Assignment One Report

2022 2班 杨璐玮

* **Step1——17\*17迷宫问题的解决:**

1. **阅读并理解提供的原代码思路：**
2. **阅读并理解变量与函数等的含义和作用**

* posi：数组posi来存放每一步所到的位置
* Int pos：pos通过对宽度取模和除以宽度，分别得到该点的坐标（x，y）。也就是说，每一个点的序号是按从左到右从上到下的顺序编排的。
* enum Direction {DOWN, LEFT, UP, RIGHT};//下左上右顺时针
* void FindEntrance(int&);用于找到入口
* bool AtExit(int);用于判断是否到达出口
* void ReportPosition(int);用于输出坐标
* void WheresRight(int,Direction,int&);找当前朝向的右边位置的序号
* bool Wall(int)用于判断某个位置是否可以通过
* void TurnRight(Direction&);用于右转
* void MoveForward(int&,Direction);用于向前走
* void WheresAhead(int,Direction,int&);找当前朝向的前方位置的序号
* void TurnLeft(Direction&);用于左转
* int counter;记录步数（路径长度）

1. **整体思路：**

从起点出发。对于每一个当前的位置，依次判断右方/前方是否可走，如果可走则执行右转/直走函数（优先向右转），如果都不可走执行左转函数，继续判断。位置的变化通过改变pos的数值实现，同时用posi记录下每一步的位置序号，便于后续使用ReportPosition函数打印路径。只要还没到终点，以上步骤在while循环里重复执行直到AtExit函数返回true再结束循环。

1. **改写原代码：**
2. **读取文件**

将题目文件夹中的maze1.txt等文本文件放入项目文件夹，使用原代码中的bool LoadMaze(const char fname[])函数实现读取。

1. **构造所需重载运算符：**

//Overload重载enum的前置自增运算符，注意到对于enum边界的特殊判断

Direction& operator++(Direction& direction)

{

if (direction == RIGHT)

{

direction = static\_cast<Direction>(0);

return direction;

}

direction = static\_cast<Direction>(direction + 1);

return (direction);

}

Direction& operator--(Direction& direction)

{

if (direction == DOWN)

{

direction = static\_cast<Direction>(3);

return direction;

}

direction = static\_cast<Direction>(direction - 1);

return (direction);

}

// This struct is used for locations within the maze

1. **补充关键函数，实现方向转换和朝向位置的判断：**

void TurnRight(Direction& heading)

{

++heading;

}

void MoveForward(int& pos, Direction heading)

{

switch (heading)

{

case(DOWN):

pos += mazeWidth;

break;

case(LEFT):

pos --;

break;

case(UP):

pos -= mazeWidth;

break;

case(RIGHT):

pos ++;

break;

}

}

void WheresAhead(int pos, Direction heading, int& ahead)

{

ahead = pos;

switch (heading)

{

case DOWN:

{

ahead+=mazeWidth;

break;

}

case LEFT:

{

ahead --;

break;

}

case UP:

{

ahead-=mazeWidth;

break;

}

case RIGHT:

{

ahead ++;

