מחלקת התור לפי משרד החינוך:

```
class Queue<T>
  {
    private Node<T> first;
    private Node<T> last;
    public Queue()
    {
      this.first = null;
      this.last = null;
    }
    public void Insert(T value)
    {
      Node<T> newNode = new Node<T>(value);
      if (this.first == null)
          this.first = newNode;
       else
         this.last.SetNext(newNode);
      this.last = newNode;
    }
    public T Remove()
    {
      T value = this.first.GetValue();
      this.first = this.first.GetNext();
      if (this.first == null)
        this.last = null;
      return value;
    }
    public T Head()
    {
      return this.first.GetValue();
    }
```

```
    public bool IsEmpty()
    {
        return this.first == null;
    }

    public override string ToString()
    {
        return "" + this.first;
    }
}
```

הסבר כללי במבנה נתונים תורים הם מייצגים לנו תורים מהמציאות כמו למשל:
תור בבנק תור במאפייה וכו' העקרון בתורים הללו הוא : הראשון שנכנס הוא הראשון שיוצא
בעולם התכנות אנחנו קוראים לזה: First in First Out או בקצרה FiFo

לאחר שהבנו עקרון זה בוא ננתח את הפעולות הפנימיות שאפשר לבצע על התור עצמו –

שם הפעולה	פעולה בונה Queue()	Insert	Remove	Head	IsEmpty	ToString
תיאור הפעולה	פעולה המייצרת תור ריק	הפעולה מקבלת טיפוס מסוג התור ומכניסה אותו לסוף התור כחלק מעקרון ה FiFo	הפעולה מסירה את הטיפוס שנמצא בראש התור ומחזירה אותו למי שזימן את הפעולה	הפעולה מחזירה את הטיפוס שנמצא בראש התור מבלי להסיר אותו	הפעולה מחזירה אמת אם התור ריק במידה ולא הפעולה מחזירה שקר	הפעולה מחזירה מחרוזת המהווה תצוגה מחרוזתית לטיפוסי התור

דוגמאות לשימוש: Queue<T> - T הוא קיצור של Type– יכול לקבל כל סוג משתנה

יצירת תור חדש – עלינו להשתמש בפעולה הבונה לדוגמא:

```
Queue<int> myQ = new Queue<int>();
```

פעולה זו יוצרת לנו תור חדש

הוספת טיפוס חדש לתור – Insert

```
myQ.Insert(5);
myQ.Insert(3);
myQ.Insert(7);
```

הפעולה מכניסה את המספר 5 לאחריו את המספר 3 ולאחריו את המספר 7

הסרת איבר מראש התור והחזרתו – Remove

```
Int x = myQ.Remove();
```

בשלב הזה ערכו של x הוא 5

האיבר שנמצא בראש התור מבלי להסירו –

```
Int y = myQ.Head();
```

בשלב הזה ערכו של y הוא 3 וגם בראש התור עדיין קיים 3

בדיקה אם התור ריק –

```
Console.WrtieLine(myQ.IsEmpty());
```

יחזיר לי false מהסיבה שהתור לא ריק

ריצה על תורים באופן כללי: תזכורת FIFO

```
0 references
public static void p(Queue<int> q)
{
    while (!q.IsEmpty()) // שימו לב לסימן קריאה בהתחלה זה כמו לשאול אם אתה לא ריק
    {
        Console.WriteLine( q.Remove()); // מדפיס את ראש התור ומוחק אותו גורם לכך שכל פעם התור מצטמצם
    }
}
```

עקרון המשפך:

בשאלות בכיתה ובשאלות בגרות אתם תצטרכו "לשמור על מבנה התור הקיים" תוך כדי שתצטרכו לפתור את השאלה בקוד שלכם לכן קיימת שיטה שלה אנחנו קוראים – עקרון המשפך
השיטה באה לשמור את המקוריות של התור אך כן מאפשרת לנו לבצע עליו פעולות לפי הצורך , השיטה פעולת באופן התאורטי הבא: אנחנו "שופכים" את התור לתוך משתנה זמני ותוך כדי ניתן כבר לבצע את הפעולות שנרצה או לחלופין , מתי "שנשפוך" בחזרה את התור מהמשתנה הזמני לתור המקורי נוכל גם כן לבצע את הפעולות הנדרשות לדוגמא:

עלי לבצע פעולת מומצע לתור מספרים שלמים תוך שמירה על מבנה התור - נבצע את זה באופן הבא:

```
public static double avgQ(Queue<int> q)
{
    Queue<int> temp = new Queue<int>();
    int cnt = 0, sum = 0;
    while (!q.IsEmpty())// שופכים את התור המקורי לתור הזמני
    {
        cnt++; // סופר כמה איברים יש בתור
        sum += q.Head();// שולף את ראש התור כדי לסכום
        temp.Insert(q.Remove());  // מעביר את ראש התור לתור החדש בהתאם
    }
    while (!temp.IsEmpty())
    {
        q.Insert(temp.Remove()); // מחזיר בחזרה מהתור הזמני לתור המקורי
    }
    return (double )sum / cnt;
}
```