Órai feladat – Bináris keresőfa

Bevezetés

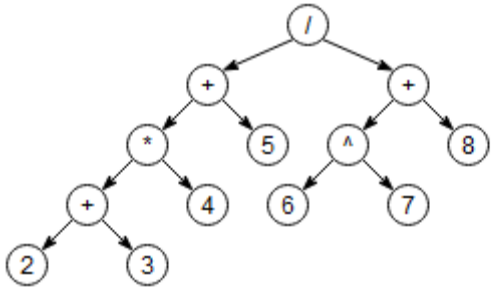
A numerikus kifejezések kiértékelésének állandó problémája a zárójelezés és az operátor precedencia kezelése, ugyanis a kiértékelés elkezdéséhez először meg kell keresni a legbelső zárójelek közé zárt részkifejezést, majd a legmagasabb precedenciájú operátorokat, végül a kötési iránynak megfelelően az első kiértékelendő operátort.

Ennek a problémának egy megoldása, ha a kifejezésből bináris kifejezésfát készítünk, ahol a fa levelei az operandusok (esetünkben konstansok: {`0-9`}), a további csomópontjai az operátorok (esetünkben szigorúan bináris operátorok: {`+`, `-`, `*`, `/`, `^`} így biztosítva, hogy az egyes csomópontoknak maximum két gyermeke lehet, ezzel alkalmassá téve a bináris keresőfát a probléma megoldására).

A felépített kifejezésfában tárolt kifejezés kiértékelése során a levél elemektől kell felfele haladni a fa gyökere felé, miközben olyan részfákat keresünk, melyek csak konstansokat és egyetlen operátort tartalmaznak. Ekkor alkalmazni kell az operátort a két konstansra, majd megtartva a kiszámított értéket az operátor csomópontjában, további részfákat keresünk. Ezt a lépéssorozatot kell addig ismételni, míg a fában a gyökér elemen szereplő operátor is helyettesítésre kerül, így megkapva a kifejezés végeredményét.

A numerikus kifejezések bonyolultságának csökkentésére, vagyis elsődlegesen a zárójelezések mellőzésére a fát nem a „hagyományos” kifejezésből kell felépíteni, hanem annak *Fordított Lengyel Jelöléséből* (RPN – Reverse Polish Notation). A fordított lengyel jelölés egy postfix jelölési forma, ami mellőzi a zárójeleket a kifejezésben, illetve megakadályozza a memória túlterhelődését a műveletek elvégzése közben, továbbá egyértelműen meghatározza a kiértékelés sorrendjét. A postfix jelöléssel például az `a + b` összeadást `ab+` módon ábrázolható. További példák:

Kifejezés	*Fordított Lengyel Jelölés*
`2 + 3 * 4`	`234*+`
`2 * 3 + 4`	`23*4+`
`2 * 3 + 4 * 5`	`23*45*+`
`(2 + 3) / (4 - 5)`	`23+45-/`
`((2 + 3) * 4 + 5) / (6 ^ 7 + 8)`	`23+4*5+67^8+/`



A `23+4*5+67^8+/` kifejezés bináris kifejezésfaként ábrázolva.

A feladat, hogy a fordított lengyel jelöléssel megadott kifejezésből építse fel a bináris kifejezésfát, majd a különböző fa bejárások segítségével lehessen az eredeti kifejezést, annak zárójelezett formáját, illetve a kiértékelés eredményét megjeleníteni. A fa, a hatékony feldolgozása érdekében legyen immutable[1], vagyis a felépítését követően csak olvasható.

[1] A csak olvashatóság garantálja, hogy az adatszerkezet bárminemű feldolgozása nem változtatja meg annak sem felépítését, sem tartalmát.

