פעולות עזר לבגרות 271 – C#

תוכן עניינים

נכתב ע״י גיא אלרם, הגימנסיה הריאלית, ראשון לציון

C# – 271 פעולות עזר לבגרות
רקורסיה

חישוב מכפלת שני מספרים

```
public static int Mult (int x, int y)
{
	if (y == 1)
		return x;
	return Mult (x, y - 1) + x;
}
```

כמה ספרות יש במספר

```
public static int DigitNum (int num)
{
	if (num < 10)
		return 1;
	 return DigitNum (num / 10) + 1;
}
```

סכום ספרות במספר

```
public static int DigitSum (int num)
{
	if (num < 10)
		return 1;
	 return DigitSum (num / 10) + num %10;
}
```

כמות הפעמים שספרה נמצאת במספר

```
public static int CountDigitInNum (int num, int d)
{
	if (num < 10)
	{
		if (num == d)
			return 1;
		return 0;
	}
	 else
	{
		if (num%10 == d)
			return 1 + CountDigitInNum (num / 10, d);
		return CountDigitInNum (num / 10, d);
	}
}
```

חישוב חזקה a בחזקת b

```
public static int Power (int a, int b)
{
	if (b == 0)
		return 1;
	return Power (a, b - 1) * a;
}
```

האם ספרה נמצאת במספר
אופציה א

```
public static bool IsExist (int num, int digit)
{
	if (n < 10)
		 return (num == digit);
	return (num%10 == digit) || IsExist (num /10, digit);
}
```

אופציה ב

```
public static bool IsExist (int num, int digit)
{
	if (num < 10)
	{
		if (num == digit)
			return true;
		return false;
	}
	if (num % 10 == digit)
		return true;
	return IsExist (num /10, digit);
}
```

האם כל הספרות במספר זוגיות

אופציה א:

```
public static bool AllEven (int num)
{
	if (num < 10)
		 return (num % 2 == 0);
	return (num % 2 == 0) && AllEven (num/10);
}
```

אופציה ב:

```
public static bool AllEven (int num)
{
	if (num < 10)
	{
		if (num % 2 == 0)
			return true;
		return false;
	}
	 if (num % 2 == 0)
		return AllEven (num /10);
	return false;
}
```

פעולות עזר לבגרות 271 – C#
רשימה מקושרת

ממשק רשימה מקושרת

הפעולה	מה מבצעת הפעולה
	פעולות בונות:
Node<T> lst = new Node<T>(value);	פעולה בונה שמקבלת ערך, next = null
Node<T> lst = new Node<T>(value, next);	פעולה בונה שמקבלת ערך והפניה לצומת הבא
<T> p.GetValue()	הפעולה מחזירה את ה-Value שיש בצומת
void p.SetValue(<T> x)	הפעולה משנה את ה-Value שיש בצומת
Node<T> p.GetNext()	הפעולה מחזירה את הכתובת של הצומת הבא ברשימה. אם אין כזה, יוחזר null
void p.SetNext(Node<T>)	הפעולה משנה את הכתובת של הצומת הבא ברשימה
bool p.HasNext()	הפעולה מחזירה true אם ה-Next שונה מ-null, אחרת false
string p.ToString()	הפעולה מדפיסה ערך של צומת בודד כמחרוזת

פעולות בסיסיות

בניית רשימה של מספרים מ-1 עד 10 (הכנסה לסוף הרשימה)

```
Node<int> lst = new Node<int> (1);
Node<int> p = lst;
for (int i = 2; i <=10; i++)
{
        p.SetNext (new Node <Int>(i));
        p = p.GetNext();
}
```

בניית רשימה הפוכה (מהסוף להתחלה) מ-1 עד 10 (הכנסה לתחילת הרשימה)

```
Node<int> lst = new Node<int> (10);
for (int i = 9; i >= 1; i--)
        lst = new Node<int>(i, lst);
```

פעולה שמחזירה את אורך הרשימה.

```
public static int ListLength (Node<int> lst)
{
        Node<int> p = lst;
        int count = 0;
        while (p != null)
        {
                count++;
                p = p.GetNext();
        }
        return count;
}
```

פעולה שמחזירה את סכום ערכי הצמתים ברשימה.

```
public static int SumList (Node<int> lst)
{
        Node<int> p = lst;
        int sum = 0;
        while (p != null)
        {
                sum += p.GetValue();
                p = p.GetNext();
        }
        return sum;
}
```

C# – 271 פעולות עזר לבגרות

פעולה שמחזירה את המספר הגדול ביותר ברשימה לא ריקה.

```
public static int LargestNum  (Node<int> lst)
{
	Node<int> p = lst;
	int max = p.GetValue();
	p = p.GetNext();
	while (p != null)
	{
		if (p.GetValue() > max)
			max = p.GetValue();
		p = p.GetNext();
	}
	return max;
}
```

פעולה שמחזירה את מס' הפעמים שבו מופיע מספר ברשימה.

```
public static int NumExist (Node<int> lst, int num)
{
	Node<int> p = lst;
	int count = 0;
	while (p != null)
	{
		if (p.GetValue() == num)
			count++;
		p = p.GetNext();
	}
	return count;
}
```

פעולה המקבלת רשימה שמחזירה true אם מספר קיים ברשימה. אחרת יוחזר false.

```
public static bool IsExist (Node<int> lst, int num)
{
	Node<int> p = lst;
	while (p != null)
	{
		if (p.GetValue() == num)
			return true;
		p = p.GetNext();
	}
	return false;
}
```

פעולה שמחזירה true אם הרשימה ממוינת בסדר עולה. אחרת יוחזר false. הנחה: הרשימה לא ריקה.

```
public static bool IsSorted (Node<int> lst)
{
	Node<int> p = lst;
	while (p.GetNext() != null)
	{
		if (p.GetValue() > p.GetNext().GetValue())
			return false;
		p = p.GetNext();
	}
	return true;
}
```

