Popis pre používateľa

Program Calc je kalkulačka, ktorá okrem výpočtu jednoduchých aritmetických výrazov zjednodušuje výrazy obsahujúce premenné, teda manipuluje s polynómami.

Používateľ do riadku zadá výraz na vyhodnotenie, stlačí Enter, a pokiaľ je vo výraze čo vyhodnotiť/upraviť, na novom riadku sa zobrazí zjednodušený výraz alebo číslo.
Platným vstupom sú čísla a aritmetické operácie na nich, ktoré sa zadávajú znakmi + pre súčet, - pre rozdiel, * pre súčin, / pre podiel, ^ pre mocnenie. Medzi číslami a znakmi operácií môže byť ľubovoľný počet medzier, ale nemusí byť žiadna.
Rovnaké operácie sú definované aj na premenných.
Násobenie premennej číslom sa píše explicitne pomocou znamienka *, teda dvojnásobok x je 2*x alebo x*2. Tvar 2x nie je podporovaný. Na vstupe môžu byť aj rôzne premenné, ale s ich súčinom nie je možné manipulovať.
Ľubovoľný podvýraz je možné dať do zátvorky, a tým mu zvýšiť prioritu pri vyhodnocovaní. Podporované sú len guľaté zátvorky ( ).

Ukážky platných vstupných výrazov a ich výstupov:

Vstup	Výstup
-2 + 6 - 1 + 9	12
a + a - 3	2*a - 3
(4 + 2)/3*2	4
5*x + z/z - (8/4)*y	5*x - 2*y + 1
3^(3 - 1)	9
(2^3)^2*x - x^2 + n^0	-x^2 + 64*x + 1
(x + 1)*(x^2 + 2*x +3)	x^3 + 3*x^2 + 5*x + 3
(x+4)^2	x^2 + 8*x + 16
(x*x)/x^5	x^-3
(2*c^2 + 1 -c)*(2-c)	-2*c^3 + 5*c^2 - 3*c + 2
(x-x)/(-3+5-2)	Exception: Division by Zero Not Defined
(3-x+2^3+2*x-5)^3	x^3 + 18*x^2 + 108*x + 216

Objektový návrh

Program je logicky členený do troch hlavných častí: triedy Tokenizer, Parser, Evaluator.

Keďže string na vstupe môže byť hocijaká postupnosť znakov, nie je možné ho priamo interpretovať. Je v ňom potrebné rozpoznať základné jednotky, ktoré sú už nedeliteľné a s ktorými vie kalkulačka pracovať. Teda číslo, premenná, znamienka operácií + - * / ^ a zátvorky (). To má na starosti trieda Tokenizer.

Trieda Tokenizer pomocou metódy GetNextToken postupne číta vstup. Prečítaný znak porovná so znakmi, ktoré sú definované ako platné vstupy, prípadne overí, či skupina znakov je číslo alebo názov premennej (postupnosť písmen). Z takto rozpoznaných znakov vstupu vytvára významové jednotky tokeny – objekty triedy Token. Tie predstavujú čísla, premenné, zátvorky alebo jednotlivé operácie. Je definovaných 10 tokenových typov INTEGER, VAR, PLUS, MINUS, MUL, DIV, LPAREN, RPAREN, POW, EOF.

Ak je na vstupe prítomný nedefinovaný znak, vyvolá to výnimku InvalidCharacterException. Medzery vo vstupe sú samozrejme ignorované, o čo sa stará metóda SkipWhitespace.

Potom je medzi tokenmi, ktoré vznikli z rozpoznaných platných znakov, potrebné rozpoznať syntaktickú štruktúru. To znamená zistiť, či sú dané znaky v správnom poradí, ktoré zodpevedá platnému tvaru matematického výrazu. Napríklad po čísle nemôže nasledovať ďalšie číslo, alebo otváracia zátvorka musí byť raz uzatvorená.

Tokeny sú odovzdávané triede Parser, ktorá pomocou metódy Parse kontroluje správnosť poradia jednotlivých tokenov a potom ich spája, pričom berie ohľad na prioritu zátvoriek a operácií.

Využíva pri tom gramatiku zloženú zo štyroch pravidiel zoradených zostupne od navyššej priority:

Atom:	+- Factor	INTEGER/VAR	LPAREN Expr RPAREN
Factor:	Atom POW Factor		
Term:	Factor MUL/DIV Factor		
Expr:	Term PLUS/MINUS Term		

Pre tieto štyri pravidlá sú definované metódy Atom, Factor, Term, Expr, vracajúce objekt typu AST. Metóda Parse teda volá metódu Expr, tá zase metódu Term atď.

V jednotlivých metódach, ktoré zodpovedajú daným štyrom pravidlám, sa z pospájaných tokenov vytvárajú uzly abstraktného syntaktického stromu (AST – Abstract Syntax Tree). Čísla sú uložené v uzloch Num. Uzol pre premenné sa nazýva Var a je v ňom uložené meno premennej, jej koeficient a tiež jej mocnina, čo umožňuje jednoduchú manipuláciu s premennými. Rozlišuje sa medzi uzlom pre unárne operácie UnaryOp a uzlom pre binárne

operácie BinOp. Tieto uzly obsahujú token danej operácie a jeden alebo dva argumenty, ktorými môžu byť literály, teda čísla a premenné, či ďalšie vnorené operácie. Všetky uzly sú dedičmi spoločnej triedy AST.

Ak sa počas vytvárania AST vyskytne neočakávané poradie tokenov, vyvolá to výnimku InvalidSyntaxException.

Trieda Evaluator potom vezme daný AST a pomocou metódy Simplify na ňom vykonáva transformácie uzlov a tým ho zjednodušuje. Metoda Simplify teda dostane nejaký uzol AST stromu a podľa toho, či je typu Num, Var, UnaryOp alebo BinOp, sa rozhoduje, ako ďalej postupovať. Pri typoch Num, Var sa vráti samotný uzol. Pri UnaryOp, čo môže byť unárne +-, sa podľa definovaného pravidla zmení znamienko všetkých členov vnútorného výrazu a metoda Simplify sa rekurzívne zavolá na výraz, ktorý bol v uzle UnaryOp. Pri BinaryOp sa rozlišuje, čo je to za operáciu, a akého typu je ľavý a pravý argument, vykoná sa potrebná transformácia, napr. pri násobení čísla nejakým šúčtom sa vytvorí uzol BinOp s operáciou súčet čísla vynásobeného ľavým argumentom daného súčtu s číslom vynásobeným pravým argumentom toho istého pôvodného súčtu. Všetky úpravy prebiehajú rekurzívne, až kým už nie je čo upravovať, a vďaka štruktúre AST sa vykonávajú v správnom poradí, podľa priority operácií. Pokiaľ v AST nie je čo upravovať už na začiatku, nastane výnimka NothingToSimplifyException a program vráti samotný nezmenený vstup.

Transformovaný AST, v ktorom sa nachádzajú už len uzly typu Num či Var, prevezme metóda Stringify, ktorá čísla a rovnaké premenné s rovnakou mocninou posčíta a potom ich prevedie do stringu, v ktorom sú premenné vypísané v abecednom poradí s klesajúcou mocninou. Na tento účel je použitý zotriedený slovník, do ktorého sa jednoducho pridávajú nové premenné či upravujú koeficienty už existujúcich premenných.