}

}

}

// This function changes heading by turning left

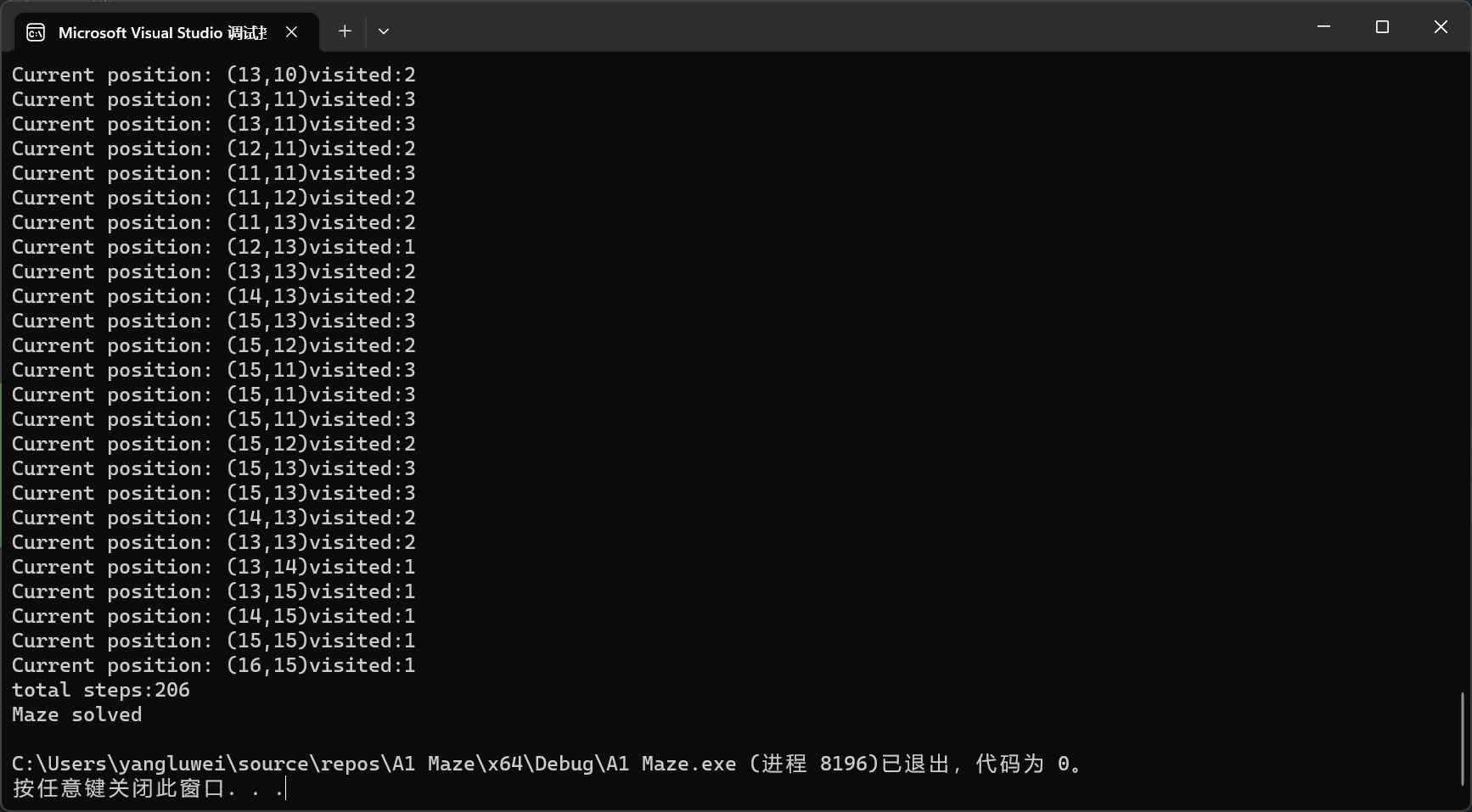
void TurnLeft(Direction& heading)

{

--heading;

}

Maze1.txt的实现效果：



* **Step2——删除回溯:**

1. **Visited数组的创建和初始化**

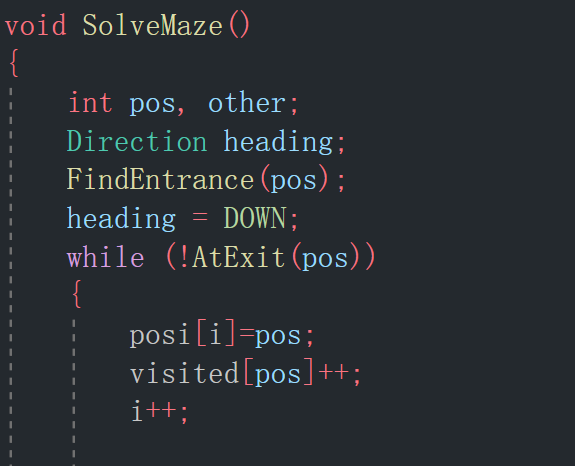
//Step2:删除回溯，创建一个数组visited检查是否经过某点，元素值代表路径经过该点的次数

