****

# **Chapter 10. Recursion**

**Prof:Muzafer Shala**

**Ass:Laberion Zebica Student: Adhurim Haziri**

**Kampusi:Ferizaj**

1. **Write a program to simulate n nested loops from 1 to n.**

using System;

namespace detyra1

{

class Program

{

static void Loops(int[] arr, int index)

{

if (index >= arr.Length)

{

foreach (int element in arr) Console.Write("{0} ", element);

Console.WriteLine();

}

else

for (int i = 1; i <= arr.Length; i++)

{

arr[index] = i;

Loops(arr, index + 1);

}

}

static void Main(string[] args)

{

Console.Write("Enter N: ");

int n = Int32.Parse(Console.ReadLine());

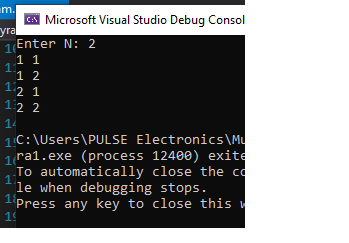
int[] arr = new int[n];

Loops(arr, 0);

}

}

}

****

**2.  Write a program to generate all variations with duplicates of n elements class k. Sample input:**

|  |
| --- |
| **n = 3**  **k = 2** |

**Sample output:**

|  |
| --- |
| **(1 1), (1 2), (1 3), (2 1), (2 2), (2 3), (3 1), (3 2), (3 3)** |

**Think about and implement an iterative algorithm for the same task.**

using System;

namespace detyra2

{

class Program

{

static void GetCombinations(int[] arr, int index, int start, int end)

{

if (index >= arr.Length)

{

Console.Write("(");

for (int i = 0; i < arr.Length; i++)

if (i < arr.Length - 1) Console.Write("{0} ", arr[i]);

else Console.Write(arr[i]);

Console.Write("), ");

}

else

for (int i = start; i <= end; i++)

{

arr[index] = i;

GetCombinations(arr, index + 1, i, end);

}

}

static void Main(string[] args)

{

Console.Write("Enter N: ");

int n = Int32.Parse(Console.ReadLine());

Console.Write("Enter K: ");

int k = Int32.Parse(Console.ReadLine());

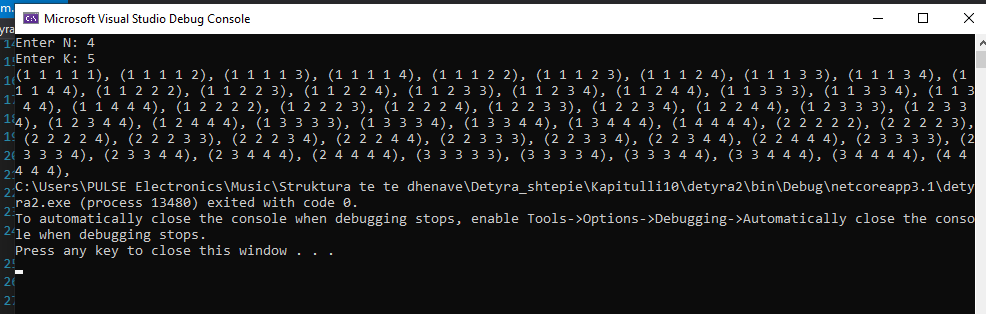
int[] arr = new int[k];

GetCombinations(arr, 0, 1, n);

}

}

}

****

**3.    Write a program to generate and print all combinations with duplicates of k elements from a set with n elements. Sample input:**

|  |
| --- |
| **n = 3**  **k = 2** |

**Sample output:**

|  |
| --- |
| **(1 1), (1 2), (1 3), (2 2), (2 3), (3 3)** |

**Think about and implement an iterative algorithm for the same task.**

using System;

namespace detyra3

{

class Program

{

static void GetCombinations(int[] arr, int index, int start, int end)

{

if (index >= arr.Length)

{

Console.Write("(");

for (int i = 0; i < arr.Length; i++)

if (i < arr.Length - 1) Console.Write("{0} ", arr[i]);

else Console.Write(arr[i]);

Console.Write("), ");

}

else

for (int i = start; i <= end; i++)

{

arr[index] = i;

GetCombinations(arr, index + 1, i, end);

}

}

static void Main(string[] args)

{

Console.Write("Enter N: ");

int n = Int32.Parse(Console.ReadLine());