פעולה שמחזירה true אם הרשימה מהווה סדרה חשבונית. אחרת יוחזר false.

```
public static bool IsArithmetic (Node<int> lst)
{
	Node<int> p = lst;
	if (lst == null)
		return false;
	if (lst.GetNext() == null)
		return true;
	int diff = p.GetNext().GetValue() – p.GetValue();
	while (p.GetNext() != null)
	{
		if (diff != p.GetNext().GetValue() – p.GetValue())
			return false;
		p = p.GetNext();
	}
	return true;
}
```

פעולות שמחזירות מצביע (הפניה)

פעולה שמחזירה הפניה (מצביע) לצומת במקום ה-step. אם step גדול מאורך הרשימה, יוחזר null.

```
public static Node<int> PointToSteps (Node<int> lst, int step)
{
	Node<int> p = lst;
	for (int i = 1; p != null && i < step; i++)
		p = p.GetNext();
	return p;
}
```

פעולה שמחזירה הפניה (מצביע) למיקום הראשון ברשימה שבו מופיע מספר שלם num. אם לא נמצא - יוחזר null.

```
public static Node<int> PointToNum (Node<int> lst, int num)
{
	Node<int> p = lst;
	while (p != null && p.GetValue() != num)
		p = p.GetNext();
	return p;
}
```

פעולה שמחזירה הפניה (מצביע) למיקום של המספר הקטן ביותר ברשימה. אם הרשימה ריקה, מוחזר null.

```
public static Node<int> PointToMinValue (Node<int> lst)
{
	if (lst == null)
		return null;
	Node<int> p = lst;
	Node <Int> minNode = lst;
	while (p != null)
	{
		if (p.GetValue() < minNode.GetValue())
			minNode = p;
		p = p.GetNext();
	}
	return minNode;
}
```

פעולה שמחזירה את המצביע לצומת שלפני המצביע pos (הנחה: המצביע pos מצביע למיקום כלשהו ברשימה)

```
public static Node<int> PointToPrev (Node<int> lst, Node<int> pos)
{
	Node<int> p = lst;
	while (p.GetNext() != pos)
		p = p.GetNext();
	return p;
}
```

פעולות עזר לבגרות 271 – C#

פעולות הכנסה והוצאה מרשימה

פעולה המקבלת רשימה של מספרים שלמים, מספר שלם step ומספר שלם num. הפעולה תחזיר רשימה חדשה שבה צומת עם הערך num תוכנס אחרי step צמתים. אם step גדול או שווה לאורך הרשימה, הצומת יוכנס לסוף הרשימה. אם step=0, הצומת יוכנס לתחילת הרשימה.

```
public static Node<int> AfterSteps (Node<int> lst, int step, int num)
{
	if (step == 0)
		lst = new Node<int> (num, lst);
	else
	{
		Node<int> p = lst;
		for (int i = 1; i < p.GetNext() != null  &&step-1; i++)
			p = p.GetNext();
		p.SetNext(new Node<int> (num, p.GetNext()));
	}
	return lst;
}
```

פעולה המקבלת רשימה ממוינת בסדר עולה של מספרים שלמים וכן מספר num. הפעולה תחזיר רשימה חדשה עם צומת עם המספר num, כך שהרשימה תישאר ממוינת.

```
public static Node<int> StillSorted (Node<int> lst, int num)
{
	if (num < lst.GetValue())
		lst = new Node<int> (num, lst);
	else
	{
		Node<int> p = lst;
		while (p.GetNext() != null && num > p.GetNext().GetValue())
			p = p.GetNext();
		p.SetNext (New Node<int>( num, p.GetNext()));
	}
	return lst;
}
```

פעולה המקבלת רשימה של מספרים שלמים ומספר step שלם המייצג מיקום ברשימה. הפעולה תחזיר רשימה חדשה ללא הצומת במיקום ה-step ברשימה. אם step=1, הרשימה תוחזר ללא הצומת הראשון ברשימה.

<u>הנחה</u>: step גדול מ-0 וקטן או שווה לאורך הרשימה

```
public static Node<int> DeleteAfterSteps (Node<int> lst, int step)
{
	if (step == 1)
		lst = lst.GetNext();
	else
	{
		Node<int> p = lst;
		for (int i = 1; p.GetNext() != null  && i < step-1; i++)
			p = p.GetNext();
		if (p.GetNext() != null)
			p.SetNext(p.GetNext().GetNext());
	}
	return lst;
}
```

C# – 271 פעולות עזר לבגרות

מיזוג רשימות

פעולה המקבלת 2 רשימות לא ריקות וממוינות בסדר עולה של מס' שלמים. הערכים בשתי הרשימות הם ייחודיים ולא חוזרים על עצמם. הפעולה תחזיר רשימה חדשה אחת ממוזגת וממוינת בסדר עולה.

אפשרות א (עם צומת פיקטיבי):

```
public static Node<int> Merge (Node<int> lst1, Node<int> lst2)
{
	Node<int> lst3 = new Node<int>(-999);
	Node<int> q = lst3;
	while (lst1 != null && lst2 != null)
	{
		if (lst1.GetValue() < lst2.GetValue())
		{
			q.SetNext(new Node<int>(lst1.GetValue());
			lst1 = lst1.GetNext();
		}
		else
		{
			q.SetNext(new Node<int>(lst2.GetValue());
			lst2 = lst2.GetNext();
		}
		q = q.GetNext();
	}
	if (lst1 == null)
		q.SetNext(lst2);
	else
		q.SetNext(lst1);
	return lst3.GetNext();
}
```