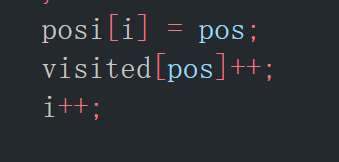
int visited[17 \* 17];

//初始化visited数组所有元素值为0

memset(visited, 0, sizeof(int) \*17\* 17);

1. **visited元素值的变化**





两个地方有坐标变化的代码。这两个地方都增加语句：visited[pos]++;

每次坐标位置变化的时候，新的位置被经过的次数自增一次。

1. **判断重复路径并输出**

在solvemaze（）函数的最后一部分是路径的输出。此处需要添加对于重复路径的判断。

走回头路的时候，重复经过的点会被标上2，因此如果检查到visited[posi[j]]的值是大于1的，则不予输出。同时考虑到原路径和新路径会有交叉点，这个交叉的点也会被标上大于1的值（通常是2），但是仍然是新路径的必经之路，因此应该为交叉点增添一个判断：注意到如果某一点是交叉点，他的下一个点的visited值应该为1。所以如果检查到visited中当前位置被标记了2的同时下一个位置也被标记了2，则该点必然不是交叉点而是重复路径上的一般点，可以肯定地去除。由此实现了重合路径的删除。输出的即为从起点直达终点的路径。

int counter=0;

for(int j=0;j<i;j++)

{

if (posi[j] < 0)

continue;

//增添了对于重复路径的判断

if ((j<i-1)&&(visited[posi[j]] > 1) && (visited[posi[j + 1]] > 1))

continue;

cout << "Current position: (" << posi[j]/mazeWidth << ',' << posi[j]%mazeWidth << ')' << "visited:"<<visited[posi[j]] << endl;

counter++;

