



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 interpretert az alábbi egyszerűsített assembly nyelvhez, amely 4 regisztert ( $A, B, C, D$ ) és 4 utasítást (MOV, ADD, SUB, JNE) tartalmaz. Dolgozza fel az utasításokat, majd írja fájlba a négy regiszter tartalmát.

A nyelv utasításai:

- **MOV DEST SRC/CNST** - az *SRC* regiszter vagy a *CNST* értéket a *DEST* registerbe másolja. Például: MOV A 1 utasítás után az A regiszter értéke 1 lesz.
- **ADD DEST SRC/CNST SRC/CNST** - összeadja az *SRC* regiszterek tartalmát vagy a *CNST* értékeket és eltárolja az eredményt a *DEST* regiszterben. Például: ADD A B 7 utasítás után az A regiszter értéke  $B + 7$  lesz.
- **SUB DEST SRC/CNST SRC/CNST** - kivonja a harmadik *SRC* regiszter tartalmát vagy *CNST* értéket a második *SRC* regiszter tartalmából vagy *CNST* értékéből és eltárolja az eredményt a *DEST* regiszterben. Például: SUB A 9 7 utasítás után az A regiszter értéke  $9 - 7 = 2$  lesz.
- **JNE CNST SRC SRC/CNST** - a *CNST*. utasításra ugrik, ha a második és a harmadik érték nem egyezik. Például: JNE 0 A 1 utasítás után a 0. sorban folytatódik a feldolgozás, ha az A regiszter tartalmaz nem egyenlő 1-el.

### Bemenet (File)

- egyetlen fájl, aminek a neve: input.txt
- az első sor a négy regiszter értékét tartalmazza vesszővel elválasztva
- a további  $N$  sorban pedig a végrehajtandó utasítások

### Kimenet (File)

- egyetlen fájl, aminek a neve: output.txt
- a fájl pontosan egyetlen sort tartalmaz, amelynek értéke a négy regiszter tartalma, vesszővel elválasztva

### Megkötés(ek)

- $1 \leq N \leq 100$
- $-2^{15} \leq A, B, C, D \leq 2^{15}$

### Megjegyzés

- Az utasítások és az operandusok minden esetben egyetlen szóközzel vannak elválasztva.
- A program az első utasítással indul, végighalad az összes utasításon, és akkor ér véget, amikor az utolsó utasítást is feldolgozta.
- Minden program érvényes, nem tartalmaz végtelen ciklust, túlsordulást, így ezeket nem kell lekezelni.
- A program első utasításának indexe 0.

### Példa

input.txt

```
1, 3, 5, 8
SUB B B 1
MOV C B
ADD C C 1
JNE 2 C 4
```

output.txt

```
1, 2, 4, 8
```

**Értelmezés**

Az első sor alapján beállításra kerülnek a regiszterek (A=1, B=3, C=5, D=8), amit a program utasításainak feldolgozása követ:

0. utasítás - a B regiszter értékéből ki kell vonni 1-et és el kell tárolni a B regiszterben

[A=1, B=2, C=5, D=8]

1. utasítás - a B regiszter tartalmát a C regiszterbe kell másolni

[A=1, B=2, C=2, D=8]

2. utasítás - a C regiszter tartalmához hozzá kell adni 1-et és el kell tárolni a C regiszterben

[A=1, B=2, C=3, D=8]

3. utasítás - a C regiszter tartalmát össze kell hasonlítani 4-el, és mivel ez nem egyezik, a 2. utasításra kell ugrani

2. utasítás - a C regiszter tartalmához hozzá kell adni 1-et és el kell tárolni a C regiszterben

[A=1, B=2, C=4, D=8]

3. utasítás - a C regiszter tartalmát össze kell hasonlítani 4-el, és mivel ez egyezik, a program futása leáll

A program futását követően a regiszterek tartalma, amit fájlba kell írni vesszővel elválasztva rendre a következők: 1, 2, 4, 8

**Tesztesetek**

Az alkalmazás helyes működését legalább az alábbi bemenetekkel tesztelje le!

1. input.txt 1, 3, 5, 8 SUB B B 1 MOV C B ADD C C 1 JNE 2 C 4	output.txt 1, 2, 4, 8
2. input.txt -1, -2, -3, 3 MOV B 3 MOV A C	output.txt -3, 3, -3, 3
3. input.txt 2, 3, 4, 5 ADD A B 1 ADD B 2 6 ADD C A B ADD D B C	output.txt 4, 8, 12, 20
4. input.txt 14, 2, 21, 9 SUB A A A SUB D 12 A SUB B 15 3 SUB C C 9	output.txt 0, 12, 12, 12
5. input.txt 1, 1, 1, 10 SUB D D 1 JNE 0 D 1	output.txt 1, 1, 1, 1, 1

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>`,