**Assignment 6 Report**

**1.Problem：**

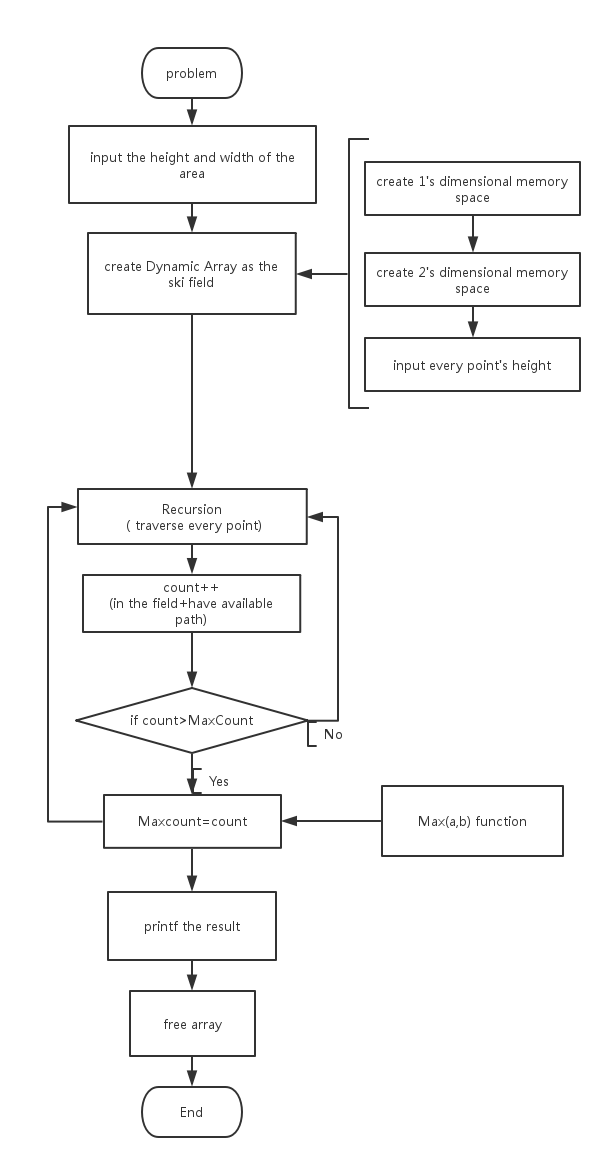
Suppose the skiing field is a rectangular area which is described by a two-dimensional array. Each element of the array denotes the height of the point. Michael can start from any point, and he can slide to one of the four adjacent points if its height is lower than current height. Michael cannot leave ski field when skiing.

e.g.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 16 | **17** | **18** | **19** | **6** |
| 15 | **24** | **25** | **20** | **7** |
| 14 | **23** | **22** | **21** | **8** |
| 13 | **12** | **11** | **10** | **9** |

