



哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2019年春季学期

计算机学院大二软件构造课程

Lab 1实验报告

姓名	肖行文
学号	1170300316
班号	1703003
电子邮件	xiaoxingwen@stu.hit.edu.cn
手机号码	13266573776

目录

1 实验目标概述	1
2 实验环境配置	1
3 实验过程	2
3.1 Magic Squares	2
3.1.1 isLegalMagicSquare()	2
3.1.2 generateMagicSquare()	3
3.2 Turtle Graphics	3
3.2.1 Problem 1: Clone and import	4
3.2.2 Problem 3: Turtle graphics and drawSquare	4
3.2.3 Problem 5: Drawing polygons	4
3.2.4 Problem 6: Calculating headings	5
3.2.5 Problem 7: Personal art	5
3.2.6 Submitting	7
3.3 Social Network	7
3.3.1 设计/实现FriendshipGraph类	7
3.3.2 设计/实现Person类	7
3.3.3 设计/实现客户端代码main()	8
3.3.4 设计/实现测试用例	8
3.4 Tweet Tweet (选作, 额外记分)	9
4 实验进度记录	11
5 实验过程中遇到的困难与解决途径	12
6 实验过程中收获的经验、教训、感想	12

1 实验目标概述

- 训练基本的 Java 编程技能, 能够利用 Java OO 开发功能模块。
- 能够阅读已有代码框架并补全代码。
- 能为所开发的代码编写基本的测试程序并完成测试, 主要使用 Junit 测试。
- 能初步保证代码的正确性。
- 用 Git 作为代码排至管理的工具, 学会 Git 及 GitHub 的基本使用方法。

2 实验环境配置

配置实验环境的过程:

1. 下载 Java SE。
2. 系统自带 Git 所以不需要额外下载安装 Git。
3. 在 Eclipse 安装器的高级选项中安装 MIT 发行版的 Eclipse。
4. Java Tutor 和 Constellation 似乎是提供给 MIT 学生使用的, 无法获得证书, 所以没有开。
5. 根据实验文档中的提示修改了一些 Eclipse 的 Preference, 并且根据自己的习惯, 如补全规则和折叠, 对 Eclipse 的 Preference 进行了调整。
6. 配置 Git 的用户名和邮箱, 连接到自己的 GitHub 仓库。

遇到的问题:

1. 在 Git push 到 GitHub 的时候, Git 提示本地更新记录与 GitHub 的不一致。

解决: 因为在没有使用 Git 的时候, 曾经手动在网页端上传项目文件到 GitHub 仓库, 后来又用 Git 对本地仓库管理, 导致本地端与 GitHub 记录并不一致。

查询到命令 `git push -f`:

<https://stackoverflow.com/questions/44678942/difference-between-git-push-and-git-push-f>

强制覆盖了 GitHub 上的项目版本, 后可正常 push。

2. 官网给出的安装 Junit 过程过于繁琐。

解决: 使用 Eclipse 中自带的 Marketplace 进行安装。

GitHub 仓库:

<https://github.com/ComputerScienceHIT/Lab1-1170300316>

3 实验过程

请仔细对照实验手册，针对四个问题中的每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

为了条理清晰，可根据需要在各节增加三级标题。

3.1 Magic Squares

Magic Squares 是一种行、纵、对角线元素和相等的方阵。在这个实验中要求这个方阵的元素是正整数。

要求 1: 验证给出的 5 个矩阵是否都是 Magic Square，同时要处理 IO 异常，使用 `myLine.split()` 方法，使用 `Integer.valueOf()` 对给出矩阵中以字符串形式存储的数字转化为 `int` 类型的数字。

要求 2: 在加入 `generateMagicSquare(int n)` 后了解出现异常的含义，并对异常进行抓取提示。

3.1.1 isLegalMagicSquare()

该函数要做到防止以下类型的错误：文件中的数据不符合 Magic Square 的定义（行列数不相等、并非矩阵等）、矩阵中的某些数字并非正整数、数字之间并非使用 `\t` 分割、等。若遇到这些情况，终止程序执行（`isLegalMagicSquare` 函数返回 `false`），并在控制台输出错误提示信息。

这些错误的检测有一定的先后顺序，比如应该先检测数据类型是否符合规范，不能出现小数之类的错误，这个错误检测应该在对各行各列之和相等的检测之前。同时，对是否用 `\t` 分割的检测也应该在行数和列数检测之前。

