所在。

* 1. 李伟楷：做大作业的经历是我在一周内三次晚于3点睡觉，肝到爆炸但回想起来却无比难忘，收获巨大的一段经历。首先我深刻体会到了团队的力量。一队三人都是零基础小白，队长带头从零开始一点点学，一点点调试，逐步搭建起基本框架和数据结构。三人通力合作，如果没有彼此的支持和鼓励，我很难想象一个人完成这样一份工作需要多久。第二，我感受到了努力和学习的力量，感受到自己实实在在地用汗水换来了收获。两三周前的我还不敢想象解释器应该怎么做，现在我已经把它完成了，成就感max。尤其集中精力学习这项技术的几天里，感觉自己每天都“一日千里”地进步，不断战胜挫折。计算机是一门时常更新，日新月异的学科，需要从业者们不断努力学习新的理念。在这片知识海洋里，只要肯干，肯学，或许几周前自己还不敢想象的目标几周后就能达成了。第三，我领略到了编程的风采。大作业虽然很难，但和平时的小作业做题相比无疑有趣了很多，我从中看到了几百行代码能够实现的潜能，感受到了其无比的有趣。愿未来的我能更好地在这个领域里徜徉。最后，再次无比感谢队友们！以及感谢老师精心设计了本次作业。
  2. 徐栋：大作业是我这学期持续最长的肝DDL经历之一。整个过程可谓是经历了“九九八十一难”——从编程小白开始，经过老师两次讲解关于位图存储的原理与整体程序框架的搭建，又经过队友的第一步基础工作，然后在组长的文件输入、输出平台和他搭建的主体数据结构下写出画线函数进行运算和存储，最后与其他组员的程序进行汇总、对接、统一等。画线函数的功能是否强大直接决定画图质量的好坏，而它相比于实际画图，没有临时调整的空间，需要设计一套尽量囊括所有情况的算法；同样，也需要考虑在画线过程中画出的每一条线的起点和终点复杂的边界情况，边界情况的复杂度也远远比画线本身要复杂。总之，为了让图像更精确，我经历了非常漫长的调试过程。在此过程中我学会了独立思考、转化问题，学会了与他人交流。总之就是利用团体解决问题的一系列过程都在这次大作业的完成过程中体现出来，完成这次大作业与我而言收获良多。