C++方向编程题答案

答案说明：

> 大家如果对本次题目或者答案有问题，可以联系下方的出题老师答疑。
>
> **出题老师：**
>
> 选择题：张文超　qq：3627274478
>
> 代码题：时亮益　qq：569334855

第七周

day42

> 题目ID：845 解读密码
>
> 链接：https://www.nowcoder.com/questionTerminal/16fa68271ee5448cafd504bb4a64b482
>
> **【题目解析】**
>
> 题目意思：给定字符串中包含其他符合一级数字，将字符串中数字解析出来。
>
> **【解题思路】**
>
> 1. 题目明确说明有多行：采用getline进行循环输入
> 2. 对输入的字符串进行遍历，如果是数字字符，将其输出即可

```
#include <iostream>
using namespace std;

#include <string>

int main()
{
 string s;
 while(getline(cin, s))
 {
     string ret;
     for(auto e : s)
     {
         if(e >= '0' && e <= '9')
             ret += e;
     }

     cout<<ret<<endl;
 }
 return 0;
}
```

> 题目ID：791 走迷宫
>
> 链接：https://www.nowcoder.com/questionTerminal/6276dbbda7094978b0e9ebb183ba37b9
>
> **【题目解析】**

采用二维数组保存地图数据，如下图所示，'#'代表墙，'.'代表通路，在每个路口时，可以向上下左右4个方向移动(除过边界)，当一条路径不通时，需要回溯到上一步，回溯到上一个路口后再向其他方向探索，直到走到出口，一条路径找到，求解出步数，再回退找其他路径，求解出最短的路径。以上方式是采用深度优先方式遍历，但是深度优先方式找到一条路径后，该路径不一定是最短路径，需要将所有路径找到后比较，才能确定出最终的最短路径。

该题最佳的解决方式是：采用广度优先遍历，即一层一层的往下移动(可参考图的广度优先遍历思想)，采用广度优先方式遍历，当找到出口时，则为最短路径。大概方式：在一个位置时，计算该位置上、下、左、右四个方向，如果该位置在出口即找到，否则检测该位置是否有效且为通路，如果是将其保存到队列中，再继续该过程，直到找到出口，具体参考解题思路。

```
#.########
#........#
########.#
#........#
#.########
#........#
########.#
#........#
#.######.#
########.#
```

【解题思路】

1. 采用一个二维数组，不断的接受迷宫地图(因为有多个地图)，获取到迷宫地图后，采用广度优先方式走迷宫，找到的第一条路径一定是最短的路径，但是深度优先则不一定。
2. 结构设定：

pos：x，y表示当前所在位置，level所经过步数

dir: 表示当前位置的上、下、左、右四个方向

queue: 广度优先遍历，需要用到队列，保存所经路径

visit：对走过的路径进行标记

start和end表示入口和出口

3. 采用广度优先方式走迷宫：将start入队列，对该位置进行标记，只要队列不为空，继续以下步骤，直到到达出口：
    - 从队列中取出当前位置cur，计算该位置的上、下、坐、右四个方向，计算要走的下一步
    - 如果next在出口的位置，已经找到返回所走步数level
    - 否则：如果该位置有效(坐标在地图中)不是墙(该位置字符为'.')，并且没有走过(visit标记为false)时，将该位置入队列，再继续其他几个方向

```
#include <iostream>
using namespace std;

#include <vector>
#include <string>
#include <queue>


struct Position
{
 int x;
```

```
 int y;
 int level;   // 表示从入口到达(x,y)坐标所走的步数
};

// 使用广度优先遍历来走迷宫，找到最短的路径
int bfs(vector<string>& map, int m, int n)
{
 // 方向数组：下  右  左  上
 int direct[4][2] = {{1, 0},{0, 1},{0, -1},{-1, 0}};
 queue<Position> q;
 Position start{0,1,0}, out{9,8,0};
 q.push(start);

 vector<vector<bool>> flag(m, vector<bool>(n, false));
 while(!q.empty())
 {
     Position cur = q.front();
     q.pop();

     // 遍历该位置,即将该位置进行标记
     flag[cur.x][cur.y] = true;

     // 如果cur已经在出口的位置，说明最短路径已经找到了
     if(cur.x == out.x && cur.y == out.y)
     {
         return cur.level;
     }

     // 向cur四个方向继续进行遍历
     Position next;
     for(int i = 0; i < 4; ++i)
     {
         next.x = cur.x + direct[i][0];
         next.y = cur.y + direct[i][1];

         // 检测位置的坐标是否合法
         // 该位置是否为通路
         // 该位置如果没有遍历过
         if(next.x >= 0 && next.x < m && next.y >= 0 && next.y < n &&
map[next.x][next.y] == '.' && !flag[next.x][next.y])
         {
             next.level = cur.level+1;
             q.push(next);
         }
     }
 }

 return 0;
}

int main()
{
 vector<string> map(10);   // 地图

 // 循环输入处理每组测试用例
 int i = 0;
 while(cin>>map[i++])
 {
```

```
        if(10 == i)
        {
            // 已经将地图中的数据接收完成
            cout<<bfs(map, 10, 10)<<endl;
            i = 0;
        }
    }
    return 0;
}
```