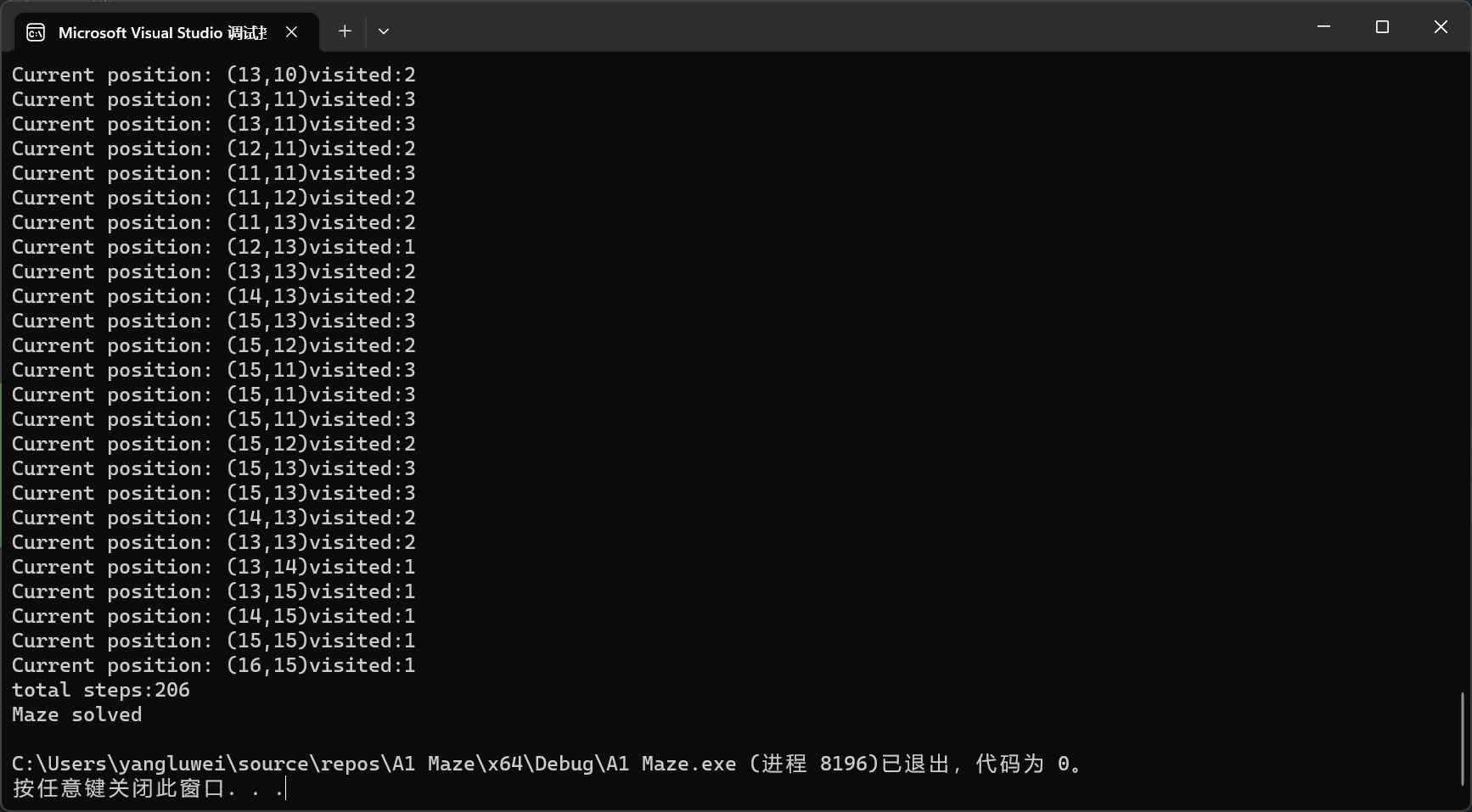
}

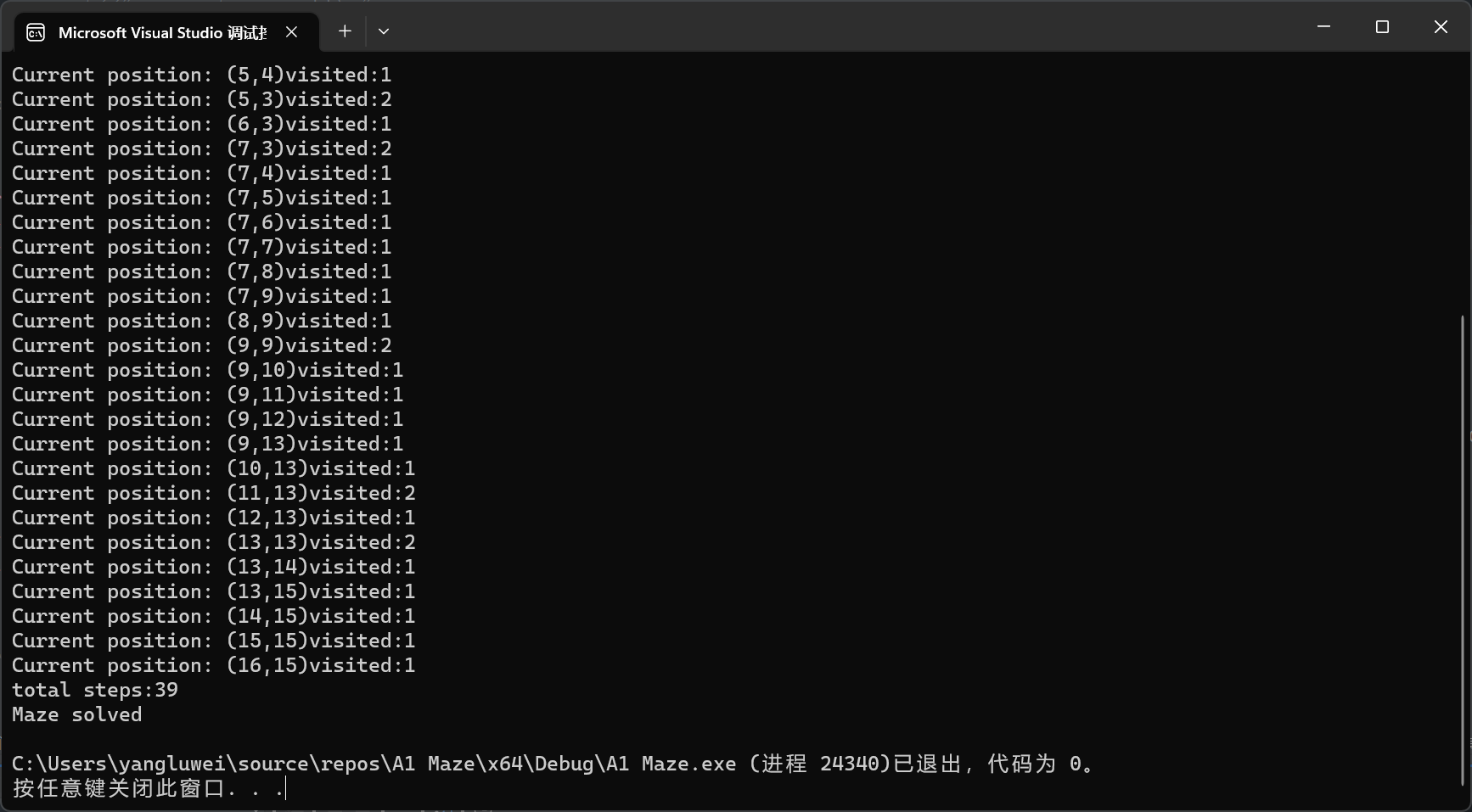
cout<<"total steps:"<<counter<<endl;

