Házi Feladat

A feladat megoldását a Program.cs fájlba készítse el, melyet beadás előtt nevezzen át. A beadandó forrásfájl elnevezése a feladat azonosítója és a saját neptunkódja legyen alulvonással elválasztva, nagybetűkkel: **AZONOSÍTÓ_NEPTUNKOD.cs**

A feladattal kapcsolatos további információk az utolsó oldalon találhatók (ezen ismeretek hiányából adódó reklamációt nem fogadunk el!).

Szabadúszóként újabb feladatot vállal el, ahol egy olyan algoritmust kell készítenie, ami hatékonyan tudja dekódolni a mátrixban eltárolt üzenetet.

A mátrix mérete $M \times N$-es, minden cellája egy (T_i) karaktert tartalmaz, bejárni pedig az alábbi minta szerint kell, hogy megkapja az eredeti (T) üzenetet (*S az elsőnek, E az utolsónak kiolvasandó karakter helye, az olvasás haladási irányát a nyilak jelölik*):

például, ha M=4, N=6
→→→→→↓
↑→→→↓↓
↑↑E←←↓
S↑←←←←

például, ha M=3, N=9
→→→→→→→→↓
↑→→→→→→E↓
S↑←←←←←←←

például, ha M=7, N=5
→→→→↓
↑→→↓↓
↑↑E↓↓
↑↑↑↓↓
↑↑↑↓↓
↑↑↑←↓
S↑←←←

A mátrixban mindig a bal alsó sarokból kell indulnia és körkörösen befele haladva kell kiolvasni a karaktereket, míg nem ér eli az utolsó karaktert (*melynek pontos helye a mátrix méretétől függ*). Vagyis a bal alsó sarokból felfelé kell menni egészen a mátrix tetejéig, ahonnan jobbra kell tovább haladni a mátrix jobb széléig, amit követően az aljáig lefelé kell kiolvasni a karaktereket. Ezután egészen a kezdőpont előtti karakterig kell haladni majd felfelé és így tovább egyre beljebb a mátrixban.

Bemenet (Console)

- *első sor* - a mátrix sorainak (M) és oszlopaink (N) száma szóközzel elválasztva
- *további M sor* - a mátrix egy sorának tartalma, ami pontosan N karakterből áll

Kimenet (Console)

- a `Console`-on megjeleníve a mátrixból kiolvasott üzenet

Megkötés(ek)

- $1 \leq M, N \leq 100$
- minden karakter a T-ben, $T_i \in \{0-9, a-z, A-Z, -\_.?!, \text{vessző}\}$

Példa

Console input
```
3 4
ABCD
EFGH
IJKL
```

Console output
```
IEABCDHLKJFG
```

Értelmezés

Az első sor alapján (`M=3`, `N=4`) a felhasználó egy 3×4-as mátrixban tárolt üzenetet szeretne dekódolni. Az első sor első értéke alapján 3 sort kell még beolvasni, ami mind 4 karaktert fog tartalmazni.

A beolvasott mátrixot az alábbi minta szerint kell bejárni (*a bal alsó sarokból indulva, körkörösen befele haladva*):

ABCD →→→↓
EFGH ↑→E↓
IJKL S↑←←

A bal alsó sarokból a tetejéig kiolvassuk a `IEA` karaktereket, amit követően jobbra fordulunk és felül kiolvassuk a `BCD` karaktereket, majd lefele haladva az aljáig kiolvassuk a `HL` karaktereket, majd balra az első karakter előtti karakterig a `KJ` karaktereket olvassuk ki. Ezután felfele olvasunk ki immár csak egy karaktert az `F`-et, amt követően jobbra kiolvassuk az utolsó karaktert a `G`-t.

A kiolvasott karaktereket egymás után fűzve megjelenítjük a felhasználónak a dekódolt üzenetet: `IEABCDHLKJFG`.

Tesztesetek

Az alkalmazás helyes működését legalább az alábbi bemenetekkel tesztelje le!

1.

Console input
```
3 4
ABCD
EFGH
IJKL
```

Console output
```
IEABCDHLKJFG
```

2.