Mivel a leadott feladatokat lehetséges, hogy automatizált módon fogjuk ellenőrizni, ezért kérünk mindenkit, hogy a lenti (ékezetek nélküli) elnevezéseket tartsa meg. Szükség esetén további mezőket fel lehet venni, bár ezekre általában nincs szükség.
Ugyanígy kérünk mindenkit, hogy próbálja meg önállóan megoldani a feladatot, mivel csak így fog bármit tanulni belőle. Szükség esetén persze a laborvezetőket nyugodtan meg lehet keresni, akik segíteni fognak.

Feladatok

Bináris kifejezésfa elkészítése

Készíts egy **BinaryExpressionTree** nevű **osztály**t az alábbiak szerint:

- Az osztály csak azon tagjait jelöld publikus láthatósággal, ahol az külön jelezve van, minden további tag csak az osztályon belül legyen elérhető
- Legyen egy **Operator** nevű **felsorolás típus (enum)** az alábbi elemekkel:
  - **Add** – értéke az ASCII tábla `+` karakterének decimális értéke
  - **Sub** – értéke az ASCII tábla `-` karakterének decimális értéke
  - **Mul** – értéke az ASCII tábla `*` karakterének decimális értéke
  - **Div** – értéke az ASCII tábla `/` karakterének decimális értéke
  - **Pow** – értéke az ASCII tábla `^` karakterének decimális értéke
- Legyen egy **Node** nevű absztrakt **osztály** az alábbiak szerint:
  - **Data** – karakter típusú csak olvasható tulajdonság
  - **Left** – *Node* típusú csak olvasható tulajdonság
  - **Right** – *Node* típusú csak olvasható tulajdonság
  - egy konstruktor, amivel a *Data, Left, Right* tulajdonságok értékeit lehet beállítani
  - egy konstruktor, amivel csak a *Data* tulajdonság értékét lehet beállítani, ez a konstruktor a törzsébe ne tartalmazzon megvalósítást, hívja meg a másik konstruktort, ahol a *Left* és a *Right* értékeknek állítson be `null` értéket
- Legyen egy **OperandNode** nevű osztály, ami a **Node** osztály leszármazottja az alábbiak szerint:
  - egyetlen, egy paraméterrel rendelkező konstruktorral rendelkezik, mely az ős osztály azonos paraméterszámú konstruktorát hívja meg, átadva neki a paraméterül kapott értéket, a metódus törzse ne tartalmazzon megvalósítást
- Legyen egy **OperatorNode** nevű osztály, ami a **Node** osztály leszármazottja az alábbiak szerint:
  - **Operator** – *Operator* típusú csak olvasható tulajdonság
  - egy háromparaméteres konstruktor, ami meghívja az ős azonos paraméterszámú konstruktorát, átadva neki a paraméterül kapott értékeket, törzse beállítja az osztály *Operator* tulajdonság értékét az első, *Data* paraméter alapján
- **Root** – *Node* típusú csak olvasható tulajdonság, a fa gyökerét fogja tárolni
- **BinaryExpressionTree(Node root)** – konstruktor, mely a *Root* tulajdonságnak értékül adja a paraméterül kapott *Node* objektumot, vagyis beállítja a fa gyökerét

  Teszt: próbálja meg példányosítani a BinaryExpressionTree osztályt…

- Jelölje meg az osztály konstruktorát privátként (habár az első pont alapján egyébként is annak kéne lennie), az osztályból példányt csak egy előre meghatározott statikus metóduson keresztül fogunk tudni létrehozni[2]

[2] Static Factory Method – legfontosabb előnye a konstruktorral való példányosítással szemben, hogy választhatjuk például, hogy `null` értékkel térünk vissza a metódusból valamilyen hiba esetén, vagy dobhatunk kivételt is (pl: int.Parse és nem new int(”..”)). A konstruktorban is dobhatunk kivételt, azonban ez egy nagyon rossz gyakorlat, mivel ezesetben egy félig inicializált állapotban marad az objektum. Használják (az előbbieken túl), ha korlátozni akarják az osztályból létrehozható példányok számát, vagy ha a visszatérési típus bármilyen leszármazottjával szeretnének visszatérni.