这些检测一般来说可以借助文档中提到的函数功能，如不是用 `\t` 分割的部分被 `myLine.split()` 分割之后使用 `Integer.valueOf()` 会抛出异常，比如中夹有空格肯定是不能转化为 `int` 的。确定为 `int` 后可判定是否为正数。确定行列数是否相等只需要比较每行每列被分割出来的 `int` 个数就可以了。使用 `try catch` 抛出异常信息。

3.1.2 generateMagicSquare()

为了探究这个函数究竟是如何出错的，可以用 `try catch` 将 `for` 循环包括起来，这样即使输入数据非法，也不会直接报错退出，而会保留打印已经生成的一部分矩阵。

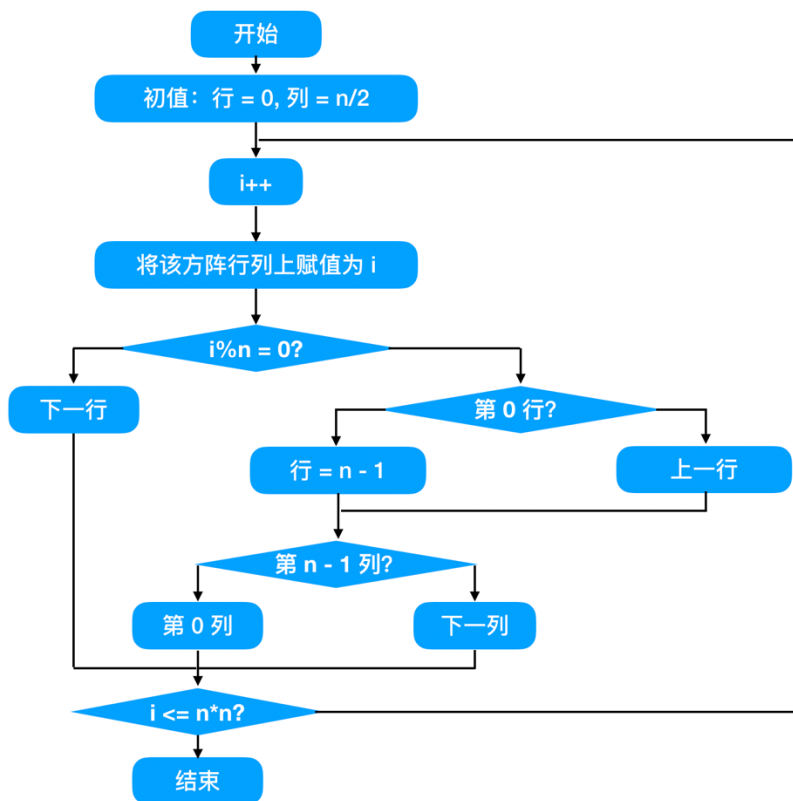
在输入偶数的时候，可以发现它生成的矩阵已经产生到了第 8 个数，这个数正好在产生的方阵底边，所以原因在这里：

```
if (i % n == 0) {
    // 原来的情况是在最后一行的情况下同时 row ++ 会越界，所以产生
    // java.lang.ArrayIndexOutOfBoundsException
    row++;
}
```

在输入负数的时候，这个矩阵根本无法创建：

```
// 如果 n 是负数，根本无法产生 magic 数组，所以会产生 java.lang.NegativeArraySizeException
try {
    int magic[][] = new int[n][n]; // 用 magic 二维数组存储 magic square
```

以下为该方法的流程图：



3.2 Turtle Graphics

最开始是关于对 GitHub 仓库克隆的一些指导，在配置环境中已经操作过了。

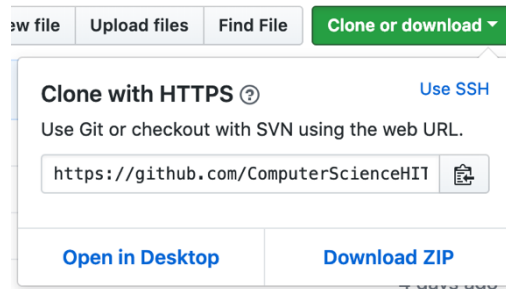
剩下部分是使用 MIT 开发的一种编程语言，内含一个叫 Turtle 的绘图工具，要求作画。其中 Problem 3 是画出一个正方形，Problem 5 是绘制多边形，这时候要使用 Junit 进行测试，当每个方法按照顺序测试通过之后，便可以使用最后一个方法调用前面已经通过测试的方法绘制多边形。这个过程是对 JUnit 作用的教学。Problem 6 和 Problem 7 是对数学问题的求解。Problem 8 是用 Turtle 自由创作，点开 Turtle 类可以发现我们可以变换画笔颜色。