אפשרות ב (ללא צומת פיקטיבי):

```
public static Node<int> Merge (Node<int> lst1, Node<int> lst2)
{
	Node<int> lst3 = null;
	Node<int> q;
	while (lst1 != null && lst2 != null)
	{
		if (lst1.GetValue() < lst2.GetValue())
		{
			if (lst3 == null)
			{
				lst3 = new Node<int>(lst1.GetValue());
				q = lst3;
			}
			else
			{
				q.SetNext(new Node<int>(lst1.GetValue()));
				q = q.GetNext();
			}
			lst1 = lst1.GetNext();
		}
		else
		{
			if (lst3 == null)
			{
				lst3 = new Node<int>(lst2.GetValue());
				q = lst3;
			}
```

```
                else
                {
                        q.SetNext(new Node<int>(lst2.GetValue()));
                        q = q.GetNext();
                }
                lst2 = lst2.GetNext();
        }
    }
    if (lst1 == null)
        q.SetNext(lst2);
    else
        q.SetNext(lst1);
    return lst3;
}
```

פעולות רקורסיה ברשימה

פעולה המקבלת רשימה של מספרים ומחזירה את אורכה.

```
public static int CountLength (Node<int> lst)
{
    if (lst == null)
        return 0;
    return 1 + CountLength(lst.GetNext());
}
```

פעולה המקבלת רשימה של מספרים ומחזירה את סכום ערכיה.

```
public static int SumNodes (Node<int> lst)
{
    if (lst == null)
        return 0;
    return lst.GetValue() + SumNodes(lst.GetNext());
}
```

פעולה המקבלת רשימה של מספרים ומחזירה את מכפלת ערכיה.

```
public static int MulNodes (Node<int> lst)
{
    if (lst == null)
        return 1;
    if (lst.GetNext() == null)
        return lst.GetValue();
    return lst.GetValue() * MulNodes (lst.GetNext());
}
```

פעולה רקורסיבית שמחזירה את המספר הגדול ביותר ברשימה לא ריקה.

```
public static int MaxNumInList (Node<int> lst)
{
    if (lst.GetNext() == null)
        return lst.GetValue();
    return Math.max(lst.GetValue(), MaxNumInList (lst.GetNext()));
}
```

C# – 271 פעולות עזר לבגרות

פעולה רקורסיבית שמחזירה את מספר הפעמים ש-num מופיע ברשימה.

```
public static int CountNum (Node<int> lst, int num)
{
	if (lst == null)
		return 0;
	if (lst.GetValue() == num)
		return 1 + CountNum (lst.GetNext(), num);
	return CountNum (lst.GetNext(), num);
}
```

פעולה רקורסיבית שמחזירה true אם num מופיע ברשימה. אחרת יוחזר false.

```
public static bool IsExist (Node<int> lst, int num)
{
	if (lst == null)
		return false;
	if (lst.GetValue() == num)
		return true;
	return IsExist (lst.GetNext(), num);
}
```

פעולה רקורסיבית שמחזירה true אם הרשימה ממוינת בסדר עולה. אחרת יוחזר false.

```
public static bool IsSorted (Node<int> lst)
{
	if (lst == null)
		return false;
	if(lst.GetNext() == null)
		return true;
	if (lst.GetValue() > lst.GetNext().GetValue())
		return false;
	return IsSorted (lst.GetNext());
}
```

פעולה רקורסיבית שמחזירה הפניה לצומת במקום ה-step. אם step גדול ממס' הצמתים ברשימה, יוחזר null.

```
public static Node<int> GoToStep (Node<int> lst, int step)
{
	if (lst == null || step == 1)
		return lst;
	return GoToStep (lst.GetNext(), step-1);
}
```

פעולה רקורסיבית שמחזירה הפניה לצומת האחרון ברשימה. אם הרשימה ריקה, יוחזר null.

```
public static Node<int> GoToLast (Node<int> lst)
{
	if (lst == null || lst.GetNext() == null)
		return lst;
	return GoToLast(lst.GetNext());
}
```

פעולה רקורסיבית שמחזירה הפניה לצומת הראשון שבו מופיע המספר num. אם הוא לא מופיע – יוחזר null.

```
public static Node<int> FirstInstancePointer (Node<int> lst, int num)
{
	if (lst == null || lst.GetValue() == num)
		return lst;
	return FirstInstancePointer (lst.GetNext(), num);
}
```

פעולות עזר לבגרות 271 – C#

רשימה של עצמים

בניית צומת עם אובייקט חדש מטיפוס Client עם תכונות name, age:

```
Node<Client> c = new Node<Client> (new Client(name, age));
```

קריאה לערך של תכונה של עצם ברשימה

```
c.GetValue().GetName()
```

שינוי ערך של תכונה של עצם ברשימה

```
c.GetValue().SetName("Yossi")
```

<u>שימו לב</u>: לא ניתן לבצע setValue לערך מטיפוס עצם, אלא רק לתכונה. לכן עלינו לקרוא <u>לכל העצם</u> באמצעות פעולת getValue, ורק עליו להפעיל את פעולת ה-set ל<u>תכונה ספציפית</u>.

C# – 271 פעולות עזר לבגרות
תור

ממשק תור

מה מבצעת הפעולה	הפעולה
פעולה בונה – יוצרת תור ריק מטיפוס T	Queue<T> q = new Queue<T>();
האם התור ריק – יוחזר true כאשר התור ריק אחרת false	bool IsEmpty()
הפעולה מכניסה איבר חדש לסוף התור	void Insert (T x)
הפעולה מוציאה ומחזירה את האיבר בראש התור	T Remove()
הפעולה מחזירה את האיבר בראש התור (מבלי להוציא אותו)	T Head()
הפעולה מדפיסה את כל האיברים בתור	string ToString()

פעולות בסיסיות

פעולה שמחזירה את אורך התור.

```
public static int QueueLength (Queue<int> q)
{
	int count = 0;
	Queue<int> temp = new Queue<int>();
	while (!q.IsEmpty())
	{
		count++;
		temp.Insert(q.Remove());
	}
	while (!temp.IsEmpty())
		q.Insert(temp.Remove());
	return count;
}
```