- **static BinaryExpressionTree Build(string expression)** – publikus metódus, a paraméterül kapott string-et alakítsa karaktertömbbé és azzal hívja meg a következő pontban lévő metódust, visszatérési értéke a meghívott metódusnak a visszatérési értéke legyen módosítás nélkül
- **static BinaryExpressionTree Build(char[] expression)** – publikus metódus, feladata a fa adatszerkezet előállítása a paraméterül kapott RPN kifejezés alapján az alábbi lépéseken keresztül:
    - készítsen egy verem[3] változót (használja a C# beépített generikus verem típust: `Stack<Node> = new Stack<Node>()`)
    - járja be a paraméterül kapott tömböt az elejétől a végéig, majd minden karakter esetén:
        - ha szám, akkor a verembe helyezzen bele egy, a számok tárolására való *Node* példányt, konstruktorának paramétere a szám
        - ha operátor, akkor a veremből vegye ki az első elemet, ez lesz az operátor jobb oldali gyereke, majd vegye ki a következő elemet, ami az operátor bal oldali gyereke lesz, végül készítse el az operátor *Node* példányt, amit helyezzen a verembe
    - miután véget ért a karakterek feldolgozása térjen vissza egy *BinaryExpressionTree* példánnyal, ahol a gyökér elemet a veremben találja (annak egyetlen eleme lesz)
- **string ToString()** – felüldefiniált *ToString()* metódus (publikus), amit meghívva visszaadja a fát bejárva az eredeti RPN kifejezést
    - megvizsgálja, hogy a *Root* rendelkezik-e értékkel, ha nem, akkor üres *string*-el térjen vissza, egyébként hívja meg a *Root* értékkel a következő metódust
- **string ToString(Node node)** – Postorder bejárása a fának, semmilyen feldolgozása ne történjen a csomópontok adataival, csak adja vissza azok értékeit

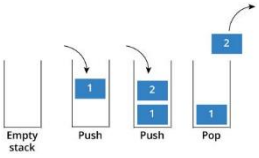
  Teszt: készítsen egy fa példányt, majd írassa ki a Console-ra, pontosan azt kell látnia, amiből felépítette a fát

- **string Convert()** – publikus metódus, amit meghívva visszaadja a fát bejárva az RPN kifejezés zárójelezett formáját
    - megvizsgálja, hogy a *Root* rendelkezik-e értékkel, ha nem, akkor üres *string*-el térjen vissza, egyébként hívja meg a *Root* értékkel a következő metódust
- **string Convert(Node node)** – Inorder bejárása a fának, egyetlen említésre méltó változtatás, hogy ha a vizsgált node *OperatorNode*, akkor a tényleges bejárás megkezdése előtt a visszatérésre használt változóhoz egy '`(`' karaktert kell fűzni, illetve a gyerekek feldolgozását követően, utolsóként egy '`)`' karaktert fűzön a változóhoz

  Teszt: készítsen egy fa példányt, majd hívja meg a Convert() metódust és írassa ki a Console-ra a visszatérési értékét, feleslegesen sok zárójelet kell látnia

- **double Evaluate()** – publikus metódus, amit meghívva visszaadja a fában tárolt RPN kifejezés eredményét
    - megvizsgálja, hogy a *Root* rendelkezik-e értékkel, ha nem, akkor 0 értékkel térjen vissza, egyébként hívja meg a *Root* értékkel a következő metódust
- **double Evaluate(Node node)** – Postorder bejárás egy módosított változata:
    - ha `null` a vizsgált csomópont, akkor térjen vissza 0 értékkel
    - ha a node *OperandNode*, akkor térjen vissza a benne tárolt adat értékével számként
    - ha nem, akkor hívja meg előbb a csomópont bal, majd a jobb gyerekére ezt a metódust
    - ha itt járunk, akkor ez egy *OperatorNode*, így a benne tárolt *Data* alapján végezzük el a bal és jobb gyerekének visszatérési értékén a műveletet, amivel térjünk vissza a metódusból
    - ha valami nem végzetes hiba jön közbe, akkor legyen „nem szám" a visszatérési értéke a metódusnak

[3] A verem (stack) egy LIFO (last in, first out) típusú adatszerkezet, ami csak két műveletet támogat: berak (*Push*), kivesz (*Pop*). A berakás során egy elemet adunk az eddigiekhez úgy, hogy a verem tetejére tesszük, míg a kivétel során az utoljára beszúrt elemet vesszük ki (a verem tetejéről).