cout << "Maze solved" << endl;

1. **删除重合路径前后比较**

Maze1.txt





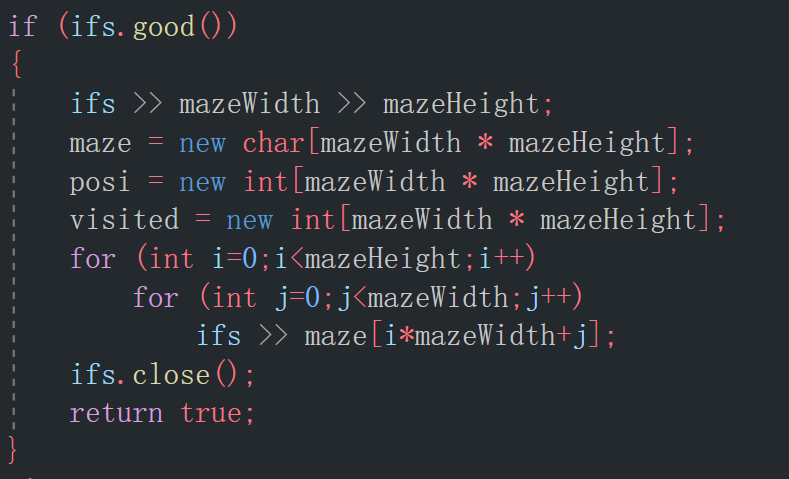
* **Step3动态读取:**

1. **数组改写**

原本用到的三个数组全部改写为在堆上创建的动态数组



在读入文件之后再根据读取到的迷宫大小进行初始化。



1. **去除错误提示和限制**

由于需要动态读取迷宫大小和迷宫，而且未知规模。因此原代码中限制并报错的代码块应该删去或者注释掉。否则在读入maze3.txt的时候会报错。

/\*

if(i>=289)

{

cout<<"array too small\n";

abort();

}

\*/

/\*

if(i>=400)

{

cout<<"array too small\n";

