Background

在本次课程训练中，将展示你对文件读写、字符串处理、动态内存管理和链表数据结构的理解，以及在程序设计、测试和调试方面的能力。在此次训练中，你将学习路线的规划，并实现简单的路线重新规划机制。

基于网格的路线（重新）规划

路线规划用于自动驾驶人员的导航，例如自动驾驶车辆、机器人。基于网格的路线计划是研究在二维空间中构造从初始格点到目标格点的路线问题，该二维空间又分为阻塞栅格和空栅格。图1a显示了一个由10行和10列组成的示例网格，第0行、第0列的初始单元格记为I，第9行、第9列的目标记为G。绿色箭头表示从I到G的计划移动，绕开了显示为橙色正方形的块。

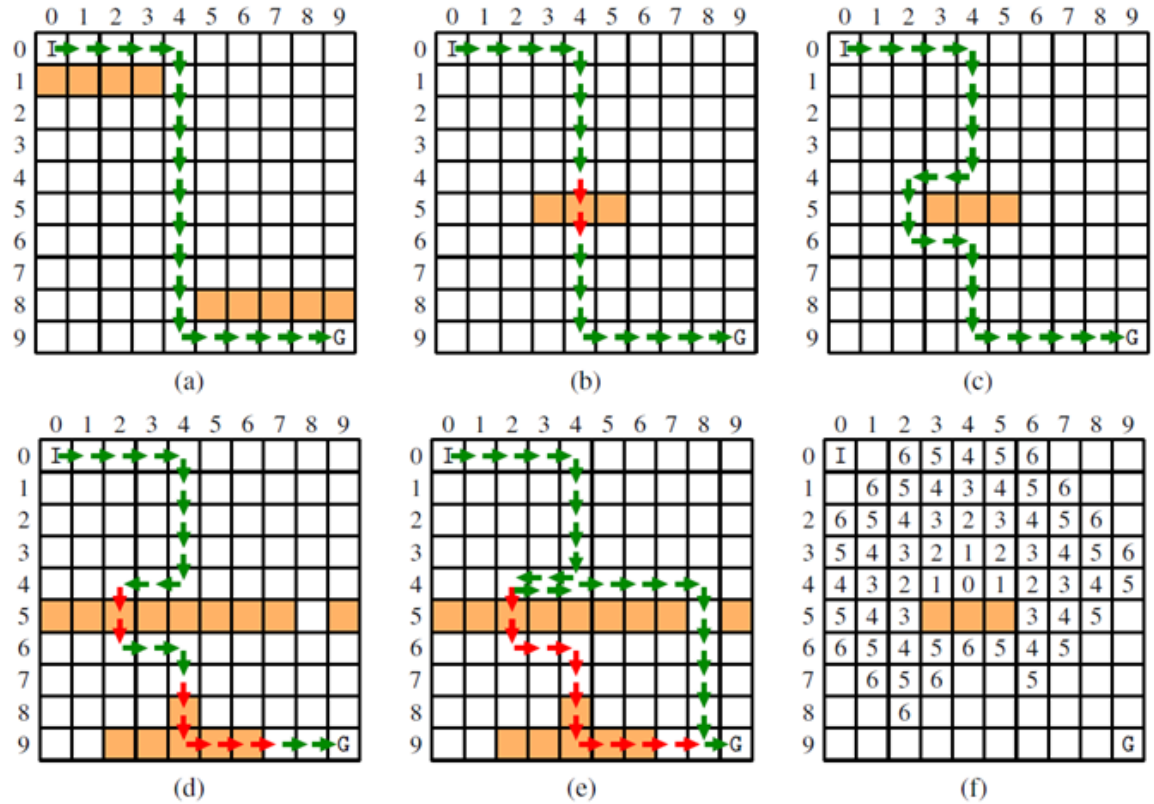


图1：带有块和路线的网格

随着阻塞栅格位置的变化，例如，阻塞栅格被移除或放置在的格点中，原计划的路线可能会变得无效。图1b显示了图1a的网格，其中包含新的阻塞栅格。 图1a中的原始路线在图1b中无效，因为它穿过了阻塞栅格，见图中的红色箭头。若可得知阻塞栅格的位置信息，则可重新规划路线避开阻塞栅格到达目的地，如图1c所示。图1d显示了阻塞栅格的后续分布，该分布在两个位置中断了图1c重新规划的路线。如果尝试遵循该路线，将在第5行、第2列的单元格中被首次阻断。若更新阻塞栅格的位置信息，则可重新规划路线，如图1e中的绿色箭头所示，图中的红色路线为无效路线，应舍弃。

训练任务

在VS 2010软件的程序编写界面进行输入，输入的第一行为网格的尺寸，例如10x20代表10行20列。输入的第二行为编码网格中的初始单元格，而第三行为目标单元格。单元格使用[r，c]格式编码，其中r和c是分别代表单元格的坐标。随后的输入指定了网格中块的位置，每行一个块单元，表示栅格信息输入结束。之后开始输入特定的路线信息（该路线自定，可以为正确行走路线，也可为错误路线）。路线由交替的单元格编码，用->表示，例如[0,0]-> [0,1]-> [0,2]编码从单元格[0,0]开始，然后依次到达[0,1]、[0,2]；路线的编码过程中不能使用空格或制表符，而->后面可以包含一个换行符。具体如下所示（坐标前的数字为行数）：

```
1  10x10
2  [0,0]
3  [9,9]
4  [1,0]
5  [1,1]
6  [1,2]
7  [1,3]
8  [8,9]
9  [8,8]
10 [8,7]
11 [8,6]
12 [8,5]
13 $
14 [0,0]->[0,1]->
15 [0,2]->[0,3]->
16 [0,4]->[1,4]->
17 [2,4]->[3,4]->
18 [4,4]->[5,4]->
19 [6,4]->[7,4]->
20 [8,4]->[9,4]->
21 [9,5]->[9,6]->
22 [9,7]->[9,8]->
23 [9,9]
24
```

