Table

Description automatically generated

StackUtils פעולות בסיסיות.

בסיום הפעולות המחסנית תשאר כמו בהתחלה.

לממש את הרוב ב- 2 צורות: איטרטיבי, רקורסיבי

1. מערך למחסנית

public static Stack<T> CreateStackFromArray<T>(T[] arr)

1. שפיכת מחסנית אחת לשנייה – SpilledOn ומחזירה את מספר האיברים שהועברו. בסיום מחסנית from ריקה

public static int SpilledOn<T>(Stack<T> to, Stack<T> from)

1. שיכפול מחסנית – Clone. בסיום מחסנית s נשארת כמו שהיתה.

public static Stack<T> Clone<T>(Stack<T> s)

1. גודל המחסנית (כמות האיברים) – Size, SizeRec – בסיום הפעולה יש לשמור על המחסנית המקורית

public static int GetSize<T>(Stack<T> s)

public static int GetSizeRec<T>(Stack<T> s)

1. סכום האיברים (בסיום הפעולה המחסנית תשאר כמו בהתחלה), איטרטיבי, רקורסיבי – Sum – בסיום הפעולה יש לשמור על המחסנית המקורית

public static int Sum(Stack<int> s)

public static int SumRec(Stack<int> s)

1. הוצאת האיבר האחרון

public static T RemoveButtom<T>(Stack<T> stk)

1. הוצאת האיבר ה-n (איבר מס' 1 הוא האיבר בראש המחסנית)

public static T RemovePos<T>(Stack<T> st, int n)

1. הכנסת איבר לסוף המחסנית

public static void AddValueToStackButtom<T>(Stack<T> stk, T e)

1. הוספת איבר במקום ה- n

public static void InsertAtPos<T>(Stack<T> st, T e, int n)

1. IsSorted – איטרטיבי, רקורסיבי

public static bool IsSorted(Stack<int> s)

1. IsExist: מחזירה אמת אם האיבר נמצא במחסנית

public static bool IsExist(Stack<int> s, int val))

public static bool IsExistRec(Stack<int> s, int val)

1. בניית מחסנית ממויינת

public static Stack<int> BuildSort(int[] arr)

public static Stack<int> BuildSort(Node<int> ls)

1. הזזה מעגלית 1 (ראשון אחרון): האיבר שבראש המחסנית יהיה האחרון

public static void MoveTop2Buttom<T>(Stack<T> stk)

1. הזזה מעגלית 2 (אחרון ראשון): האיבר שהאחרון המחסנית יהיה בראש

public static void MoveButtom2Top<T>(Stack<T> stk)

1. מיון מחסנית (ללא שימוש במערכים או שרשרת חוליות)

public static void SortInput(Stack<int> s)

1. מחסניות דומות. שתי מחסניות נקראות דומות אם הן מכילות את אותן האיברים, לאו דוקא באותו הסדר. מה הוא סדר גודל של הפעולה?
2. הוצאת האיבר המקסימלי/מינימלי (בסיום הפעולה המחסנית תשאר כמו בהתחלה ללא האיבר), איטרטיבי, רקורסיבי (צריך פעולת מעטפת) (אתגר: רקורסיבי ללא פעולת מעטפת)

public static int RemoveMaxRec(Stack<int> stk)

namespace Unit4

{

public class Stack<T>

{

private Node<T> data;

public Stack()

{

}

public bool IsEmpty()

{

}

public void Push(T x)

{

}

public T Pop()

{

}

public T Top()

{

}

public override string ToString()

{

string text = "[";

return text + "]";

}

}

}

using System;

namespace Unit4

{

class StackUtils

{

public static Stack<T> CreateStackFromArray<T>(T[] arr)

{

Stack<int> s = new Stack<int>();

return s;

}

public static void SpilledOn<T>(Stack<T> to, Stack<T> from)

{

}

public static Stack<T> Clone<T>(Stack<T> s)

{

Stack<T> t = new Stack<T>();

return t;

}

public static Stack<T> CloneRec<T>(Stack<T> s)

{

}

public static Stack<T> ReverseStack<T>(Stack<T> s)

{

}

public static int GetSize<T>(Stack<T> s)

{

int count = 0;

return count;

}

public static int GetSizeRec<T>(Stack<T> s)

{

return count;

}

public static int Sum(Stack<int> s)

{

int sum = 0;

return sum;

}

public static int SumRec(Stack<int> s)

{

return sum;

}

public static bool IsSorted(Stack<int> s)

{

}

public static bool IsSortedRec(Stack<int> s)

{

return res;

}

public static bool IsExist(Stack<int> s, int val)

{

bool found = false;

return found;

}

public static bool IsExistRec(Stack<int> s, int val)

{

bool found = false;

return found;

}

public static void Sort(Stack<int> s)

{

}

public static Stack<int> BuildSort(int[] arr)

{

Stack<int> numbers = new Stack<int>();

return numbers;

}

public static Stack<int> BuildSort(Node<int> ls)

{

Stack<int> numbers = new Stack<int>();

return numbers;

}

public static void AddValueToStackButtom<T>(Stack<T> stk, T e)

{

}

public static T RemoveButtom<T>(Stack<T> stk)

{

}

public static void InsertButtom<T>(Stack<T> stk, T elm)

{

}

public static void MoveButtom2Top<T>(Stack<T> stk)

{

}

public static void MoveTop2Buttom<T>(Stack<T> stk)

{

}

public static void InsertAtPos<T>(Stack<T> st, T e, int n)

{

}

public static int RemoveMax(Stack<int> stk)

{

}

public static int RemoveMaxRec(Stack<int> stk)

{

return RemoveMaxRec(stk, stk.Top());

}

public static int RemoveMaxRec(Stack<int> stk, int currMax)

{

}

public static void PrintTopButtomRec<T>(Stack<T> stk)

{

}

public static void PrintButtomTopRec<T>(Stack<T> stk)

{

}

public static int GetMax(Stack<int> stk)

{

}

public static int GetMin(Stack<int> stk)

{

}

public static T GetLast<T>(Stack<T> stk, bool remove)

{

}

}

}