每一步都要进行 Git 的更新。

3.2.1 Problem 1: Clone and import

如何从 GitHub 获取该任务的代码、在本地创建 git 仓库、使用 git 管理本地开发。

Clone GitHub 仓库需要利用 clone URL:



用 `git clone` 命令存到本地。Git add 命令提交指定文件到暂存区，`git add .` 是提交工作区的变化文件，`git commit` 为更新打注释。

3.2.2 Problem 3: Turtle graphics and drawSquare

调用 `forward()` 和 `turn()` 这两个方法，重复四次画出正方形。

```
public static void drawSquare(Turtle turtle, int sideLength) {  
    // throw new RuntimeException("implement me!");  
    turtle.forward(sideLength);  
    turtle.turn(90);  
    turtle.forward(sideLength);  
    turtle.turn(90);  
    turtle.forward(sideLength);  
    turtle.turn(90);  
    turtle.forward(sideLength);  
    turtle.turn(90);  
}
```

3.2.3 Problem 5: Drawing polygons

计算正多边形内角的数学公式为 $180 - (\text{sides} - 2) * 180 / \text{sides}$ ，由此完成 `calculateRegularPolygonAngle(int sides)`。

在 `drawRegularPolygon(Turtle turtle, int sides, int sideLength)` 中调用边数次 `calculateRegularPolygonAngle(int sides)` 进行转向, 便可以画出正多边形。

3.2.4 Problem 6: Calculating bearings

在 `calculateBearingToPoint()` 方法中通过二维坐标计算两点间转向过于复杂 (需要分类讨论), 所以采用极坐标计算, 所幸 Java 提供 `Math.atan2` 方法可以直接算出极坐标, 而且范围在 $-\pi$ 到 π 之间, `Math.atan2(targetX - currentX, targetY - currentY) * 180 / Math.PI` 便为两点间的角度差, 减去出发点原偏向可求出该函数要求的返回值。

`calculateBearings` 直接调用数次 `calculateBearingToPoint()`。

3.2.5 Problem 6: Convex Hull

题目说明给出了礼物 gift-wrapping 算法, 即先找到所有点中最左边的一点

```
for (int i = 1; i < n; i++)  
    if (pointsArray[i].x() < pointsArray[l].x())  
        l = i;
```

然后再找到以该点为原点, 以另一点为终点, 使得它们连线的角度最小, 然后设终点为起点, 如此循环, 直到回到第一点。下面这个函数用来找到角最小的点。

```
for (int i = 0; i < n; i++) {  
    if (orientation(pointsArray[p], pointsArray[i], pointsArray[q]) == 2)  
        q = i;  
}
```

为了实现这个功能, 加入函数 `orientation()` 以判断该点是在相对点 x 轴平行方向上面还是下面, 如果无法找到更“下面”的一点, 那么就设为终点。

3.2.6 Problem 7: Personal art

文档中给出了一些示例: [Here are some examples](#)。

同时 turtle 中可以用枚举类型 PenColor 修改颜色。

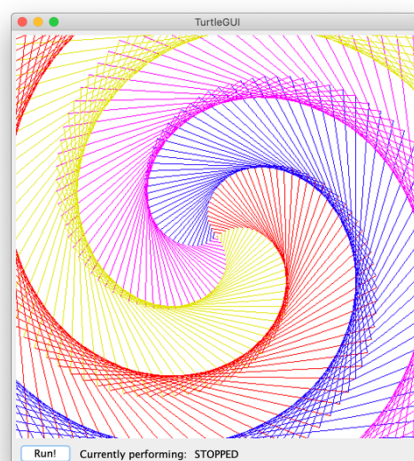
```
package P2.turtle;

/**
 * Enumeration of turtle pen colors.
 *
 * You may not modify this enumeration.
 */
public enum PenColor {
    BLACK,
    GRAY,
    RED,
    PINK,
    ORANGE,
    YELLOW,
    GREEN,
    CYAN,
    BLUE,
    MAGENTA;
}
```