Console.Write("Enter K: ");

int k = Int32.Parse(Console.ReadLine());

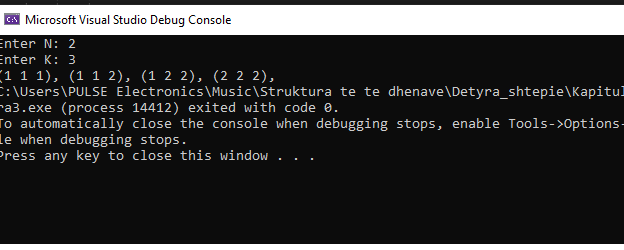
int[] arr = new int[k];

GetCombinations(arr, 0, 1, n);

}

}

}

****

**4.    You are given a set of strings. Write a recursive program, which generates all subsets, consisting exactly k strings chosen among the elements of this set. Sample input:**

|  |
| --- |
| **strings = {'test', 'rock', 'fun'}**  **k = 2** |

**Sample output:**

|  |
| --- |
| **(test rock), (test fun), (rock fun)** |

**Think about and implement an iterative algorithm as well.**

using System;

namespace detyra4

{

class Program

{

static string[] wordsArr;

static void FindSubsets(int[] arr, int index, int start, int end)

{

if (index >= arr.Length)

{

Console.Write("(");

for (int i = 0; i < arr.Length; i++)

{

Console.Write("{0}", wordsArr[arr[i]]);

if (i != arr.Length - 1) Console.Write(" ");

}

Console.Write("), ");

}

else

for (int i = start; i < end; i++)

{

arr[index] = i;

FindSubsets(arr, index + 1, i + 1, end);

}

}

static void Main(string[] args)

{

Console.Write("Enter array length: ");

int length = Int32.Parse(Console.ReadLine());

wordsArr = new string[length];

Console.WriteLine();

for (int i = 0; i < wordsArr.Length; i++)

{

Console.Write("Enter {0} word: ", i + 1);

wordsArr[i] = Console.ReadLine();

}

Console.Write("\nEnter K: ");

int k = Int32.Parse(Console.ReadLine());

int[] arr = new int[k];

Console.WriteLine();

FindSubsets(arr, 0, 0, length);

Console.ReadLine();

}

}

}

**5.    Write a recursive program, which prints all subsets of a given set of N words. Example input:**

|  |
| --- |
| **words = {'test', 'rock', 'fun'}** |

**Example output:**

|  |
| --- |
| **(), (test), (rock), (fun), (test rock), (test fun),**  **(rock fun), (test rock fun)** |

**Think about and implement an iterative algorithm for the same task.**

using System;

namespace detyra5

{

class Program

{

static string[] strings, str;

static int length;

static void cycle(int iter, int index, int k)

{

if (iter == k)

{

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

Console.WriteLine("({0})", str[i]);

return;

}

for (int i = index; i < strings.Length; i++)

{

str[iter] = strings[i];

cycle(iter + 1, i + 1, k);

}

}

static void Main(string[] args)

{

Console.Write("Enter words length: ");

length = Int32.Parse(Console.ReadLine());

strings = new string[length];

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

{

Console.Write("Enter {0} word: ", i + 1);

strings[i] = Console.ReadLine();

}

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

{

str = new string[length];

cycle(0, 0, i);

}

}

}

}

**6.    Implement the merge-sort algorithm recursively. In it the initial array is divided into two equal in size parts, which are sorted (recursively via merge-sort) and after that the two sorted parts are merged in order to get the whole sorted array.**

using System;

namespace detyra6

{

class Program

{

static public void DoMerge(int[] numbers, int left, int mid, int right)

{

int[] temp = new int[25];

int i, left\_end, num\_elements, tmp\_pos;

left\_end = (mid - 1);

tmp\_pos = left;

num\_elements = (right - left + 1);

while ((left <= left\_end) && (mid <= right))

{

if (numbers[left] <= numbers[mid])

temp[tmp\_pos++] = numbers[left++];

else

temp[tmp\_pos++] = numbers[mid++];

}

while (left <= left\_end)

temp[tmp\_pos++] = numbers[left++];

while (mid <= right)

temp[tmp\_pos++] = numbers[mid++];

for (i = 0; i < num\_elements; i++)

{

numbers[right] = temp[right];

right--;