פעולה שמחזירה את סכום האיברים בתור.

```
public static int QueueSum (Queue<int> q)
{
	int Sum = 0;
	Queue<int> temp = new Queue<int>();
	while (!q.IsEmpty())
	{
		sum += q.Head();
		temp.Insert(q.Remove());
	}
	while (!temp.IsEmpty())
		q.Insert(temp.Remove());
	return Sum;
}
```

פעולה שמחזירה תור חדש שהוא העתק של התור המקורי.

```
public static Queue<int> Clone (Queue<int> q)
{
	Queue<int> copy = new Queue<int>();
	Queue<int> temp = new Queue<int>();
	while (!q.IsEmpty())
	{
		copy.Insert(q.Head());
		temp.Insert(q.Remove());
	}
	while (!temp.IsEmpty())
		q.Insert(temp.Remove());
	return copy;
}
```

פעולה שמחזירה ומוציאה את האיבר האחרון בתור.

```
public static int LastNum(Queue<int> q)
{
	Queue<int> temp = new Queue<int>();
	int num;
	while(!q.IsEmpty())
	{
		num = q.Remove();
		if (!q.IsEmpty())
			temp.Insert(num);
	}
	while (!temp.IsEmpty())
		q.Insert(temp.Remove());
	return num;
}
```

פעולה שמחזירה את האיבר הכי גדול בתור לא ריק.

```
public static int MaxNum (Queue<int> q)
{
	int max = q.Head();
	Queue<int> temp = new Queue<int>();
	while (!q.IsEmpty())
	{
		if (q.Head() > max)
			max = q.Head();
		temp.Insert(q.Remove());
	}
	while (!temp.IsEmpty())
		q.Insert(temp.Remove());
	return max;
}
```

פעולה שמחזירה את **מיקום** של num בתור. המיקום של האיבר הראשון בתור הוא 1. אם המס' לא נמצא, יוחזר 1-.

```
public static int NumIndexInQueue (Queue<int> q, int num)
{
	Queue<int> temp = new Queue<int>();
	int index = 1;
	bool found = false;
	while (!q.IsEmpty() && q.Head() != num)
	{
		temp.Insert(q.Remove());
		index++;
	}
	if (!q.IsEmpty())
	{
		found = true;
		while (!q.IsEmpty())
			temp.Insert(q.Remove());
	}
	while (!temp.IsEmpty())
		q.Insert(temp.Remove());
	if (found)
		return index;
	return -1;
}
```

פעולות עזר לבגרות 271 – C#

פעולה המקבלת תור של מס' שלמים חיוביים ומס' k שלם וחיובי. הפעולה מחזירה את המספר שנמצא במיקום ה-k בתור.
אם k גדול ממס' האיברים בתור, הפעולה תחזיר 1-.

```
public static int NumInQueue (Queue<int> q, int k)
{
	Queue<int> temp = new Queue<int>();
	int found = -1;
	while (!q.IsEmpty() && k > 1)
	{
		temp.Insert(q.Remove());
		k--;
	}
	if (!q.IsEmpty())
	{
		found = q.Head();
		while (!q.IsEmpty())
			temp.Insert(q.Remove());
	}
	while (!temp.IsEmpty())
		q.Insert(temp.Remove());
	return found;
}
```

פעולה שמחזירה true אם מספר נמצא בתור. אחרת יוחזר false.

```
public static bool IsExist (Queue<int> q, int num)
{
	Queue<int> temp = new Queue<int>();
	bool numInQueue = false;
	while (!q.IsEmpty())
	{
		if (q.Head() == num)
			numInQueue = true;
		 temp.Insert(q.Remove());
	}
	while (!temp.IsEmpty())
		q.Insert(temp.Remove());
	return numInQueue;
}
```

פעולה שמחזירה true אם התור ממוין בסדר עולה. אחרת יוחזר false.

```
public static bool Ascend (Queue<int> q)
{
	Queue<int> temp = new Queue<int>();
	bool sorted = true;
	int num;
	while (!q.IsEmpty())
	{
		 num = q.Head();
		temp.Insert(q.Remove());
		if (!q.IsEmpty() && q.Head() < num)
			sorted = false;
	}
	while (!temp.IsEmpty())
		q.Insert(temp.Remove());
	return sorted;
}
```

פעולה שמחזירה true אם שני תורים זהים זה לזה. אחרת יוחזר false.

```
public static bool SameQueues (Queue<int> q1, Queue<int> q2)
{
        if (QueueLength (q1) != QueueLength (q2))
                return false;
        bool sameQueues = true;
        Queue<int> temp1 = new Queue<int>();
        Queue<int> temp2 = new Queue<int>();
        while (!q1.IsEmpty())
        {
                if (q1.Head() != q2.Head())
                        sameQueues = false;
                 temp1.Insert(q1.Remove());
                 temp2.Insert(q2.Remove());
        }
        while (!temp1.IsEmpty())
        {
                 q1.Insert(temp1.Remove());
                q2.Insert(temp2.Remove());
        }
        return sameQueues;
}
```

פעולה המקבלת תור ממוין (בסדר עולה) של מספרים שלמים ומספר שלם, ומכניסה את המספר לתור, כך שיישאר ממוין.

```
public static void RemainSorted (Queue<int> q, int num)
{
        Queue<int> temp = new Queue<int>();
        while (!q.IsEmpty() && q.Head() < num)
                temp.Insert(q.Remove());
        temp.Insert(num);
        while (!q.IsEmpty())
                temp.Insert(q.Remove());
        while (!temp.IsEmpty())
                q.Insert(temp.Remove());
}
```