<u>Teszt:</u> készítsen egy fa példányt, majd hívja meg az Evaluate() metódust és írassa ki a Console-ra a visszatérési értékét, a kifejezés kiértékelésének eredményét kell látnia

Készíts egy **InvalidExpressionException** nevű **kivétel** osztályt az alábbiak szerint:

- őse legyen az **Exception** osztály
- rendelkezzen egy kétparaméteres konstruktorral, első paramétere *string*-ként a kifejezés lesz, második paramétere pedig egy szám lesz, ami leírja, hogy hányadik karakterrel volt probléma a kifejezésben
- a konstruktor hívja meg az ős konstruktorát a következő üzenettel:
  `$"Invalid character found at position: {position}, in the following expression: '{expression}'!"`
- **string ToString()** – felüldefiniált *ToString()* metódus (annak érdekében, hogy elkerüljük a StackTrace kiírását, mikor közvetlenül *string*-é alakítjuk a kivételünket), ami a következő üzenettel térjen vissza:
  `$"InvalidExpressionException: {Message}"`
- a kivételt a bináris kifejezésfa építése során akkor kell dobni (a *Build* metódusban), ha egy olyan karakter található a kifejezésben, ami se nem szám, se nem az elfogadott operátorok egyike

Tesztelés

Teszteld a **BinaryExpressionTree** osztályt:

- egyetlen számmal, mint kifejezés, pl: `7`
- minden operátorral, külön-külön, pl: `28+`, `28-`, `28*`, `28/`, `28^`
- valami bonyolultabb kifejezéssel, pl: `234*+`, `23*4+`, `23*45*+`, `23+45-/`, `23+4*5+67^8+/`
- valamivel, amivel kiválthatod a kivételt (kapd is el, de csak a Main-ben), pl: `12-3-A-45`

```
Test 'number' is only node..
7                           = 7                           = 7

Test 'simple' operators..
28+                         = (2+8)                       = 10
28-                         = (2-8)                       = -6
28*                         = (2*8)                       = 16
28/                         = (2/8)                       = 0,25
28^                         = (2^8)                       = 256

Test 'example' expressions..
234*+                       = (2+(3*4))                   = 14
23*4+                       = ((2*3)+4)                   = 10
23*45*+                     = ((2*3)+(4*5))               = 26
23+45-/                     = ((2+3)/(4-5))               = -5
23+4*5+67^8+/               = ((((2+3)*4)+5)/((6^7)+8))   = 0,00009

InvalidExpressionException: Invalid character found at position: 5, in the following expression: '12-3-A-45'!
```

További kiegészítési lehetőségek, megoldásuk nem kötelező

- a Reverse Polish Notation alapja a Polish Notation, amit a fa Preorder bejárásával kaphatunk meg
- a fa építése során ne csak a helytelen karakter esetén dobjon *InvalidExpressionException* kivételt, hanem például akkor is, ha a kifejezés helytelen, pl: `12++`
- bővítse a fa képességeit, kezelje az unary operátort (előjelet), a változókat (pl: x, y), és további műveleteket (pl: min, max)
- vagy a Convert() metódust alakítsa át, vagy készítsen egy újabbat, ami képes a felesleges zárójelezéseket eltávolítani az átalakított kifejezésből, mielőtt azt visszaadná a metódus
- készítsen metódust, ami helyes RPN kifejezéseket tud generálni