Console input
```
1 1
x
```

Console output
```
x
```

3.

Console input
```
4 9
ABCDEFGHI
JKLMNOPQR
STUVWXYZ0
123456789
```

Console output
```
1SJABCDEFGHIR098765432TKLMNOPQZYXWVU
```

4.

Console input
```
7 6
or_not
_s_th_
eionet
b_i._o
_tt.q_
oaseub
Tht_,e
```

Console output
```
To_be_or_not_to_be,_that_is_the_question..
```

5.

Console input
```
3 8
do not f
 uters.e
Ipmoc ra
```

Console output
```
I do not fear computers.
```

6.

Console input
```
6 11
omputer was
coblems th
 rexist bab
ep .erofeto
h ton did r
Tevlos ot n
```

Console output
```
The computer was born to solve problems that did not exist before.
```

Tájékoztató

A feladattal kapcsolatosan általános szabályok:

- A feladat megoldását egy Console Application részeként kell elkészíteni.
- A feladat megoldásaként beadni vagy a betömörített solution mappa egészét vagy a `Program.cs` forrásfájlt kell (hogy pontosan melyiket, azt minden feladat külön definiálja), melynek elnevezése a feladat azonosítója és a saját neptunkódja legyen alulvonással elválasztva, nagybetűkkel:
  AZONOSÍTÓ_NEPTUNKOD[.zip|.cs]
- A megvalósítás során lehetőség szerint alkalmazza az előadáson és a laboron ismertetett programozási tételeket és egyéb algoritmusokat figyelembe véve a *Megkötések* pontban definiáltakat, ezeket leszámítva viszont legyen kreatív a feladat megoldásával kapcsolatban.
- Az alkalmazás elkészítése során minden esetben törekedjen a megfelelő típusok használatára, illetve az igényes (*formázott, felesleges változóktól, utasításoktól mentes*) kód kialakítására, mely magába foglalja az elnevezésekkel kapcsolatos ajánlások betartását is (bővebben).
- **Ne másoljon vagy adja be más megoldását!** Minden ilyen esetben az összes (felépítésben) azonos megoldás duplikátumként lesz megjelölve és a megoldás el lesz utasítva.
- **Idő után leadott vagy helytelen elnevezésű megoldás vagy a kiírásnak nem megfelelő megoldás vagy fordítási hibát tartalmazó vagy (helyes bemenetet megadva) futásidejű hibával leálló kód nem értékelhető!**
- A feladat leírása az alábbiak szerint épül fel (* - opcionális):
  - *Feladat leírása* - a feladat megfogalmazása
  - *Bemenet* - a bemenettel kapcsolatos információk
  - *Kimenet* - az elvárt kimenettel kapcsolatos információk
  - *Megkötések* - a bemenettel, a kimenettel és az algoritmussal kapcsolatos megkötések, melyek figyelembevétele és betartása kötelező, továbbá az itt megfogalmazott bemeneti korlátoknak a tesztek minden eseteben eleget tesznek, így olyan esetekre nem kell felkészülni, amik itt nincsenek definiálva
  - **Megjegyzések* - további, a feladattal, vagy a megvalósítással kapcsolatos megjegyzések
  - *Példa* - egy példa a feladat megértéséhez
  - *Tesztesetek* - további tesztesetek az algoritmus helyes működésének teszteléséhez, mely nem feltétlenül tartalmazza az összes lehetséges állapotát a be- és kimenet(ek)nek
- **Minden eseteben pontosan azt írja ki és olvassa be az alkalmazás, amit a feladat megkövetel, mivel a megoldás kiértékelése automatikusan történik!** Így például, ha az alkalmazás azzal indul, hogy kiírja a konzolra a `"Kérem a számot:"` üzenetet, akkor a kiértékelés sikertelen lesz, a megoldás hibásnak lesz megjelölve, ugyanis egy számot kellett volna beolvasni a kiírás helyett.
- A kiértékelés során csak a *Megkötések* pont szerinti helyes bemenettel lesz tesztelve az alkalmazás, a "tartományokon" kívüli értéket nem kell lekezelnie az alkalmazásnak.
- Elősegítve a fejlesztést, a beadott megoldás utolsó utasításaként szerepelhet egyetlen `Console.ReadLine()` metódushívás.
- A kiértékelés automatikusan történik, így különösen fontos a megfelelő alkalmazás elkészítése, ugyanis amennyiben nem a leírtaknak megfelelően készül el a megoldás úgy kiértékelése sikertelen lesz, a megoldás pedig hibás.
- Az automatikus kiértékelés négy részből áll:
  - Unit Test-ek - az alkalmazás futásidejű működésének vizsgálatára
  - Szintaktikai ellenőrzés - az alkalmazás felépítésének vizsgálatára
  - Duplikációk keresése - az azonos megoldások kiszűrésére
  - Metrikák meghatározása - tájékoztató jelleggel
- A kiértékelések eredményéből egy HTML report generálódik, melyet minden hallgató megismerhet.
- A leadott megoldással kapcsolatos minimális elvárás:
  - Nem tartalmazhat fordítás idejű figyelmeztetést (`Solution contains 0 compile time warning(s)`).
  - Nem tartalmazhat fordítási hibát (`Solution contains 0 compile time error(s)`).
  - Minden szintaktikai tesztet teljesít (`0 test warning, 0 test failed`).
  - Minden unit test-et teljesít (`0 test failed, 0 optional test failed, 0 test was not run`).