פעולות רקורסיה בתור

פעולה רקורסיבית שהופכת את סדר האיברים בתור.

```
public static void Opposite (Queue<int> q)
{
        if (!q.IsEmpty())
        {
                int x = q.Remove();
                Opposite (q);
                q.Insert(x);
        }
}
```

פעולה רקורסיבית שמחזירה את אורך התור (שימו לב: התור מתהפך בסיום)

```
public static int QLength (Queue<int> q)
{
        if (q.IsEmpty())
                return 0;
        int x = q.Remove();
        int num = 1 + QLength (q);
        q.Insert (x);
        return num;
}
```

פעולה רקורסיבית שמחזירה את סכום המספרים בתור (שימו לב: התור מתהפך בסיום)

```
public static int QSum (Queue<int> q)
{
	if (q.IsEmpty())
		return 0;
	int x = q.Remove();
	int num = x + QSum (q);
	q.Insert (x);
	return num;
}
```

פעולה רקורסיבית שמחזירה את כמות המספרים הזוגיים בתור (שימו לב: התור מתהפך בסיום)

```
public static int CountEven (Queue<int> q)
{
	if (q.IsEmpty())
		return 0;
	int x = q.Remove();
	int num;
	if (x % 2 == 0)
		num = 1 + CountEven (q);
	else
		num = CountEven (q);
	q.Insert (x);
	return num;
}
```

פעולה רקורסיבית שמחזירה true אם מספר קיים בתור. אם לא, הפעולה תחזיר false (שימו לב: התור מתהפך בסיום).

```
public static bool IsExist (Queue<int> q, int num)
{
	if (q.IsEmpty())
		return false;
	int x = q.Remove();
	bool exist = (x == num) || IsExist(q, num);
	q.Insert(x);
	return exist;
}
```

תור של עצמים

בניית תור ריק חדש של אובייקטים מטיפוס Client עם תכונות name, age:

```
Queue<Client> q = new Queue <Client>();
```

הכנסת אובייקט חדש מטיפוס Client לסוף התור

```
q.Insert (new Client(name, age));
```

קריאה לערך של תכונה של אובייקט בראש התור

```
q.Head().GetName()
```

שינוי ערך של תכונה של אובייקט בראש התור

```
q.Head().SetName("Yossi")
```

הוצאת ושמירת אובייקט מראש התור של הלקוחות

```
Client c = q.Remove();
```

C# – 271 פעולות עזר לבגרות
מחסנית

ממשק מחסנית

מה מבצעת הפעולה	הפעולה
פעולה בונה – יוצרת מחסנית ריקה מטיפוס T	Stack<T> s = new Stack<T>();
האם מחסנית ריקה – יוחזר true אחרת false	bool IsEmpty()
הפעולה מכניסה איבר חדש לראש המחסנית	void Push (T x)
הפעולה מוציאה ומחזירה את האיבר בראש המחסנית	T Pop ()
הפעולה מחזירה את האיבר בראש המחסנית (מבלי להוציא אותו)	T Top ()
הפעולה מדפיסה את כל האיברים במחסנית כמחרוזת	string ToString()

פעולות בסיסיות

פעולה שמחזירה את אורך המחסנית.

```
public static int StackLength (Stack<int> s)
{
	int count = 0;
	Stack<int> temp = new Stack<int>();
	while (!s.IsEmpty())
	{
		count++;
		temp.Push (s.Pop());
	}
	while (!temp.IsEmpty())
		s.Push (temp.Pop());
	return count;
}
```

פעולה שמחזירה את סכום האיברים במחסנית.

```
public static int SumStack (Stack<int> s)
{
	int sum = 0;
	Stack<int> temp = new Stack<int>();
	while (!s.IsEmpty())
	{
		sum += s.Top();
		 temp.Push (s.Pop());
	}
	while (!temp.IsEmpty())
		s.Push (temp.Pop());
	return sum;
}
```

פעולה שמחזירה מחסנית חדשה שהיא העתק של המחסנית המקורית.

```
public static Stack<int> Clone (Stack<int> s)
{
	Stack<int> copy = new Stack<int>();
	Stack<int> temp = new Stack<int>();

	while (!s.IsEmpty())
		temp.Push(s.Pop());
	while (!temp.IsEmpty())
	{
		s.Push(temp.Top());
		copy.Push(temp.Pop();
	}
	return copy;
}
```

פעולה שמחזירה ומוציאה את האיבר בתחתית המחסנית.

```
public static int LastNum(Stack<int> s)
{
	Stack<int> temp = new Stack<int>();
	int num;
	while(!s.IsEmpty())
	{
		num = s.Pop();
		if (!s.IsEmpty())
			temp.Push(num);
	}
	while (!temp.IsEmpty())
		s.Push(temp.Pop());
	return num;
}
```

פעולה שמחזירה את האיבר הכי גדול במחסנית לא ריקה.

```
public static int MaxNum (Stack<int> s)
{
	int max = s.Top();
	Stack<int> temp = new Stack<int>();
	while (!s.IsEmpty())
	{
		if s.Top() > max)
			max = s.Top();
		temp.Push(s.Pop());
	}
	while (!temp.IsEmpty())
		s.Push(temp.Pop());
	return max;
}
```

פעולה שמחזירה את **מיקום** של num במחסנית. המיקום של האיבר הראשון במחסנית הוא 1. אם המס' לא נמצא, יוחזר 1-.

```
public static int NumIndexInStack (Stack<int> s, int num)
{
	Stack<int> temp = new Stack<int>();
	int index = 1;
	bool found = false;
	while (!s.IsEmpty() && s.Top() != num)
	{
		temp.Push(s.Pop());
		index++;
	}
	if (!s.IsEmpty())
	{
		found = true;
		while (!s.IsEmpty())
			temp.Push(s.Pop());
	}
	while (!temp.IsEmpty())
		s.Push(temp.Pop());
	if (found)
		return index;
	return -1;
}
```