为了方便周期性修改颜色，在 turtlesoup 中创建如下方法：

```
public static PenColor paintColor(int colorCode) {
    switch (colorCode) {
        case 0:
            return PenColor.RED;
        case 1:
            return PenColor.YELLOW;
        case 2:
            return PenColor.MAGENTA;
        case 3:
            return PenColor.BLUE;
    }
    return PenColor.BLACK;
}
```

这样能够通过周期性数字变化更换画笔颜色，最后实现了如下图案：



3.2.7 Submitting

有 3 条代码在 Terminal 中必须执行:

1. `git add ./<filename>` 表示将哪些文件加入暂存区, 一般是用 `git add .` 将工作区所有变化文件加入暂存区。
2. 用 `git commit -m "****"` 确认更新及注释。
3. `git push **` 执行推送到 GitHub 仓库。一般是 `git push origin master` 推到主分支。

3.3 Social Network

本实验是对一个社交网络的模拟, 社交网络被抽象成无向图。这个图要有如下的功能: 能添加顶点、添加边以及输出两点之间的最短距离。

3.4 设计/实现 FriendshipGraph 类

根据题意, 互相两个 Person 之间的关系有如下几种情况: 1. 无关联; 2. 单向连接; 3. 互相连接。为这几种情况分别使用不同的标记:

```
public static final int INF = Integer.MAX_VALUE;  
public static final int CONNECTED = -2;
```

这种标记方便实现寻找最短路的算法。遍历所有的边添加二维图每个顶点的信息。

```
public void addEdge(Person person1, Person person2) {  
    relationGraph[friendName.indexOf(person1)][friendName.indexOf(person2)] = 1;  
}
```

由于互相相连的两个顶点之间的长度被设为 1, 所以直接找两任意两点间的最短路可以得到 `getDistance` 结果。

3.4.1 设计/实现 Person 类

Person 类只用存储姓名, 所以只有一个字符串。

```
public class Person {  
    String name = new String();  
  
    public Person(String name) {  
        this.name = name;  
    }  
}
```

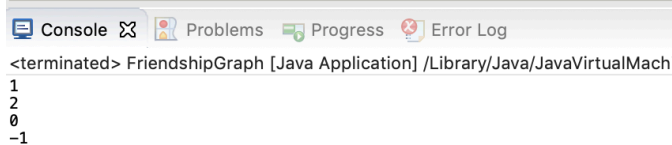
3.4.2 设计/实现客户端代码main()

依照题目文档中的代码当做 main() 内容。运行结果如下:

```

89 public static void main(String[] args) {
90     FriendshipGraph graph = new FriendshipGraph();
91     Person rachel = new Person("Rachel");
92     Person ross = new Person("Ross");
93     Person ben = new Person("Ben");
94     Person kramer = new Person("Kramer");
95     graph.addVertex(rachel);
96     graph.addVertex(ross);
97     graph.addVertex(ben);
98     graph.addVertex(kramer);
99     graph.addEdge(rachel, ross);
100    graph.addEdge(ross, rachel);
101    graph.addEdge(ross, ben);
102    graph.addEdge(ben, ross);
103    System.out.println(graph.getDistance(rachel, ross));
104    // should print 1
105    System.out.println(graph.getDistance(rachel, ben));
106    // should print 2
107    System.out.println(graph.getDistance(rachel, rachel));
108    // should print 0
109    System.out.println(graph.getDistance(rachel, kramer));
110    // should print -1
111 }
112 }

```



```

<terminated> FriendshipGraph [Java Application] /Library/Java/JavaVirtualMachi
1
2
0
-1

```

根据代码, Rachel 和 Ross 有关系, Ross 和 Ben 有关系, Kramer 独立, 所以结果满足题目要求。

3.4.3 设计/实现测试用例

首先添加源数据:

```

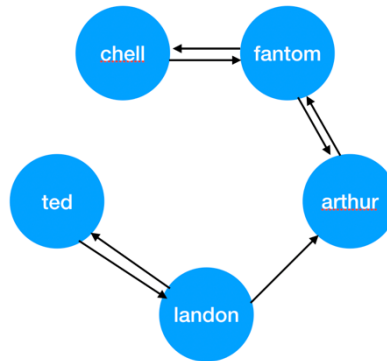
Person chell = new Person("Chell");
Person ted = new Person("Ted");
Person landon = new Person("Landon");
Person arthur = new Person("Arthur");
Person fantom = new Person("Fantom");
Person anyoneelse = new Person("Anyoneelse");
graph.addVertex(chell);
graph.addVertex(ted);
graph.addVertex(landon);
graph.addVertex(arthur);
graph.addVertex(fantom);
graph.addEdge(chell, ted);
graph.addEdge(chell, fantom);
graph.addEdge(ted, landon);
graph.addEdge(landon, ted);
graph.addEdge(landon, arthur);
graph.addEdge(arthur, fantom);
graph.addEdge(fantom, arthur);
graph.addEdge(fantom, chell);

