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!).

Az alábbi, `eredeti`ként megjelölt kódrészlet vajon hány különböző kimenetet tud produkálni?

eredeti
```
begin
  if (C1) then
    A1
  else
    A2
  endif

  if (C1) then
    A3
  else
    A4
  endif
end
```

normalizált
```
begin
if
else
endif
if
else
endif
end
```

A helyes válasz a **4**, ami a két feltétel (`C1` és a `C2`) variációi alapján a következők lehetnek:

`A1, A3` vagy `A1, A4` vagy `A2, A3` vagy `A2, A4`

A feladatunk az lesz, hogy a fájlból beolvasott függvény (F) logikai felépítését kiértékeljük és meghatározzuk, hogy hány variációja állhat elő a feltételeknek. A kiértékelés során nem fog érdekelni minket, hogy mi a tényleges feltétel, illetve, hogy mi fog történni az akciók során.

A kiértékelés megkönnyítése érdekében normalizáljuk a függvény szintaxisát az alábbiak szerint:

- Az "`if..then`" kifejezést "`if`"-ként írjuk le és feltételezzük, hogy mindig van egy rejtett, egyedi feltétel az "`if`" kulcsszót követően.
- Továbbá, az "`if`" utasítást tartalmazó sor után mindig van (nem feltétlenül azt követően) pontosan egy hozzá tartozó "`else`" utasítást tartalmazó sor. Az "`if..else`" kifejezést egy "`endif`" utasítást tartalmazó sor fogja lezárni a szerkezet teljességének érdekében.
- Tehát egy sor az alábbiak közül egyetlen elemet tartalmazhat csak: `if`, `else`, `endif`.
- Előfordulhatnak egymásba ágyazott feltételek.
- A feltételeket követő utasítás(oka)t töröltük, de feltételezzük, hogy mindig van egy utasítás ami lefutna.
- A függvény a "`begin`" és az "`end`" között szerepel.
- A sorok nincsenek behúzva, illetve üres sorokat nem tartalmaz a függvény.

A fenti, `eredeti`ként megjelölt kódrészlet normalizált változata mellette látható.

Egy másik példa beágyazott feltételekre, ami 3 különböző kimenetet tud adni:

eredeti
```
begin
  if (C1) then
    A1
  else
    if (C2) then
      A2
    else
      A3
    endif
  endif
end
```

normalizált
```
begin
if
else
if
else
endif
endif
end
```

Bemenet (Fájl)

- egyetlen fájl, aminek a neve: `input.txt`
- a fájl felépítése:
  - *1. sor* - a fájlban lévő további sorok száma (N), ami az F függvényt tartalmazza
  - *további N sor* - a feldolgozandó normalizált F függvény

Kimenet (Fájl)

- egyetlen fájl, aminek a neve: `output.txt`
- egyetlen sor, ami az F függvény feltételeinek lehetséges variációinak számát tartalmazza

Megkötés(ek)

- $2 \leq N \leq 100$
- F elemei $\in \{begin, end, if, else, endif\}$

Példa

input.txt
```
8
begin
if
else
if
else
endif
endif
end
```

output.txt
```
3
```

Értelmezés

A fájl első sora alapján további 8 sort kell beolvasni, ami `begin-end` között fogja tartalmazni a feldolgozandó függvényt. A függvény két darab `if-else-endif` blokkot tartalmaz egymásba ágyazva, úgy, hogy a külső blokk `else` ágába került a második blokk.

Formázott
```
if
else
    if
    else
    endif
endif
```

1. varióáció
```
if          <<
else
    if
    else
    endif
endif
```

2. varióáció
```
if
else        <<
    if      <<
    else
    endif
endif
```

3. varióáció
```
if
else        <<
    if
    else    <<
    endif
endif
```

Az első lehetőség, mikor a külső elágazás `if` ágába fut a vezérlés. Második lehetőség, mikor a külső elágazás `else` ágában található `if` ágba fut a vezérlés, míg a harmadik lehetőség, ha az `else` ágon belüli `else` ágba fut a vezérlés. Ezek alapján a fájlba írandó szám a `3`.

Tesztesetek

Az alkalmazás helyes működését legalább az alábbi bemenetekkel tesztelje le!

1.

input.txt
```
8
begin
if
else
endif
if
else
endif
end
```

output.txt
```
4
```

2.

input.txt
```
8
begin
if
else
if
else
endif
endif
end
```

output.txt
```
3
```

3.

input.txt
```
2
begin
end
```

output.txt
```
1
```

4.

input.txt
```
23
begin
if
if
else
endif
if
else
endif
if
else
endif
else
if
if
else
endif
if
else
endif
else
endif
endif
end
```

output.txt
```
13
```

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 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 test warning, 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>`,