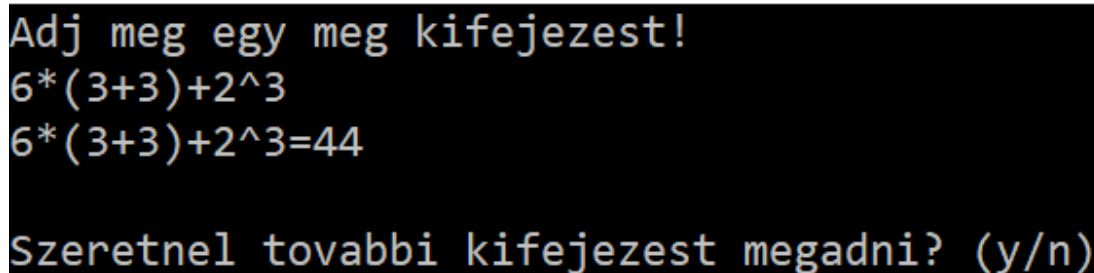


Beadandó a „Programozás C++ nyelven” tantárgyhoz:

Input kifejezés alapján kifejezésfa építése, majd kiértékelése a műveletnek megfelelő műveleti hierarchia alapján

A program működését a cím egyszerűen leírja. A futtatás eredményeképp kapott parancssorba begépeljük a kívánt, kiértékelni kívánt kifejezést és a program elvégzi helyezzünk. Minden kiértékelte helyesen megadott kifejezés után a program felteszi a kérdést, hogy kívánunk-e további inputokat megadni. Minden válasz esetén, ami nem a „y”, azaz igen, a program működése megáll.



```
Adj meg egy meg kifejezest!  
6*(3+3)+2^3  
6*(3+3)+2^3=44  
  
Szeretnel további kifejezest megadni? (y/n)
```

1. ábra Egy kifejezés megadása

A program képes az alpműveletek (összeadás, kivonás, szorzás, osztás) mellett hatványozásra is.

A kifejezés kiértékelésének lépései:

1. Az input bemenet tokenekre bontása – pl.: a „2+3” kifejezés 3 tokenből áll. Az első a „2” a második a „+” és a harmadik a „3”. Itt figyelembe vettem a nyitó és csukó zárójeleket, valamint a változókat is. Sajnos a változókkal való művelet kiértékelésére a program nem képes.
2. A kifejezés kiértékelése
 - a. lengyelforma algoritmus – precedencia és zárójelezett műveletek figyelembevételével
 - b. kifejezés kiértékelése, fa felépítése az Expression osztály gyerekei segítségével

Osztályok és rövid leírásuk:

A Tokenizer osztály felelős az input kifejezések helyes szétbontásáért.

Az Expression osztály objektum orientált szemszögből egy interfész, de mivel a C++ nyelv nem definiál külön interfész típust, ezért nevezhetjük absztrakt osztálynak is. A tartalma a leírtaknak megfelelően olyan virtuális metódusok melyek a leszármazottakban specifikusan kifejtésre kerültek.

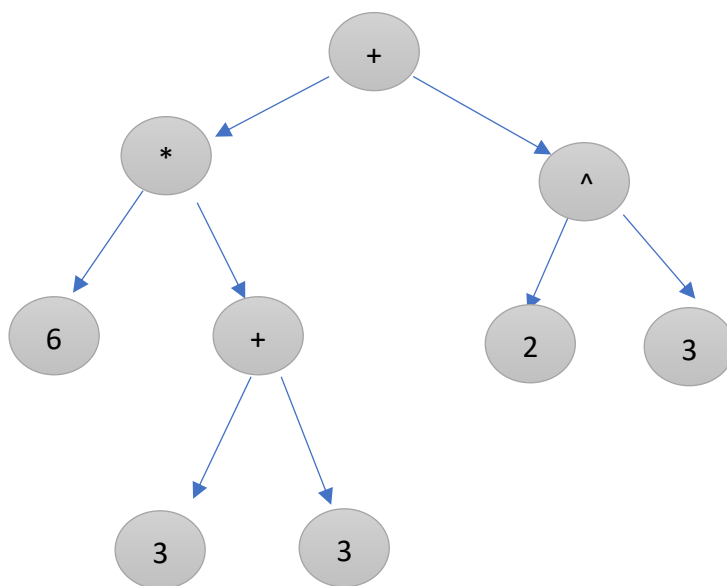
A Constant osztály az Expression egy gyereke, melynek egyetlen feladata, hogy ha az adott token konstans akkor az osztályon belül beállítja az értékét konstansra. Ezen felül a precedence függvény

visszaad egy egész számot, aminek a segítségével a precedencia a kiértékelés során meghatározásra kerül. Ez a szám lényegében mindegy, hogy mi, csak nagyobbnek kell lennie, mint az operátorok precedenciája.

Az Operator osztály kiértékelése egy rekurzív függvényt valósít meg, ami mindig a megfelelő műveleti jelet a jobb és a bal gyerekekkel végzi el. Pl. „2+3” esetén a 2 a bal a 3 a jobb gyerek és a „+” az operátor. Ezen felül a műveleti jelek hierarchiájának meghatározása is itt zajlik.

A Variable osztályban a precedencia ugyancsak meghatározásra kerül. Itt a változó egy map-be kerül változónév és egy érték szerint, melyet később felhasználhatunk.

A main.cpp fájlban lévő postfixExpr függvény végzi a kifejezés felírását a lengyelforma szerint. Az algoritmusnak megfelelően egy stack tárolóba helyezzzük addig a tokeneket, amíg egy műveletet minden komponense nem szerepel a stackben, majd ezt egy vectorba helyezzük. Majd nem túl elegánsan egy, a main függvényben lévő ciklus végig iterál a megkapott postfix vectoron majd egy stack felhasználásával felrajzolásra kerül a fa és annak gyökérelemének segítségével és a fentebb leírt osztályokkal megkapjuk a végeredményt.



2. ábra Az 1.ábrán látható kifejezés fa formában

Kiegészítés (2018.05.29):

A program képes változók kezelésére is. A kifejezés megadása előtt, ha szeretnénk, akkor megadhatjuk a felhasználni kívánt változókat, melyeket később a kifejezés megadásakor felhasználhatunk és a végeredmény ezek szerint kerül kiértékelésre. A változók megadása üres sorig megy. Ezen felül a kifejezés megadásánál a szóközöket a Tokenizer osztály getNext() függvénye figyelmen kívül veszi.

A program futása képekben:

1. A felhasználni kívánt változók megadása

```
Adj meg egy változót.  
x  
Adj meg a változó értékét.  
2  
Adj meg egy változót.  
y  
Adj meg a változó értékét.  
7
```

2. A kiértékelni kifejezés(ek) megadása

```
-----  
A felhasználható változók a következők:  
x = 2  
y = 7  
-----  
  
Adj meg egy meg kifejezést!  
(x+2)*3-(4+y)^2  
(x+2)*3-(4+y)^2=-109  
  
Szeretnél további kifejezést megadni? (y/n)  
n
```