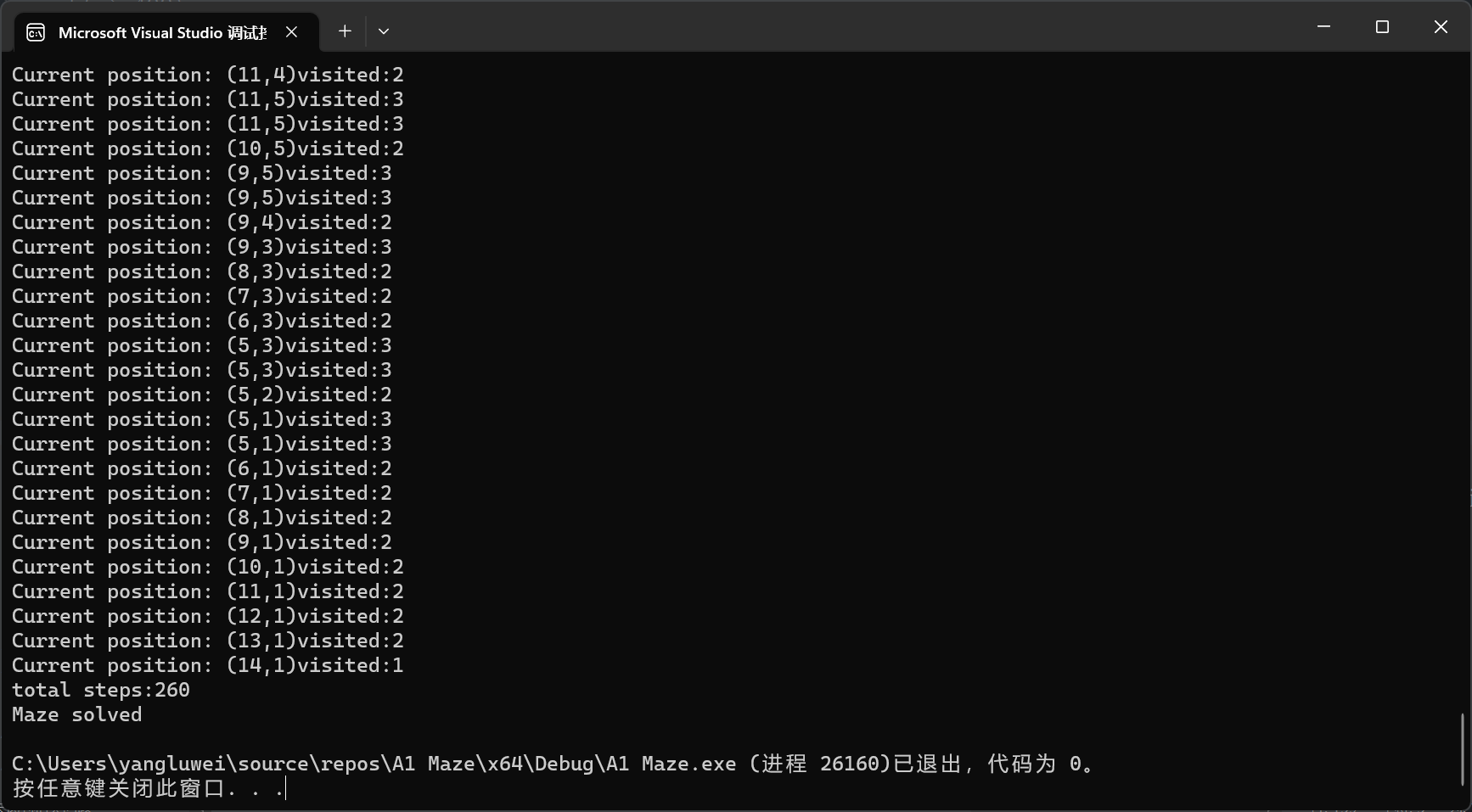
abort();