}

}

static public void MergeSort\_Recursive(int[] numbers, int left, int right)

{

int mid;

if (right > left)

{

mid = (right + left) / 2;

MergeSort\_Recursive(numbers, left, mid);

MergeSort\_Recursive(numbers, (mid + 1), right);

DoMerge(numbers, left, (mid + 1), right);

}

}

static void Main(string[] args)

{

Console.Write("Enter array length: ");

int length = Int32.Parse(Console.ReadLine());

int[] arr = new int[length];

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

{

Console.Write("Enter {0} element: ", i + 1);

arr[i] = Int32.Parse(Console.ReadLine());

}

Console.WriteLine("\nSorted");

MergeSort\_Recursive(arr, 0, length - 1);

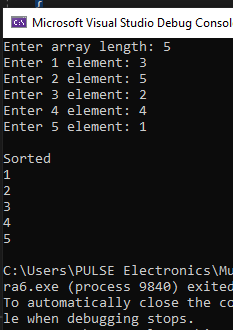
for (int i = 0; i < length; i++)

Console.WriteLine(arr[i]);

}

}

}

**7.    Write a recursive program, which generates and prints all permutations of the numbers 1, 2, …, n, for a given integer n. Example input:**

|  |
| --- |
| **n = 3** |

**Example output:**

|  |
| --- |
| **(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)** |

**Try to find an iterative solution for generating permutations.**

using System;

namespace detyra7

{

class Program

{

static void FindCombinations(int[] arr, int index)

{

if (index == arr.Length)

{

Console.Write("{");

for (int i = 0; i < arr.Length; i++)

if (i < arr.Length - 1) Console.Write("{0} ", arr[i]);

else Console.Write(arr[i]);

Console.Write("}, ");

}

else

{

for (int i = index; i < arr.Length; i++)

{

int temp = arr[i];

arr[i] = arr[index];

arr[index] = temp;

FindCombinations(arr, index + 1);

temp = arr[i];

arr[i] = arr[index];

arr[index] = temp;

}

}

}

static void Main(string[] args)

{

Console.Write("Enter N: ");

int n = Int32.Parse(Console.ReadLine());

int[] arr = new int[n];

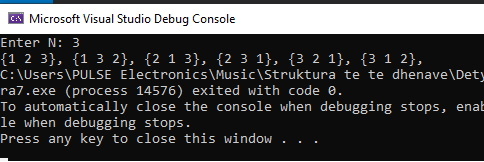
for (int i = 0; i < arr.Length; i++) arr[i] = i + 1;

FindCombinations(arr, 0);

}

}

}



**8.    You are given an array of integers and a number N. Write a recursive program that finds all subsets of numbers in the array, which have a sum N. For example, if we have the array {2, 3, 1, -1} and N=4, we can obtain N=4 as a sum in the following two ways: 4=2+3-1; 4=3+1.**

using System;

namespace detyra8

{

class Program

{

public static bool isSubsetSum(int[] arr, int n, int sum)

{

bool[,] subset = new bool[sum + 1, n + 1];

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

subset[0, i] = true;

for (int i = 1; i <= sum; i++)

subset[i, 0] = false;

for (int i = 1; i <= sum; i++)

for (int j = 1; j <= n; j++)

{

subset[i, j] = subset[i, j - 1];

if (i >= arr[j - 1])

subset[i, j] = subset[i, j] || subset[i - arr[j - 1], j - 1];

}

return subset[sum, n];

}

static void Main(string[] args)

{

Console.Write("Enter arr length: ");

int length = Int32.Parse(Console.ReadLine());

int[] arr = new int[length];

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

{

Console.Write("Enter {0} element: ", i + 1);

arr[i] = Int32.Parse(Console.ReadLine());

}

Console.Write("Enter sum: ");

int sum = Int32.Parse(Console.ReadLine());

if (isSubsetSum(arr, arr.Length, sum) == true)

Console.WriteLine("Found a subset with given sum");

else

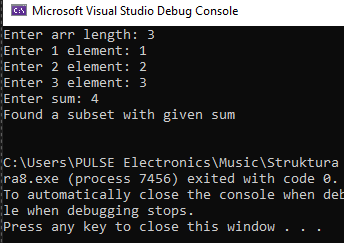
Console.WriteLine("No subset with given sum");

