



A feladat megoldása a Program.cs fájl legyen, 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 találhatóak (ezen ismeretek hiányából adódó reklamációt nem fogadunk el!).

Írjon programot, amely kirajzol egy üres négyzetet a konzolra. A négyzet kirajzolásához használt karaktert (C) és a négyzet oldalhosszát (N) a felhasználó adja meg.

Bemenet (Console)

- a négyzet oldalhossza, az N
- a kirajzolásához használt karakter, a C

Kimenet (Console)

- az N oldalhosszú négyzet körvonala megjelenítve a C karakterrel

Megkötés(ek)

- $1 \leq N \leq 50$
- $C \in \{0 - 9a - zA - Z * - +\}$

Példa

Console input

4

+

Console output

++++

+ +

+ +

++++

Értelmezés

A felhasználó + karakterrel szeretne egy 4-es oldalhosszú négyzetet kirajzolni. A négyzet közepe "üres" vagyis szóköz karakterekkel kerül kitöltésre. Az alkalmazás a felhasználó bemenete alapján ($N = 4$, $C = +$) az alábbi írja ki a konzolra:

++++

+ +

+ +

++++

Tesztesetek

Az alkalmazás helyes működését legalább az alábbi bemenetekkel tesztelje le!

1.

Console input

4

+

Console output

++++

+ +

+ +

++++

2.

Console input

2

a

Console output

aa

aa

3.

Console input

1

o

Console output

o

**4.**

Console input

7

p

Console output

ppppppp

p p

p p

p p

p p

p p

ppppppp

A fenti tesztesetek nem feltétlenül tartalmazzák az összes lehetséges állapotát a be- és kimenet(ek)nek, így saját tesztekkel is próbálja ki az alkalmazás helyes működését!

Tájékoztató

A feladattal kapcsolatosan általános szabályok:

- A feladat megoldását egy Console App részeként kell elkészíteni a "top-level statements" mellőzése, illetve az importok megtartása mellett.
- A feladat megoldásaként beadni a Program.cs forrásfájlt kell, melynek elnevezése a feladat azonosítója és a saját neptunkódja legyen alulvonással elválasztva, nagybetűkkel: **AZONOSÍTÓ_NEPTUNKOD.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 figyelembevételre és betartásra kötelező, továbbá az itt megfogalmazott bemeneti korlátoknak a tesztek minden esetében 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 esetben 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.
- 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 test warning, 0 test was not run`).
- A feladat megoldásának minden esetben fordíthatónak és futtathatónak kell lennie a **.NET 6** keretrendszer felett **C# 10**-ben. Ettől függetlenül az elkészítés során használható egyéb változata a .NET keretrendszernek és a C# nyelvnek, azonban leadás előtt győződjön meg róla, hogy a



megoldása kompatibilis a .NET 6 és C# 10 verzióval.

- A keretrendszer mellett 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: `?:` - *ternary*, `!` - *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` - *expression, operator*
 - Types: `dynamic`, `interface`, `object`, `Object`, `var`, `struct`, `nullable`, `pointer`, `record`, `Tuple`, `Func<T>`, `Action<T>`,