C# – 271 פעולות עזר לבגרות

פעולה המקבלת מחסנית של מס' שלמים חיוביים ומס' k שלם וחיובי. הפעולה מחזירה את המספר שנמצא במיקום ה-k במחסנית. אם k גדול ממס' האיברים במחסנית, הפעולה תחזיר 1-.

```
public static int NumInStack (Stack<int> s, int k)
{
	Stack<int> temp = new Stack<int>();
	int found = -1;
	while (!s.IsEmpty() && k > 1)
	{
		temp.Push(s.Pop());
		k--;
	}
	if (!q.IsEmpty())
	{
		found = s.Top();
		while (!s.IsEmpty())
			temp.Push(s.Pop());
	}
	while (!temp.IsEmpty())
		s.Push(temp.Pop());
	return found;
}
```

פעולה שמחזירה true אם מספר נמצא במחסנית. אחרת יוחזר false.

```
public static bool IsExist (Stack<int> s, int num)
{
	Stack<int> temp = new Stack<int>();
	bool exists = false;
	while (!s.IsEmpty())
	{
		if(s.Top() == num)
			exists = true;
		temp.Push(s.Pop());
	}
	while (!temp.IsEmpty())
		s.Push(temp.Pop());
	return exists;
}
```

פעולה שמחזירה true אם המחסנית ממוינת בסדר עולה. אחרת יוחזר false.

```
public static bool Ascend (Stack<int> s)
{
	Stack<int> temp = new Stack<int>();
	bool sorted = true;
	int num;
	while (!s.IsEmpty())
	{
		temp.Push(s.Pop());
		if (!s.IsEmpty() && s.Top() > temp.Top())
			sort = false;
	}
	while (!temp.IsEmpty())
		s.Push(temp.Pop());
	return sort;
}
```

פעולה שמחזירה true אם שתי מחסניות זהות זו לזו. אחרת יוחזר false.

```
public static bool Same (Stack<int> s1, Stack<int> s2)
{
	if (stackLength(s1) != stackLength(s2))
		return false;
	Stack<int> temp1 = new Stack<int>();
	Stack<int> temp2 = new Stack<int>();
	bool same = true;
	while (!s1.IsEmpty())
	{
		if (s1.Top() != s2.Top())
			same = false;
		temp1.Push(s1.Pop());
		temp2.Push(s2.Pop());
	}
	while (!temp1.IsEmpty())
	{
		s1.Push(temp1.Pop());
		s2.Push(temp2.Pop());
	}
	return same;
}
```

פעולה המקבלת מחסנית ממוינת (בסדר עולה) של מספרים שלמים ומספר שלם, ומכניסה את המספר למחסנית, כך שתישאר ממוינת.

```
public static void RemainSorted (Stack<int> s, int num)
{
	Stack<int> temp = new Stack<int>();
	while (!s.IsEmpty() && s.Top() < num)
		temp.Push(s.Pop());
	temp.Push(num);
	while (!s.IsEmpty())
		temp.Push(s.Pop());
	while (!temp.IsEmpty())
		s.Push(temp.Pop());
}
```

פעולות רקורסיה במחסנית

פעולה רקורסיבית שמחזירה את אורך המחסנית

```
public static int StackLength (Stack<int> s)
{
	if (s.IsEmpty())
		return 0;
	int num = s.Pop();
	int ans = 1 + StackLength (s);
	s.Push(num);
	return ans;
}
```

פעולה רקורסיבית שמחזירה את סכום המספרים במחסנית

```
public static int StackSum (Stack <Int> s)
{
	if (q.IsEmpty())
		return 0;
	int num = s.Pop();
	int sum = num + StackSum (s);
	s.Push (num);
	return sum;
}
```

פעולה רקורסיבית שמחזירה את כמות המספרים הזוגיים במחסנית

```
public static int CountEven (Stack<int> s)
{
        if (s.IsEmpty())
                return 0;
        int ans;
        int num = s.Pop();
        if (num %2 == 0)
                ans = 1 + CountEven (s);
        else
                ans = CountEven (s);
        s.Push(num);
        return ans;
}
```

פעולה רקורסיבית שמחזירה true אם num קיים במחסנית. אם לא, הפעולה תחזיר false

```
public static bool IsExist (Stack<int> s, int num)
{
        if (s.IsEmpty())
                return false;
        if (num == s.Top())
                return true;
        int n = s.Pop();
        bool ans = IsExist (s, num);
        s.Push(n);
        return ans;
}
```

מחסנית של עצמים

בניית תור ריק חדש של אובייקטים מטיפוס Client עם תכונות name, age:

```
Stack<Client> s = new Stack <Client>();
```

הכנסת אובייקט חדש מטיפוס Client לראש המחסנית

```
s.Push (new Client(name, age));
```

קריאה לערך של תכונה של אובייקט בראש המחסנית

```
s.Top().GetName()
```

שינוי ערך של תכונה של אובייקט בראש המחסנית

```
s.Top().SetName("Yossi")
```

הוצאת ושמירת אובייקט מראש המחסנית של הלקוחות

```
Client c = s.Pop();
```

C# – 271 פעולות עזר לבגרות
עצים בינאריים

ממשק עץ בינארי

מה מבצעת הפעולה	הפעולה
פעולות בונות:	
פעולה בונה צומת עם ערך, left=null, right=null	BinNode<T> bt = new BinNode<T>(value)
פעולה בונה צומת עם ערך והפניות ל-left, right	BinNode<T> bt = new BinNode<T>(left, value, right)
פעולה המחזירה מצביע לבן השמאלי	BinNode<T> bt.GetLeft ()
פעולה המחזירה מצביע לבן הימני	BinNode<T> bt.GetRight ()
פעולה היוצרת צומת חדש משמאל	void bt.SetLeft (new BinNode<T>(value))
פעולה היוצרת צומת חדש מימין	void bt.SetRight (new BinNode<T>(value))
פעולה המחזירה true אם לצומת יש בן שמאלי, אחרת false	bool bt.HasLeft ()
פעולה המחזירה true אם לצומת יש בן ימני, אחרת false	bool bt.HasRight ()
פעולה המחזירה את הערך של הצומת	T bt.GetValue ()
פעולה המשנה את הערך של הצומת	void bt.SetValue (T x)
פעולת ההדפסה של ערך של צומת בעץ כמחרוזת	string bt.ToString ()

