Házi Feladat

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 találhatók (ezen ismeretek hiányából adódó reklamációt nem fogadunk el!).

Írjon programot, amely a beolvasott analóg feszültség szintek sorozatát (S) feldolgozza majd megjeleníti a kimeneten a felhasznált logikai kapu (L) által adott kimenetet (0 vagy 1) minden időpillanatban.

A bemeneti analóg feszültségszint egy adott időpillanatban alacsony szintűnek, vagyis 0-nak tekinthető, ha az érték 0 V és 0.8 V között található (inkluzív módon, tehát az intervallum határait is beleértve). A feszültség magas szintűnek, vagyis 1-nek tekinthető, ha az érték 2.7 V és 5 V között található (inkluzív módon). Az előbbiek közé eső érték érvénytelen jelszintként értelmezhető, ilyen esetben a kimeneten egy E karakternek kell megjelennie, függetlenül a felhasznált kapu fajtájától. Egy analóg jelsorozat t-edik időpillanatbeli értéke két számjeggyel van kódolva (*például* 32), amelynek első számjegye az egész rész értéket, míg második számjegye a tizedes rész értéket képviseli (*vagyis a* 32 *szám* 3.2 *V-ként értelmezendő*).

Az alábbi logikai kapuk lehetségesek:

- **AND** - a bemenetek logikai *ÉS* kapcsolatát valósítja meg
- **OR** - a bemenetek logikai *VAGY* kapcsolatát valósítja meg
- **NOT** - a bemenet értékét ellenkezőjére változtatja
- **NAND** - a bemenetek logikai *NEM ÉS* kapcsolatát valósítja meg
- **NOR** - a bemenetek logikai *NEM VAGY* kapcsolatát valósítja meg
- **XOR** - a bemenetek logikai *KIZÁRÓ VAGY* kapcsolatát valósítja meg
- **XNOR** - a bemenetek logikai *NEM KIZÁRÓ VAGY* kapcsolatát valósítja meg

A lehetséges logikai kapuk igazságtáblája:

A	**B**	**A AND B**	**A OR B**	**NOT A**	**A NAND B**	**A NOR B**	**A XOR B**	**A XNOR B**
0	**0**	0	0	1	1	1	0	1
0	**1**	0	1	1	1	0	1	0
1	**0**	0	1	0	1	0	1	0
1	**1**	1	1	0	0	0	0	1

Bemenet (Console)

- első sor, a logikai kapu neve, az első, majd ha szükséges, a második bemenetére adott jelsorozat neve szóközzel elválasztva
- további N sor, soronként: a jelsorozat neve, majd egy szóközt követően a hozzá tartozó analóg feszültségszintek sorozata elválasztás nélkül

Kimenet (Console)

- a logikai kapu által adott jelszintek egyetlen sorban, elválasztás nélkül

Megkötés(ek)

- $L \in \{AND, OR, NOT, NAND, NOR, XOR, XNOR\}$
- $1 \leq |N| \leq 26$
- S_i neve $\in \{A - Z\}$
- $1 \leq |S_i$ számpárok$| \leq 1\ 000$
- $00 \leq S_i$ jelszint≤ 50
- S_i minden esetben pontosan kettő értékkel van kódolva
- minden S jelsorozat pontosan ugyan annyi értéket tartalmaz
- az S jelsozoratok nem feltétlenül ábécé sorrendben vannak, de a kapu működéséhez szükséges jelsorozatok biztos megtalálhatók a bemenetek között

Megjegyzés

- amennyiben szükséges, és a `double.Parse` metódust használná, úgy vagy használja nem egész számok esetén a tizedes vesszőt, vagy állítsa be a lebegőpontos szám formátumát

Példa

Console input
```
AND A C
C 0205094150
B 0001020304
A 0028033517
```

Console output
```
00E1E
```

Értelmezés

Az első sor (`AND A C`) alapján az *A* és a *C* analóg feszültségszintek *ÉS* kapcsolatát kell megjeleníteni a kimeneten.

Az `A` analóg feszültségszintek az alábbi lebegőpontos számmá alakított értékeket és az általuk képviselt logikai szinteket tartalmazza: $0.0 \rightarrow \mathbf{0}$, $2.8 \rightarrow \mathbf{1}$, $0.3 \rightarrow \mathbf{0}$, $3.5 \rightarrow \mathbf{1}$, $1.7 \rightarrow \mathbf{E}$.