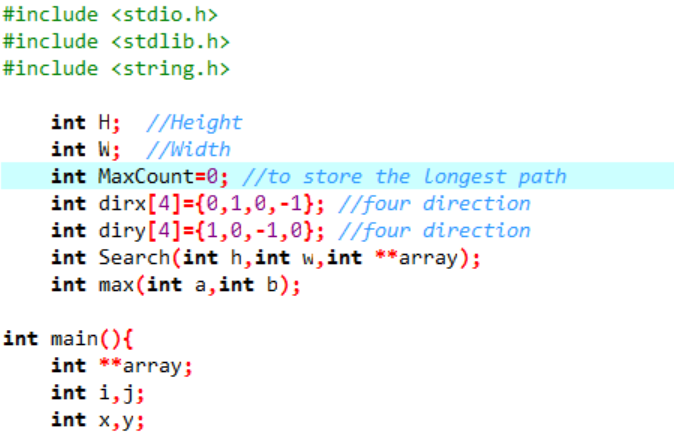
Output: 25

**2.Main Idea:**

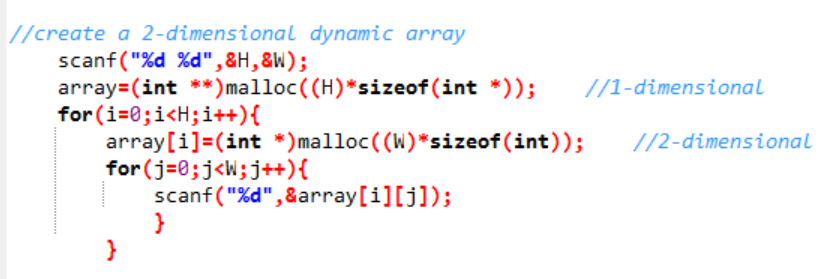
****

**3.Details about the program:**

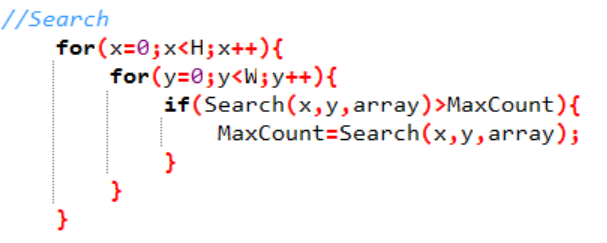
1. **Define:**



1. Use H and W which are entered by user to store the height and width of the ski field.
2. Use MaxCount to compare with the value of the current path and always store the value of longest path.
3. Use dirx[4] and diry[4] to indicate four direction.
4. **Create a 2-dimensional dynamic array:**

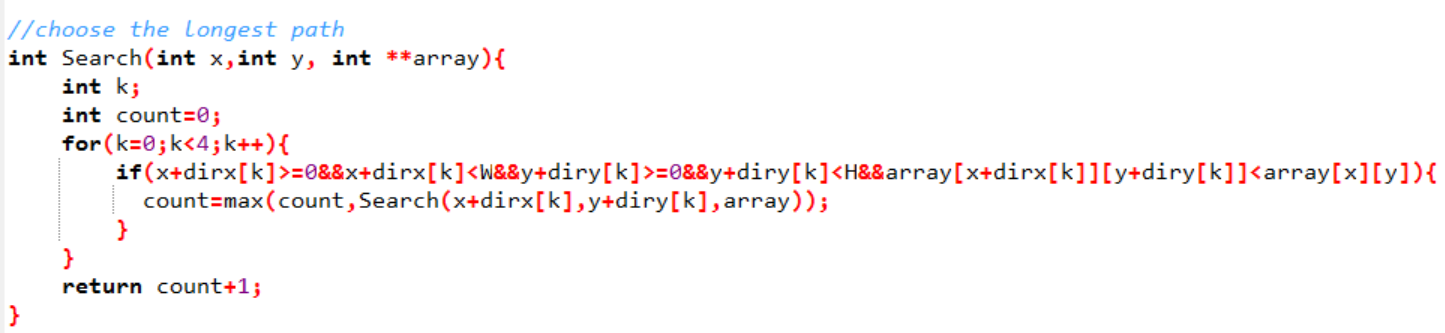


1. Because in this program, the height and width of the ski field are unknown before user input value, it would be better to use dynamic array.
2. Use malloc( ) to give 1-dimensional memory space.
3. **Search:**



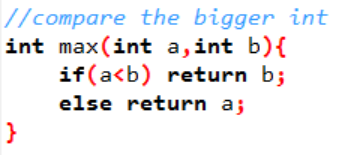
Because the program need to search the longest path without knowing the start of the way, it’s necessary to test every point’s way and compare it to the value of MaxCount.

1. **Recursion:**



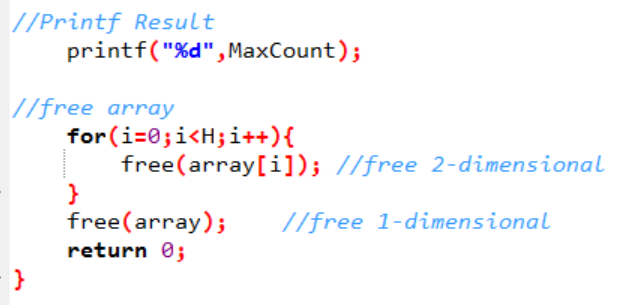
Use breadth-first search. When move to a point, check if it go out of the boarder or if it is larger than all the four number near it. If so, return count+1 and go back to find out the value of count.

1. **Compare :**



A function two compare which number is bigger than the other and this function can improve the program.

1. **Printf result and free dynamic array:**



1. Output the result.
2. As a dynamic array, it’s quite important to free array before the end of execution.

**4.Result：**