פעולות מונות/סוכמות בעץ (כל הפעלות מתייחסות לעץ של מס' שלמים)

פעולה המקבלת עץ ומחזירה את כמות הצמתים בעץ

```
public static int TreeSize (BinNode<int> bt)
{
	if (bt == null)
		return 0;
	return 1 + TreeSize (bt.GetLeft()) + TreeSize (bt.GetRight());
}
```

פעולה המקבלת עץ ומחזירה את סכום הצמתים בעץ

```
public static int SumTree(BinNode<int> bt)
{
	if (bt == null)
		return 0;
	return bt.GetValue() + SumTree(bt.GetLeft()) + SumTree(bt.GetRight());
}
```

פעולה המקבלת עץ ומחזירה את סכום הערכים הזוגיים בעץ

```
public static int SumEven(BinNode<int> bt)
{
	if (bt == null)
		return 0;
	if(bt.GetValue() % 2 == 0)
		return bt.GetValue() + SumEven(bt.GetLeft()) +
			sumEven(bt.GetRight());
	return SumEven(bt.GetLeft()) + SumEven(bt.GetRight());
}
```

פעולות עזר לבגרות 271 – C#

פעולה המקבלת עץ ומחזירה את כמות העלים בעץ

```
public static int CountLeaf(BinNode<int> bt)
{
        if (bt == null)
                return 0;
        if(bt.GetLeft() == null && bt.GetRight() == null)
                return 1;
        return CountLeaf (bt.GetLeft()) + CountLeaf (bt.GetRight());
}
```

פעולה המקבלת עץ ומחזירה את סכום העלים בעץ

```
public static int SumLeaf(BinNode<int> bt)
{
        if (bt == null)
                return 0;
        if(bt.GetLeft() == null && bt.GetRight() == null)
                return bt.GetValue();
        return SumLeaf(bt.GetLeft()) + SumLeaf(bt.GetRight());
}
```

פעולה המקבלת עץ ומחזירה את מספר הצמתים שיש להם שני בנים.

```
public static int TwoSons(BinNode<int> bt)
{
    if (bt == null)
        return 0;
    if(bt.GetLeft() != null  && bt.GetRight() != null)
        return 1 + TwoSons (bt.GetLeft()) + TwoSons (bt.GetRight());

    return TwoSons (bt.GetLeft()) + TwoSons (bt.GetRight());
}
```

פעולה המקבלת עץ ומחזירה את מספר הצמתים, שערכם גדול מערך הבנים שלהם. עלים לא נספרים.

```
public static int BigPapa(BinNode<int> bt)
{
    if (bt == null)
        return 0;
    if (bt.GetLeft() != null  && bt.GetRight() != null)
    {
        if (bt.GetValue() > bt.GetRight().GetValue() && bt.GetValue() > bt.GetLeft().GetValue())
                return 1 + BigPapa(bt.GetLeft()) + BigPapa(bt.GetRight());
    }
    if (bt.GetLeft() != null  && bt.GetRight() == null)
    {
        if (bt.GetValue() > bt.GetLeft().GetValue())
                return 1 + BigPapa(bt.GetLeft()) + BigPapa(bt.GetRight());
    }
    if (bt.GetLeft() == null  && bt.GetRight() != null)
    {
        if (bt.GetValue() > bt.GetRight().GetValue())
                return 1 + BigPapa(bt.GetLeft()) + BigPapa(bt.GetRight());
    }
    return BigPapa(bt.GetLeft()) + BigPapa(bt.GetRight());
}
```

פעולה המקבלת עץ ומחזירה את סכום הבנים היחידים (אלה שאין להם אחים)

```
public static int SumOnlySons(BinNode<int> bt)
{
	if (bt == null)
		return 0;
	if (bt.GetLeft() == null  && bt.GetRight() != null)
		return bt.GetRight().GetValue() + SumOnlySons(bt.GetLeft()) + SumOnlySons(bt.GetRight());
	if (bt.GetLeft() != null  && bt.GetRight() == null)
		return bt.GetLeft().GetValue() + SumOnlySons(bt.GetLeft()) + SumOnlySons(bt.GetRight());
	return SumOnlySons(bt.GetLeft()) + SumOnlySons(bt.GetRight());
}
```

פעולה המקבלת עץ וערך num, ומחזירה את מס' הפעמים ש-num מופיע בעץ

```
public static int CountNum(BinNode<int> bt, int num)
{
	if (bt == null)
		return 0;
	if(bt.GetValue() == num)
		return 1 + CountNum (bt.GetLeft(), num) + CountNum (bt.GetRight(), num);
	return CountNum (bt.GetLeft(), num) + CountNum (bt.GetRight(), num);
}
```

פעולה המקבלת עץ ורמה בעץ level, ומחזירה את מס' הצמתים באותה רמה

```
public static int CountLevel(BinNode<int> bt, int level)
{
	if (bt == null)
		return 0;
	if (level == 0)
		return 1;
	return CountLevel (bt.GetLeft(), level-1) + CountLevel (bt.GetRight(), level-1);
}
```

פעולה המקבלת עץ ורמה העץ level, ומחזירה את סכום הצמתים באותה רמה

```
public static int SumLevel(BinNode<int> bt, int level)
{
	if (bt == null)
		return 0;
	if (level == 0)
		return bt.GetValue();
	return SumLevel(bt.GetLeft(), level-1) + SumLevel(bt.GetRight(), level-1);
}
```

