HÁZI FELADAT

Programozás alapjai 2.

Pontosított feladatspecifikáció

Németh Dániel
FTYYJR

2019. március 31.

TARTALOM

1. Feladat

```
Szoftver laboratórium II. házi feladat
Teszt Elek (ELEK07) részére:


Kifejezés fa
Készítsen kifejezés fát reprezentáló osztályt! Definiáljon egyszerű
általánosított műveleteket, amelyeket kiértékel a kifejezésfa segítségével!
Valósítsa meg az összes értelmes műveletet operátor átdefiniálással
(overload), de nem kell ragaszkodni az összes operátor átdefiniálásához!
Demonstrálja a működést külön modulként fordított tesztprogrammal! A
megoldáshoz ne használjon STL tárolót!
```

2. Pontosított feladatspecifikáció

A feladat egy kifejezésfát kezelő osztály elkészítése. Az osztály segítségével kifejezéseket hozhatunk létre, tárolhatunk, kombinálhatunk és kiértékelhetünk. Az osztállyal bármekkora, a memóriában elférő kifejezést előállíthatunk (dinamikus memóriakezelés).

Az Expression osztály képes többféle típusú kifejezést is kezelni (int, double) bemenetként és kimenetként, ezt template segítségével fogom megoldani.

Az alábbi funkciókat tervezem megvalósítani:

Expression(): számértéknél nullára inicializál, esetlegesen egyéb használt osztálynál az osztály default konstruktorát hívja.
Expression(érték): a felhasználó által megadott értékre inicializál, ennek az értéknek a típusával fog dolgozni a fa.
Expression(string): Képes infix formátumban megadott stringből kifejezésfát építeni
Támogatott műveletek: +, -, *, /, ^ (hatványozás): két kifejezést ezekkel tudunk összekapcsolni akár helyben is(működik pl a +=, -=, stb. operátorok), akár új változóba (copy constructor)
Kiírás: az osztály képes olvasható formában megjeleníteni a gráfot (postfix, infix vagy prefix alakban, string formátumban). Inserter operátor (<<) alapértelmezetten infixet használ.
Eval(): a kifejezés értékét adja vissza.
Tesztelés: különböző kifejezéseket állítok elő, és a kiértékeléskor kapott értékeket összevetem a c++ beépített kiértékelője által visszaadott értékekkel.

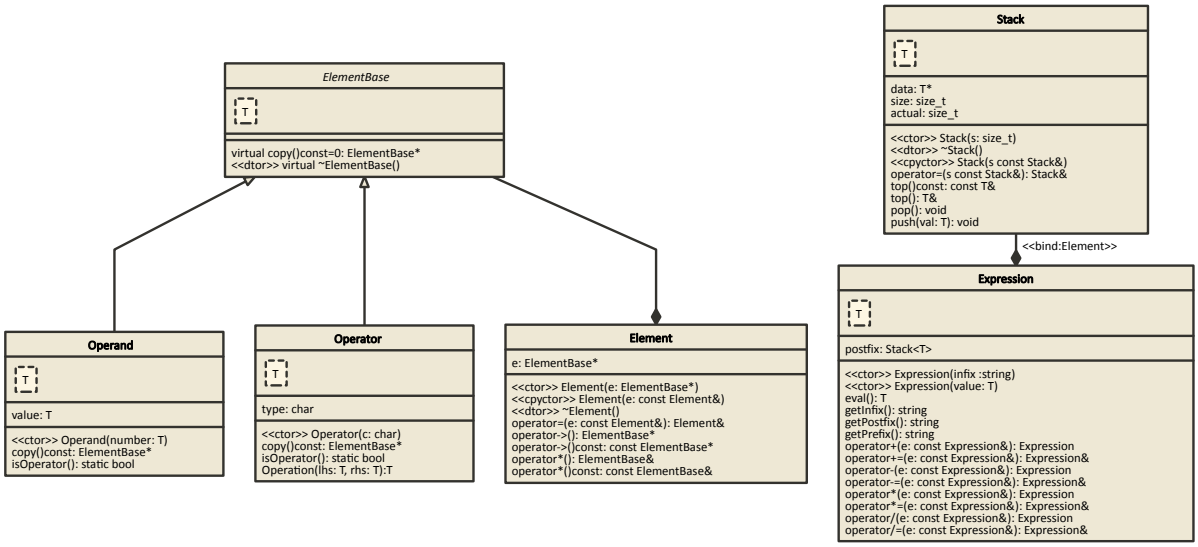
3. Terv

3.1. Objektumterv

A kifejezést egy stackben fogom tárolni postfix alakban, ahol minden elem egy érték vagy egy operátor (heterogán kollekció). Ez előnyösebb egy fával szemben olyan szempontból, hogy nem kell feltétlenül minden új elem beillesztésénél memóriafoglalást végezni. Bevett algoritmusok állnak rendelkezésre a postfixből és a postfixbe konvertáláshoz.

Az ElementBase absztrakt ősosztályból származnak le az Operator és az Operand osztályok. Az Element osztály becsomagolja az ElementBase típusúakat és érték szerint másolhatóvá teszi. Az Expression osztály egy Element osztályú elemeket tartalmazó stackben tárolja a kifejezést. A stack ezenfelül szükséges még a konverziós algoritmusokhoz.

Globális függvényként elkészítem az Expression-höz tartozó inserter(<<) operátort.



3.2. Algoritmusok

3.2.1. Infixből postfix

Balról jobbra olvas(amíg a string végére érünk):
Ha operandus: push postfixbe.
Egyébként:

Ha nagyobb a precedencia, mint a stakben levőé, vagy a stack üres, vagy nyitó zárójel van benne: push postfixbe

Egyébként: pop minden operátorra, aminek a precedenciája nagyobb vagy egyenlő a beolvasottnál. Beolvasott push stackbe
(: push
): amíg nem) „(”: pop és push postfixbe

3.2.2. Postfixből infix

Létrehozunk egy temp stacket
Balról jobba:
Operandus: push tempbe
Egyébként:
stackből pop az utolsó két elemre
string= operand1 operator operand2

stringet pusholjuk a tempbe
Ha csak egy érték maradt a tempben: ez az infix

3.2.3. Postfixből prefix

Balról jobbra:
Operandus: push stackbe
Operator: stackól pop két operandus, összefűzni az operátort a két operandussal, stringet push stackbe

3.2.4. Kifejezések összekapcsolása operátorral

A meglévő postfix végére push(új operátor), majd push(új expression elemei sorban balról jobbra)

3.2.5. Tesztprogram algoritmusai

A lekódolt metódusokat egy alkalmazási példa lapaján meghívja a tesztprogram.
A tesztprogram a standard inputról file végéig olvas. Az első beolvasott adat egy teszteset sorszámot jelent. Ezt követően egy megjegyzés lehet az adott sorban. A beolvasott szám dönti el, hogy melyik teszteset fut a megjegyzés pedig az adott tesztesetre vonatkozhat.
Az 1. tesztben minden első sorból egy infix stringet olvasunk be, minden második sorból pedig az elvárt eredményt.
A 2. tesztben az inputról kapjuk az elvárt infix vagy prefix stringet, a program pedig teszteli a konverzió helyességét.
A 3. tesztben a tesztprogramban meghatározott műveleteket végzünk kifejezésekkel, az inputról pedig soronként kapjuk az elvárt eredményt.