```

为了方便验证，在 `addEdge` 中已经添加过验证边是否已经添加的代码，直接验证返回值是 `true` 还是 `false`。

```
assertFalse(graph.addEdge(chell, ted));
assertFalse(graph.addEdge(ted, landon));
assertTrue(graph.addEdge(landon, ted));
assertFalse(graph.addEdge(landon, arthur));
assertFalse(graph.addEdge(arthur, fantom));
assertTrue(graph.addEdge(fantom, arthur));
```

添加的边实际上形成了如下的图，考虑到了双向连通，单向连通（在结果中被视为不连通）以及完全不连通三种等价类，可知结果正确。



3.5 Tweet Tweet（选作，额外记分）

该实验是对社交网络的一个模拟。实验主要的输入内容是不同用户发的 `tweet`，`tweet` 中包含着各种信息，包括发推人、时间戳、@的人。共有信息提取、过滤搜索、好友推测以及一个更智能的好友系统功能。

3.5.1 从 tweets 中提取信息

`getTimespan()` 要求从已有 `tweets` 中获得一个最小的时间跨度，使得这个时间跨度能覆盖已有所有的 `tweets`。只需要求出最后一个 `tweets` 时间及最早一个 `tweets` 时间，那么这两个 `tweets` 的时间差就是最小时间跨度。

`getMentionedUsers()` 在 `spec` 中给出一个规则，及被 @ 的用户前后都不得是用户名的合法字符，根据这点可以判断是否是被 @ 的用户。被 @ 的用户位于一条 `tweets` 语句的开头和结尾时需要特别注意，因为这时用户前或后是没有字符的，应该单独判断。

3.5.2 过滤 tweets

分别是按照姓名、时间跨度和关键字筛选 `tweets`。直接遍历所有 `tweets` 寻找就可以。

```
for (Tweet element : tweets) {
    for (String key : words) {
```

3.5.3 推测社交网络

通过一个人@另外一个用户来推测关注关系，只需要调用 3.5.1 中的 `getMentionedUsers()`，将返回的用户与发推者建立映射关系。需要注意的是 `getMentionedUsers()` 的参数是一个 list，需要将 tweets 转化为 list，同时有用户会 @自己，这种情况需要排除在外。

3.5.4 智能化

这里选择第二种推测方式，即三角闭包。依题得，A 与 B 互关，B 与 C 互关，则 A 与 C 互关。需要注意的是如果 C 等于 A 则需要排除在外。

并且建立的 smarter 关系网要不同于 3.5.3 中的关系网。可以先复制 3.5.3 的关系网，依据它进行判断，将新的关系加入新的 smarter 关系网。如果沿用 3.5.3 的关系网，关系网的每个划分最终会进化成一个完全图，不合题意。

3.5.6 测试样例

以下是 `getMentionedUsers()` 的测试样例：

```
"TeddyBear", "you Hello my love @Chell. @Chell! A nice @His day! My new email:tb@hotmail.com", d1);  
"Chell", "@TeddyBear U misstake @dad me as chell. U may wanna @chell", d2);
```

考虑到了被@人出现在开头、结尾、@符号出现在邮件中、@自己这几个等价类，结果正确。

以下是 `guessFollowGraph()` 的样例：

```
Tweet tweet0 = new Tweet(1, "Tom", "I don't want to invite @Mary to my party.", d1);  
Tweet tweet1 = new Tweet(1, "Tom", "@Sam should be included in the invitation list.", d1);  
Tweet tweet2 = new Tweet(2, "Sam", "Thank you @Tom.", d1);  
Tweet tweet3 = new Tweet(3, "Mary", "@Tom such an incredible freak.", d1);  
Tweet tweet4 = new Tweet(1, "Tom", "Do not bother @Sam!", d2);  
Tweet tweet5 = new Tweet(4, "Chell", "Go reply @Mary, Tom.", d2);  
Tweet tweet6 = new Tweet(4, "Chell", "You are so rude.", d1);  
Tweet tweet7 = new Tweet(5, "Bruce", "I'll notify her.", d2);  
Tweet tweet8 = new Tweet(5, "Bruce", "Hi @Mary, Tom didn't invite you.", d1);  
Tweet tweet9 = new Tweet(7, "Sam", "It's a pity I don't agree with @Bruce.", d1);  
Tweet tweet10 = new Tweet(5, "Bruce", "@Sam's stupid. From @Bruce.", d1);
```