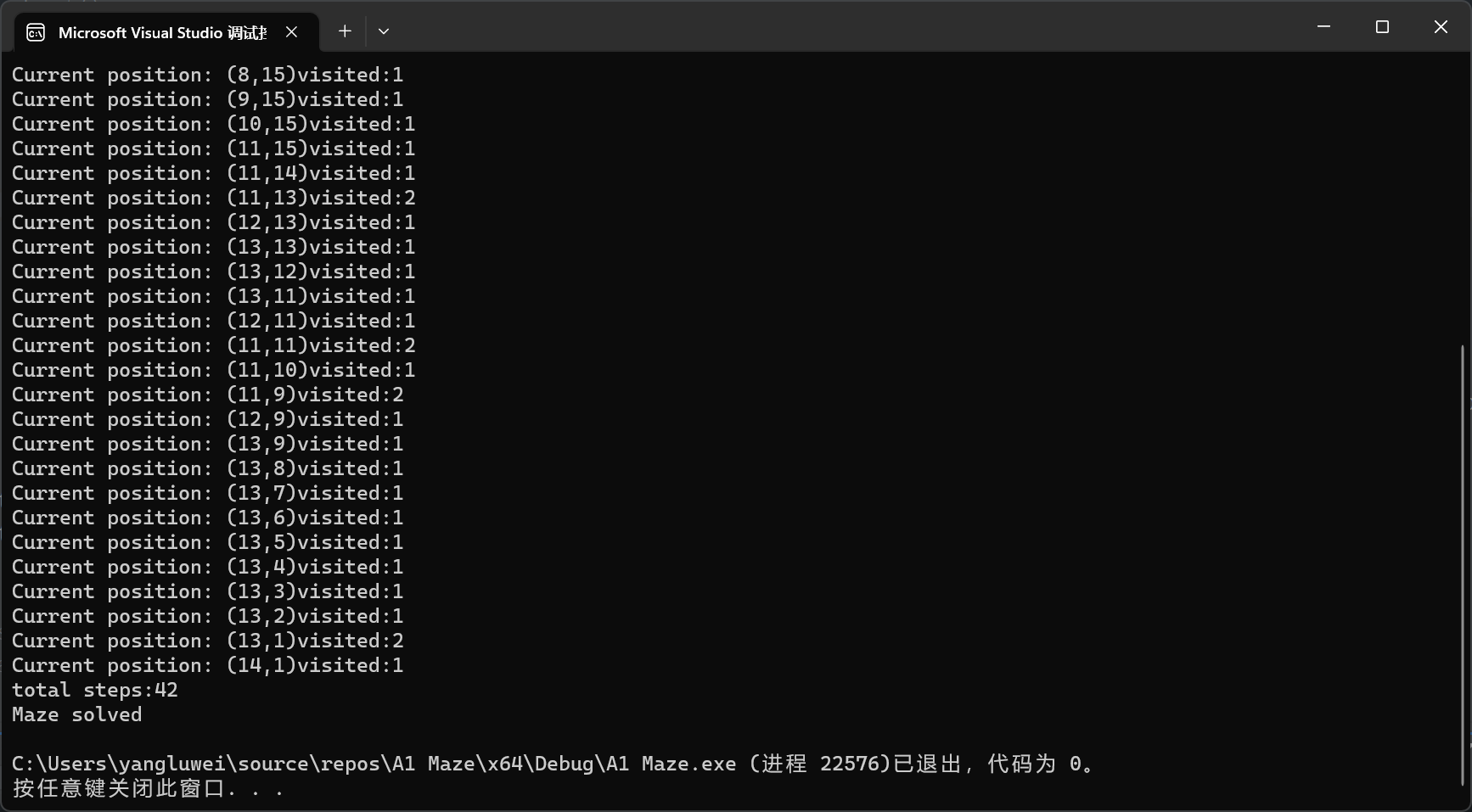
}

\*/

1. **剩余两个文件的删除重合路径的前后对比**

Maze2.txt





Maze3.txt