פעולות בוליאניות בעצים (כל הפעלות מתייחסות לעץ של מס' שלמים)

פעולה המקבלת צומת בעץ ומחזירה true אם הוא עלה, אחרת false.

```
public static bool IsLeaf (BinNode<int> bt)
{
	if (bt == null)
		return false;
	return (bt.GetLeft() == null && bt.GetRight() == null);
}
```

פעולה המקבלת עץ ומחזירה true אם כל הערכים בעץ חיוביים, או false אחרת.

```
public static bool AllPositive (BinNode<int> bt)
{
	if (bt == null)
		return true;
	return bt.GetValue() > 0 && AllPositive (bt.GetLeft()) && AllPositive (bt.GetRight());
}
```

פעולה המקבלת עץ ומחזירה true אם אין בנים יחידים בעץ, או false אחרת.

```
public static bool NoOnlySon (BinNode<int> bt)
{
	if (bt == null)
		return true;
	return (IsLeaf(bt) || (bt.HasLeft() && bt.HasRight())) &&
			NoOnlySon (bt.GetLeft()) && NoOnlySon (bt.GetRight());
}
```

פעולה המקבלת עץ וערך num, ומחזירה true אם num נמצא בעץ או false אחרת

```
public static bool IsExist(BinNode<int> bt, int num)
{
	if (bt == null)
		return false;
	if (bt.GetValue() == num)
		return true;
	return IsExist(bt.GetLeft(), num) || IsExist(bt.GetRight(), num);
}
```

פעולה המקבלת עץ לא ריק וערכי x ו-**y**, ומחזירה true אם קיים צומת עם ערך x, שהוא צאצא של צומת עם ערך y. ניתן להשתמש בפעולת עזר.

```
public static bool Descendant (BinNode<int> bt, int x, int y)
{
	if (bt == null)
		return false;
	if (bt.GetValue() == y)
		return IsExist(bt, x);
	return Descendant(bt.GetLeft(), x, y) || Descendant(bt.GetRight(), x, y);
}
```

תרגילים נוספים (כל הפעלות מתייחסות לעץ של מס' שלמים)

פעולה המקבלת עץ לא ריק ומחזירה את גובהו של העץ. ניתן להשתמש בפעולות עזר.

```
public static int High (BinNode<int> bt)
{
	if (bt == null))
		return -1;
	int h1 = 0, h2 = 0;
	if (bt.HasLeft())
		h1 = 1 + High(bt.GetLeft());
	if (bt.HasRight())
		h2 = 1 + High(bt.GetRight());
	return Math.Max(h1, h2);
}
```

פעולה המקבלת עץ וצומת t, הפעולה תחזיר את הרמה שבה נמצא הצומת. ניתן להשתמש בפעולות עזר.

```
public static int NodeLevel (BinNode<int> bt, BinNode<int> t)
{
	return FindLevel(bt, t, 0);
}
```

פעולת עזר המעבירה את הרמה הנוכחית כפרמטר (Current Level)

```
private static int FindLevel (BinNode<int> bt, BinNode<int> t, int currentLevel)
{
	if (bt == null)
		return -1;
	if (bt == t)
		return currentLevel;
	int left = FindLevel (bt.GetLeft(), t, currentLevel + 1);
	if (left != -1)
		return left;
	return FindLevel (bt.GetRight(), t, currentLevel + 1);
}
```

C# – 271 פעולות עזר לבגרות

פעולה המקבלת עץ ומחזירה את הערך המקסימלי בעץ. ניתן להשתמש בפעולת עזר.
(לשים לב שבמבחן צריך לכתוב את הפעולה IsLeaf אם רוצים להשתמש בה. היא לא חלק מהממשק)

```
public static int GetMax (BinNode<int> bt)
{
        if (IsLeaf(bt))
                return bt.GetValue();
        if (bt.HasLeft() && bt.HasRight())
                return Math.Max(bt.GetValue(), Math.Max(GetMax(bt.GetLeft()), GetMax(bt.GetRight())));
        if (bt.HasLeft())
                return Math.Max(bt.GetValue(), GetMax(bt.GetLeft());
        return Math.Max(bt.GetValue(), GetMax(bt.GetRight());
}
```

פעולה שמקבלת שני עצים t1, t2 של מספרים שלמים – הפעולה תחזיר true אם כל ערכי הצמתים של t2 נמצאים ב-t1.
אחרת יוחזר false.

(לשים לב שבמבחן צריך לכתוב את הפעולה IsExist אם רוצים להשתמש בה. היא לא חלק מהממשק)

```
public static bool AreAllValuesIn (BinNode<int> t1, BinNode<int> t2)
{
    if (t2 == null)
            return true;
    if (!IsExist (t1, t2.GetValue()))
            return false;
    return AreAllValuesIn (t1, t2.GetLeft()) && AreAllValuesIn (t1, t2.GetRight());
}
```

סריקות עומק בעץ

סריקה תחילית – pre-order

```
public static void PreOrderTraversal (BinNode<T> bt)
{
        if bt != null)
        {
          Console.Write (bt.GetValue() + " ");
          PreOrderTraversal (bt.GetLeft());
          PreOrderTraversal (bt.GetRight());
        }
}
```

סריקה תוכית – in-order

```
public static void InOrderTraversal (BinNode<T> bt)
{
        if (bt != null)
        {
          InOrderTraversal (bt.GetLeft());
          Console.Write (bt.GetValue() + " ");
          InOrderTraversal (bt.GetRight());
        }
}
```

סריקה סופית – post-order

```
public static void PostOrderTraversal (BinNode<T> bt)
{
        if (bt != null)
        {
          PostOrderTraversal (bt.GetLeft());
          PostOrderTraversal (bt.GetRight());
          Console.Write (bt.GetValue() + " ");
        }
}
```