考虑到单向关注、互关、三角闭包、未关注、关注了不存在的用户等情况，测试相对完备，结果正确。

同时这个样例还能完成 `influencers()` 的测试：

```

List<String> verificationList = SocialNetwork.influencers(followsGraph
List<String> resultList = new ArrayList<String>();
resultList.add("mary");
resultList.add("sam");
resultList.add("bruce");
resultList.add("tom");
resultList.add("chell");
assertEquals(resultList, verificationList);

```

结果正确。

4 实验进度记录

请尽可能详细的记录你的进度情况。

日期	时间段	计划任务	实际完成情况
2019-02-28	13:45-15:30	按照课程官网指导配置完环境	按计划完成
2019-03-01	13:45-15:30	完成 P1 的 isLegalMagicSquare() 方法	按计划完成
2019-03-02	13:45-15:30	完成 P1 剩余部分并提交至 GitHub	按计划完成
2019-03-03	13:45-17:00	按照 P2 要求对实验材料进行环境配置、导入	开始遇到困难无法使用 JUnit, 后解决
2019-03-04	15:00-18:00	完成 drawSquare, Drawing polygons 和 Calculating bearings	按计划完成
2019-03-06	19:00-21:00	完成 P2 的剩余部分	遇到了一些数学上的困难, 但按时完成
2019-03-07	18:00-21:00	完成 P3	按计划完成
2019-03-09	19:30-21:00	完成 P4 的问题 1 两个方法	按计划完成
2019-03-10	20:00-22:00	初步完成 P4 剩余问题	按计划完成
2019-03-11	20:00-22:00	写 P4 的 test 文件, 对 P4 debug	作业量大, 未完成
2019-03-12	20:00-22:00	写 P4 的 test 文件, 对 P4 debug	按计划完成
2019-03-13	14:00-16:30	检查 P4, 修改一些细节, 推送到 GitHub	按计划完成
2019-03-15	13:45-18:30	检查所有实验内容, 补全实验报告	按计划完成

5 实验过程中遇到的困难与解决途径

1. 对于 Java 一些方便的函数不熟悉用法。
解决：一般是直接看 Oracle 官方的 Java 用户帮助，对于适用范围会有更详尽的了解。
2. 对面向对象的一些特性不够熟悉的时候。
解决：一般是做一些实验 + 搜索解决问题。大部分的问题其实都在 Java 帮助里有提到。
3. Eclipse 编码不统一。
解决：Eclipse 的 Windows 版用的默认编码是 GBK，这并不是一个现行通用的标准，所以在 UTF 编码的环境中会出现乱码。在偏好设置中改成 UTF-8 编码就可以解决问题。

6 实验过程中收获的经验、教训、感想

本节除了总结你在实验过程中收获的经验教训，也可就以下方面谈谈你的感受（非必须）：

- (1) Java 编程语言是否对你的口味？
Java 很方便，提供很灵活的编程方式，总体觉得还可以。适合于多人协作的项目。
- (2) 关于 Eclipse IDE
Eclipse IDE 相较于 Visual Studio 功能比较缺乏，但是经过适当配置之后任处于可以使用的状态。
- (3) 关于 Git
Git 的引入让文件记录管理变得更加方便，很容易追溯到出错的点，也容易恢复程序到修改之前的状态。
- (4) 关于 CMU 和 MIT 的作业
CMU 和 MIT 官方课程作业说明非常详尽，提供了很大的帮助。他们对课程的设计作出了大量的工作。几乎我的疑问都能被他们提到，不用费心找我需要的资料。
- (5) 关于本实验的工作量、难度、deadline
如果每天花一定时间做的话，工作量和难度不算大。难度主要体现在找自己写的 bug。
- (6) 关于初接触“软件构造”课程

最初很惊讶这门课程所用到的资源之多。因为 piazza 本来是国外大学所用，国内采用 piazza 作为答疑平台的学校不多，问问题很方便。采用说明详尽的作业也很好，不用担心作业目标含糊不清的状况，难度不会跳跃过大，总体是循序渐进的。