- A feladat megoldásához minden esetben elegendő a ***.NET Framework 4.7.2***, illetve a ***C# 7.3***, azonban megoldását elkészítheti ***.NET 5***-öt, illetve a ***C# 9***-et használva is, viszont a nyelv újjításait nem használhatja. További általános, nyelvi elemekkel való megkötés, melyet a házi feladatok során nem használhat a megoldásában (*a felsorolás változásának jogát fenntartjuk, a mindig aktuális állapotot a report HTML fogja tartalmazni*):
  - Methods: `Array.Sort`, `Array.Reverse`, `Console.ReadKey`, `Environment.Exit`
  - LINQ: `System.Linq`
  - Attributes
  - Collections: `ArrayList`, `BitArray`, `DictionaryEntry`, `Hashtable`, `Queue`, `SortedList`, `Stack`
  - Generic collections: `Dictionary<K,V>`, `HashSet<T>`, `List<T>`, `SortedList<T>`, `Stack<T>`, `Queue<T>`
  - Keywords:
    - Modifiers: `protected`, `internal`, `abstract`, `async`, `event`, `external`, `in`, `out`, `sealed`, `unsafe`, `virtual`, `volatile`
    - Method parameters: `params`, `in`, `out`
    - Generic type constraint: `where`
    - Access: `base`
    - Contextual: `partial`, `when`, `add`, `remove`, `init`
    - Statement: `checked`, `unchecked`, `try-catch-finally`, `throw`, `fixed`, `foreach`, `continue`, `goto`, `yield`, `lock`, `break` - *in loop*
    - Operator and Expression:
      - Member access: `^` - *index from end*, `..` - *range*
      - Type-testing: `is`, `as`, `typeof`
      - Conversion: `implicit`, `explicit`
      - Pointer: `*` - *pointer*, `&` - *address-of*, `*` - *pointer indirection*, `->` - *member access*
      - Lambda: `=>` - *expression, statement*
      - Others: `?:` - *tenary*, `!` - *null forgiving*, `?.` - *null conditional member access*, `?[]` - *null conditional element access*, `??` - *null coalescing*, `??=` - *null coalescing assignment*, `::` - *namespace alias qualifier*, `await`, `default` - *operator, literal*, `delegate`, `is` - *pattern matching*, `nameof`, `sizeof`, `stackalloc`, `switch`, `with` - *expressiong*, `operator`
    - Types: `dynamic`, `interface`, `object`, `Object`, `var`, `struct`, `nullable`, `pointer`, `record`, `Tuple`, `Func<T>`, `Action<T>`,