A `C` analóg feszültségszintek az alábbi lebegőpontos számmá alakított értékeket és az általuk képviselt logikai szinteket tartalmazza: $0.2 \rightarrow \mathbf{0}$, $0.5 \rightarrow \mathbf{0}$, $0.9 \rightarrow \mathbf{E}$, $4.1 \rightarrow \mathbf{1}$, $5.0 \rightarrow \mathbf{1}$.

Az *ÉS* kapu csak akkor 1 a kimeneten, ha mindkét bemenete 1. Ennek értelmében, illetve figyelembevéve az érvénytelen jelszinteket, az alkalmazás kimeneten a `00E1E` sorozatot jelenít meg.

Tesztesetek

Az alkalmazás helyes működését legalább az alábbi bemenetekkel tesztelje le!

1.

Console input
```
AND A C
C 0205094150
B 0001020304
A 0028033517
```

Console output
```
00E1E
```

2.

Console input
```
OR A C
Z 0001020304
C 0205094150
A 0028033517
```

Console output
```
01E1E
```

3.

Console input
```
NOT A
A 0028033517
```

Console output
```
1010E
```

4.

Console input
```
NAND A C
A 0028033517
B 0001020304
C 0205094150
```

Console output
```
11E0E
```

5.

Console input
```
NOR A B
A 0028033517
B 0001020304
```

Console output
```
1010E
```

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 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.
- Az automatikus kiértékelés négy részből áll:
  - Unit Test-ek - az alkalmazás futásidejű működésének vizsgálatára
  - Szintaktikai ellenőrzés - az alkalmazás felépítésének vizsgálatára
  - Tervezési irányelvek ellenőrzése - az alkalmazás "kinézetének" vizsgálatára
  - Duplikációk keresése - az azonos megoldások kiszűrésére
  - Metrikák meghatározása - tájékoztató jelleggel
- A kiértékelések eredményéből egy HTML report generálódik, melyet minden hallgató megismerhet.
- A feladat megoldásának minden esetben fordíthatónak és futtathatónak kell lennie ***C# 10*** és ***.NET 6*** keretrendszer használatával. 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, melyek a házi feladatok során nem használhatók a megoldásában:
  - LINQ: `System.Linq` - *all query expressions within the namespace*
  - Attributes
  - Collections: `ArrayList`, `BitArray`, `DictionaryEntry`, `Hashtable`, `Queue`, `SortedList`, `Stack`
  - Keywords:
    - Modifiers: `abstract`, `async`, `event`, `external`, `in` - *generic modifier*, `new` - *member modifier*, `out` - *generic modifier*, `sealed`, `unsafe`, `virtual`, `volatile`
    - Statement: `break` - *in a loop*, `continue`, `goto`, `throw`, `try-catch-finally`, `checked`, `unchecked`, `fixed`, `lock`, `yield`
    - Namespace: `extern alias`
    - Generic type constraint: `new`, `where`
    - Access: `base`
    - Contextual: `add`, `partial` - *type, method*, `remove`, `required`, `when` - *filter condition*,
    - Query: `from`, `where`, `select`, `group`, `into`, `orderby`, `join`, `let`, `in`, `out`, `equals`, `by`, `ascending`, `descending`
  - Operators and Expressions:
    - Null-conditional operators: `?.` - *null conditional member access*, `?[]` - *null conditional element access*
    - User-defined conversion operators: `implicit`, `explicit`
    - Pointer: `*` - *pointer*, `&` - *address-of*, `*` - *pointer indirection*, `->` - *pointer member access*, `[]` - *pointer element access*, `+`, `-`, `++`, `--` - *pointer arithmetic operators*
    - Assignment: `ref`
    - Lambda: `=>` - *expression, statement*
    - Others: `!` - *null forgiving*, `??` - *null coalescing*, `??=` - *null coalescing assignment*, `::` - *namespace alias qualifier*, `await`, `default` - *operator, literal*, `delegate`, `sizeof`, `stackalloc`, `with` - *expression*, `operator`
  - Types: `dynamic`, `interface`, `object`, `Object`, `var`, `struct`, `nullable`, `pointer`, `record`, `Tuple`, `Func<T>`, `Action<T>`, `Expression<T>`, `Nullable<T>`, `Span<T>`
  - Preprocessor directives: `#nullable`, `#if`, `#elif`, `#else`, `#endif`, `#define`, `#undefine`, `#undef`, `#error`, `#warning`, `#line`, `#pragma`
- Névterek, melyek kizárólagosan importálhatók a megoldásban (minden további névtér import törlésre kerül, illetve, ha az alábbiak közül valamelyik hiányzik, akkor hozzáadásra kerül a megoldáshoz):
  - `System`
  - `System.Collections.Generic`
  - `System.IO`
  - `System.Threading`