阶段1读取和分析输入数据

输入相关网格以及路径信息，并将其输出，以确保输入正确。首先，程序应确保所提供的特定路线有效，即它在初始单元格中开始，在目标单元格中结束，由合法移动（一个单元格移动且没有对角线移动）组成，并且避开有阻塞栅格。该阶段所需的输出内容如下所示：

```
mac: ass2-soln < test0.txt
==STAGE 0========================================
The grid has 10 rows and 10 columns.
The grid has 9 block(s).
The initial cell in the grid is [0,0].
The goal cell in the grid is [9,9].
The proposed route in the grid is:
[0,0]->[0,1]->[0,2]->[0,3]->[0,4]->
[1,4]->[2,4]->[3,4]->[4,4]->[5,4]->
[6,4]->[7,4]->[8,4]->[9,4]->[9,5]->
[9,6]->[9,7]->[9,8]->[9,9].
The route is valid!
```

输出的最后一行内容是以下五种状态之一：

Status 1: Initial cell in the route is wrong!
Status 2: Goal cell in the route is wrong!
Status 3: There is an illegal move in this route!
Status 4: There is a block on this route!
Status 5: The route is valid!

如果路径中的起始栅格与第2行中提供的初始单元格不同，则输出状态1。如果路线中的终点与第3行给出的目标单元格不同，则输出状态2。如果路线包含的移动跨越一个以上的单元格，则输出状态3。 状态4报告该路径未避开阻塞栅格。若输入的路线无以上四种情况，则输出状态5。状态检查应按照显示的顺序进行，如果具有某种状态，则应进行报告，并且不进行下一步检查。输出路线时，每行最多输出五个单元格，并用->隔开。输出过程中，请勿使用空格或制表符。换行符可以跟在->之后。

阶段2绘图和重新规划

扩展程序为可视化输出栅格与路线，并尝试修复断开的路线，最终可视化输出已修复的路线。所有第0阶段的输出都应保留，然后第1阶段的输出应从以下这一行开始：

```
  0123456789
0I****
1####*
2    *
3    *
4    *
5    *
6    *
7    *
8    ******
9    *****G
```
(a)

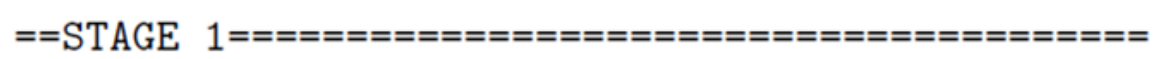
```
  0123456789
0I****
1    *
2    *
3    *
4    *
5   ###
6    *
7    *
8    *
9    *****G
```
(b)

```
  0123456789
0I****
1    *
2    *
3    *
4   ***
5   *###
6   ***
7    *
8    *
9    *****G
```
(c)

```
  0123456789
0I****
1    *
2    *
3    *
4   ***
5########  #
6   ***
7    *
8    #
9   #######G
```
(d)

```
  0123456789
0I****
1    *
2    *
3    *
4   *******
5########*#
6        *
7        *
8    #   *
9   ##### *G
```
(e)

图2：使用ASCII字符的网格可视化示例



随后应使用ASCII字符可视化网格中的输入路径，如图2所示；请注意，图2中的五个子图中的每个子图都使用11行进行编码，其中每行包含11个字符，空白栅格使用ASCII码32的字符编码。例如，图2a显示了例程的栅格和路线的ASCII编码（也显示在图1a中）。

在ASCII可视化中，初始栅格被编码为字符I，目标栅格被编码为G，阻塞栅格被编码为＃，路线用*表示。如果单元格是起始栅格、目标栅格或阻塞栅格，则应分别打印，G或＃，而不是*。如果输入路线的状态不等于4，请参见阶段0的描述，阶段1的输出终止。否则，程序应尝试修复路线中第一个被阻断的部分，并可视化输出网格中已修复的路线，以及路线的所有步骤和状态修复路线。例如，图2b显示了来自例程文件的栅格和路线的ASCII可视化（也如图1b所示），而图2c显示了相应的修复路径的ASCII可视化（也如图1c所示）。输入网格和路径的可视化、栅格中已修复路径的可视化以及已修复路径的步骤和状态应由下面的行分隔。

修复后的步行的输出应遵循阶段0中给出的指示。

以1b的修复过程进程举例分析：路线中存在阻塞块，需对其进行修复，阻塞块起始于[4,4]，终止于[6,4]，为了便于修复，建立组合([a,b],c)，其中a、b分别为坐标值，c为0或1，([a,b],0)表示坐标为[a,b]的网格为阻塞块，否则为空白栅格。对于1b中路线的阻断处[4,4]按上下左右的顺序建立组合{([3,4],1)([5,4],0),([4,3],1),([4,5],1)}，由此组合可知，除了向下，路线向其他方向修复都为可行方案。现规定修复方向的优先级：向下修复>向左修复>向右修复>向上修复，则路线从[4,4]开始向右修复。再将[4,3]作为中心点建立上述组合，进行下一步修复，如此循环反复，即可到达目的地。

请注意，上述过程应可以修复多个断开的栅格，如图1d中的断开路径以及从图1e中所示的单元格[4,2]开始修复其断开的段的结果（ASCII 可视化分别在图2d和图2e中显示）。

利用你的创造力和扩展的ASCII和/或UTF字符，以有趣，艺术的方式输出栅格，并与我们分享你的杰作。