**Розробка алгоритмiв**

**Лабораторна робота №1**

Варіант 2

**Перше завдання**

using System;

using System.Collections.Generic;

using System.Text;

namespace task1

{

public class MyStack<T>

{

private Stack<T> stack = new Stack<T>();

public void Push(T item)

{

stack.Push(item);

}

public T Pop()

{

if (stack.Count != 0)

{

return stack.Pop();

}

else

{

throw new Exception("The stack is empty");

}

}

public int Size()

{

return stack.Count;

}

}

}

**Друге завдання**

using System;

namespace task1

{

class Program

{

static char[,] maze = {

{'#', 'S', '#', '.', '.', '.', '.', '.'},

{'#', '.', '#', '.', '#', '#', '.', '#'},

{'#', '.', '.', '.', '#', '.', '.', '#'},

{'#', '#', '#', '#', '.', '.', '#', '#'},

{'#', '.', '.', '.', '.', '#', 'F', '#'},

{'#', '.', '#', '.', '.', '#', '.', '#'},

{'#', '.', '#', '#', '#', '#', '.', '#'},

{'#', '.', '.', '.', '.', '.', '.', '#'},

};

static int rows = maze.GetLength(0);

static int cols = maze.GetLength(1);

static bool IsValidPosition(int row, int col)

{

return row >= 0 && row < rows && col >= 0 && col < cols;

}

static bool FindExit(int row, int col)

{

MyStack<int[]> stack = new MyStack<int[]>();

stack.Push(new int[] { row - 1, col });

stack.Push(new int[] { row, col + 1 });

stack.Push(new int[] { row + 1, col });

stack.Push(new int[] { row, col - 1 });

while (stack.Size() > 0)

{

var currentPosition = stack.Pop();

row = currentPosition[0];

col = currentPosition[1];

if (!IsValidPosition(row, col) || maze[row, col] == '#' || maze[row, col] == 'x')

{

continue;

}

if (maze[row, col] == 'F')

{

return true;

}

if (maze[row, col] == 'S')

{

maze[row, col] = 'S';

}

else

{

maze[row, col] = 'x';

}

stack.Push(new int[] { row - 1, col });

stack.Push(new int[] { row, col + 1 });

stack.Push(new int[] { row + 1, col });

stack.Push(new int[] { row, col - 1 });

}

return false;

}

static void PrintMaze()

{

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

{

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

{

Console.Write(maze[i, j] + " ");

}

Console.WriteLine();

}

}

static void Main()

{

int startRow = 0;

int startCol = 0;

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

{

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

{

if (maze[i, j] == 'S')

{

startRow = i;

startCol = j;

break;

}

}

}

if (FindExit(startRow, startCol))

{

PrintMaze();

}

else

{

Console.WriteLine("The exit from the maze is not found");

}

}

}

}

**Третє завдання**

1. static bool IsValidPosition(int row, int col)

Початок

return

row >= 0 && row < rows && col >= 0 && col < cols

1. static bool FindExit(int row, int col)

\_

**+**

Початок

MyStack<int[]> stack = new MyStack<int[]>();

stack.Push(new int[] { row - 1, col });

stack.Push(new int[] { row, col + 1 });

stack.Push(new int[] { row + 1, col });

stack.Push(new int[] { row, col - 1 });

var currentPosition = stack.Pop();

row = currentPosition[0];

col = currentPosition[1];

!IsValidPosition(row, col) || maze[row, col] == '#' || maze[row, col] == 'x'

stack.Size() > 0

continue;

\_

**+**

\_

**+**

maze[row, col] == 'F'

Return

true

maze[row, col] == 'S'

maze[row, col] = 'S';

maze[row, col] = 'x';

stack.Push(new int[] { row - 1, col });

stack.Push(new int[] { row, col + 1 });

stack.Push(new int[] { row + 1, col });

stack.Push(new int[] { row, col - 1 });

Return false

1. static void PrintMaze()

int i = 0; i < rows; i++

int j = 0; j < cols; j++

Console.Write(maze[i, j] + " ");

Console.WriteLine();

Початок

Кінець

1. static void Main()

\_

**+**

\_

**+**

Початок

int startRow = 0;

int startCol = 0;

int i = 0; i < rows; i++

int j = 0; j < cols; j++

Кінець

maze[i, j] == 'S'

startRow = i;

startCol = j;

break;

FindExit(startRow, startCol)

PrintMaze();

Console.WriteLine("The exit from the maze is not found");

Розглянемо, як реалізований алгоритм відповідає властивостям алгоритму.

Дискретність: Алгоритм представляє процес вирішення задачі як послідовне виконання простих кроків. Кожен крок включає перевірку поточної позиції у лабіринті, оновлення стану лабіринту та додавання нових можливих кроків до стеку. Кожен крок виконується за скінченний проміжок часу.

Детермінованість: Кожен крок алгоритму визначений точно. Дії, які необхідно здійснити, чітко та недвозначно визначені для кожного можливого випадку. Оновлення стану лабіринту та додавання нових кроків відбувається з урахуванням умов (наприклад, перевірка чи не виходить за межі лабіринту, чи не є поточна позиція стіною).

Масовість: Алгоритм забезпечує розв'язання будь-якої задачі пошуку виходу з лабіринту за будь-якими вхідними даними, які представляють собою лабіринт. Він може працювати з лабіринтами різних розмірів та конфігурацій.

Скінченність (результативність): Алгоритм завжди завершується після виконання скінченної кількості кроків. Процедура завершується, якщо знайдено вихід з лабіринту ('F') або якщо всі можливі кроки перевірені, і вихід не знайдено. Це дозволяє алгоритму завершити роботу за обмежений час.

Правильність: Результати виконання алгоритму відповідають умові постановки задачі. Алгоритм оновлює стан лабіринту та перевіряє, чи досягнуто вихід ('F'). Якщо вихід знайдено, то програма виводить лабіринт з шляхом до виходу, в іншому випадку повідомляє, що вихід не знайдено.

Таким чином, реалізація алгоритму відповідає властивостям алгоритму, які були описані в першій секції лабораторної роботи.