Console.ReadLine();

}

}

}

****

**9.    You are given an array of positive integers. Write a program that checks whether there is one or more numbers in the array (subset), whose sum is equal to S. Can you solve the task efficiently for large arrays?**

using System;

namespace detyra9

{

class Program

{

static void Main(string[] args)

{

Console.Write("Enter array length: ");

int length = Int32.Parse(Console.ReadLine());

int[] arr = new int[length];

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

{

Console.Write("Enter {0} element: ", i + 1);

arr[i] = Int32.Parse(Console.ReadLine());

}

Console.Write("Enter sum: ");

int sum = Int32.Parse(Console.ReadLine());

for (int i = 0; i < arr.Length; i++)

{

int first = arr[i];

for (int j = i + 1; j < arr.Length; j++)

{

int second = arr[j];

if ((first + second) == sum)

Console.WriteLine("({0}, {1}) ", first, second);

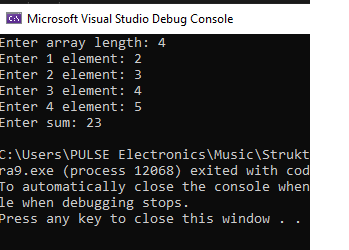
}

}

}

}

}

****

**10.   You are given a matrix with passable and impassable cells. Write a recursive program that finds all paths between two cells in the matrix.**

using System;

namespace detyra10

{

class Program

{

static int numberOfPaths(int m, int n)

{

int[,] count = new int[m, n];

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

count[i, 0] = 1;

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

count[0, j] = 1;

for (int i = 1; i < m; i++)

for (int j = 1; j < n; j++)

count[i, j] = count[i - 1, j] + count[i, j - 1];

return count[m - 1, n - 1];

}

static void Main(string[] args)

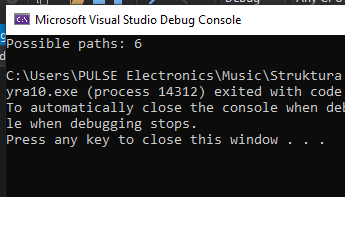
{

Console.WriteLine("Possible paths: {0}", numberOfPaths(3, 3));

}

}

}

****

**11.   Implement the algorithm BFS (breadth-first search) for finding the shortest path in a labyrinth.**

using System;

using System.Collections.Generic;

namespace detyra11

{

class Program

{

static int ROW = 9;

static int COL = 10;

public class Point

{

public int x;

public int y;

public Point(int x, int y)

{

this.x = x;

this.y = y;

}

};

public class queueNode

{

public Point pt;

public int dist;

public queueNode(Point pt, int dist)

{

this.pt = pt;

this.dist = dist;

}

};

static bool isValid(int row, int col)

{

return (row >= 0) && (row < ROW) &&

(col >= 0) && (col < COL);

}

static int[] rowNum = { -1, 0, 0, 1 };

static int[] colNum = { 0, -1, 1, 0 };

static int BFS(int[,] mat, Point src,

Point dest)

{

if (mat[src.x, src.y] != 1 ||

mat[dest.x, dest.y] != 1)

return -1;

bool[,] visited = new bool[ROW, COL];

visited[src.x, src.y] = true;

Queue<queueNode> q = new Queue<queueNode>();

queueNode s = new queueNode(src, 0);

q.Enqueue(s);

while (q.Count != 0)

{

queueNode curr = q.Peek();

Point pt = curr.pt;

if (pt.x == dest.x && pt.y == dest.y)

return curr.dist;

q.Dequeue();

for (int i = 0; i < 4; i++)

{

int row = pt.x + rowNum[i];

int col = pt.y + colNum[i];

if (isValid(row, col) &&

mat[row, col] == 1 &&

!visited[row, col])

{

visited[row, col] = true;

queueNode Adjcell = new queueNode

(new Point(row, col),

curr.dist + 1);

q.Enqueue(Adjcell);

}

}

}

return -1;

}

public static void Main(String[] args)

{

int[,] mat = {{ 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },

{ 1, 0, 1, 0, 1, 1, 1, 0, 1, 1 },

{ 1, 1, 1, 0, 1, 1, 0, 1, 0, 1 },

{ 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 },

{ 1, 1, 1, 0, 1, 1, 1, 0, 1, 0 },

{ 1, 0, 1, 1, 1, 1, 0, 1, 0, 0 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },

{ 1, 1, 0, 0, 0, 0, 1, 0, 0, 1 }};

Point source = new Point(0, 0);

Point dest = new Point(3, 4);

int dist = BFS(mat, source, dest);

if (dist != -1)

Console.WriteLine("Shortest Path is " + dist);

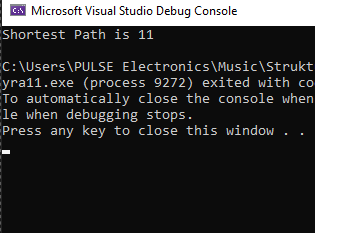
else

Console.WriteLine("Shortest Path doesn't exist");

}

}

}

****

**12.   Modify the previous program to check whether a path exists between two cells without finding all possible paths. Test the program with a matrix 100x100 filled only with passable cells.**

using System;

namespace detyra12

{

class Program

{

static void isPath(int[,] matrix, int n)

{

bool[,] visited = new bool[n, n];

bool flag = false;

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

{

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

{

if (matrix[i, j] == 1 &&

!visited[i, j])

if (isPath(matrix, i, j,

visited))

{

flag = true;

break;

}

}

}

if (flag)

Console.WriteLine("YES");

else

Console.WriteLine("NO");

}

public static bool isSafe(int i, int j,

int[,] matrix)

{

if (i >= 0 && i < matrix.GetLength(0) &&

j >= 0 && j < matrix.GetLength(1))

return true;

return false;

}

public static bool isPath(int[,] matrix, int i,

int j, bool[,] visited)

{

if (isSafe(i, j, matrix) &&

matrix[i, j] != 0 &&

!visited[i, j])

{

visited[i, j] = true;

if (matrix[i, j] == 2)

return true;

bool up = isPath(matrix, i - 1,

j, visited);

if (up)

return true;

bool left = isPath(matrix, i,

j - 1, visited);

if (left)

return true;

bool down = isPath(matrix, i + 1,

j, visited);

if (down)

return true;

bool right = isPath(matrix, i, j + 1,

visited);

if (right)

return true;

}

return false;

}

static void Main()

{

int[,] matrix = { { 0, 3, 0, 1 },

{ 3, 0, 3, 3 },

{ 2, 3, 3, 3 },

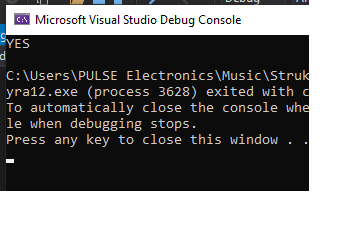
{ 0, 3, 3, 3 } };

isPath(matrix, 4);

}

}

}

****

**13.   You are given a matrix with passable and impassable cells. Write a program that finds the largest area of neighboring passable cells.**

using System;

namespace detyra13

{

class Program

{

static int N = 4;

static int MaximumPath(int[,] Mat)

{

int result = 0;

int[,] dp = new int[N, N + 2];

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

dp[0, i + 1] = Mat[0, i];

for (int i = 1; i < N; i++)

for (int j = 1; j <= N; j++)

dp[i, j] = Math.Max(dp[i - 1, j - 1],

Math.Max(dp[i - 1, j],

dp[i - 1, j + 1])) +

Mat[i, j - 1];

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

result = Math.Max(result, dp[N - 1, i]);

return result;

}

public static void Main()

{

int[,] Mat = { { 4, 2, 3, 4 },

{ 2, 9, 1, 10 },

{ 15, 1, 3, 0 },

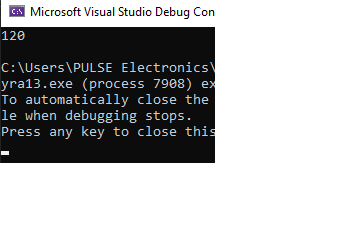
{ 16, 92, 41, 44 } };

Console.WriteLine(MaximumPath(Mat));

}

}

}



**14.   Write a recursive program that traverses the whole hard disk C:\ recursively and prints all folders and files.**

using System;

using System.IO;

class Program

{

static void Main()

{

string[] files = Directory.GetFiles("C:\\Users\\PulseElectronics\\",

"\*.\*",

SearchOption.AllDirectories);

foreach (string file in files)

{

Console.WriteLine(file);

}

}

}