СУ “Св. Климент Охридски”,
ФМИ – Софтуерно инженерство
Курсов проект по Структури от данни и алгоритми

Mini-Mathematica

Валентин Кирилов Кирилов

Факултетен № 61701

Съдържание

1. Въведение
2. Описание на приложените алгоритми
3. Описание на програмния код
4. Използвани технологии
5. Инсталация и настройки
6. Приложимост на проекта

1. Въведение

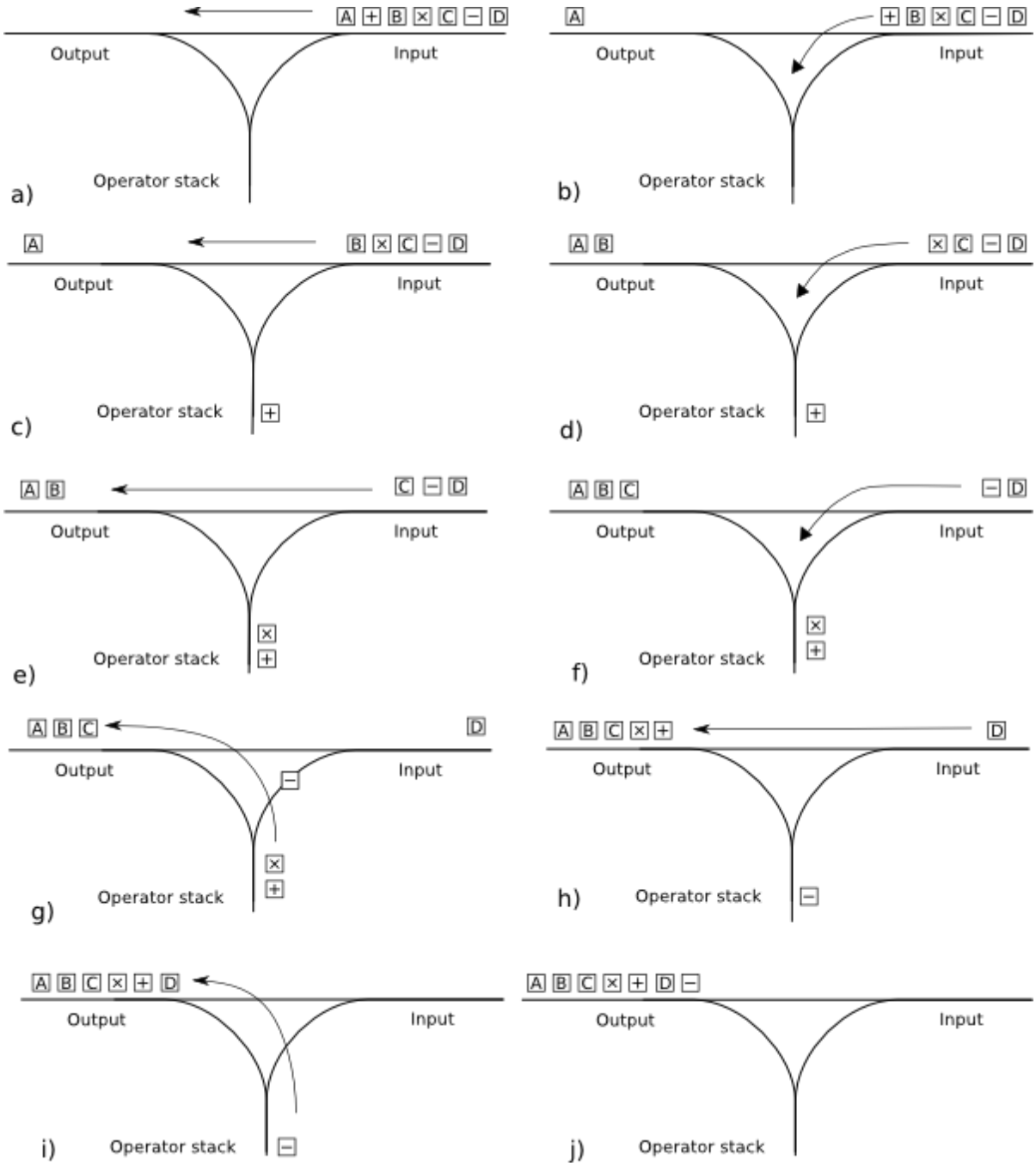
Проекта представлява прост калкулатор, който може да пресметне математически изрази състоящи се от аритметични операции (+, -, *, /), тригонометрични функции (sin, cos, tan, cotg) и няколко допълнителни функции (pow, log, sqrt). За работата си калкулатора използва основно два алгоритъма - Shunting Yard, за привеждане на израза в постфиксен запис и съответно RPN за пресмятането на стойността му.

2. Описание на приложените алгоритми

Пресмятането на математически изрази е разделено на няколко етапа, за които се използват различни алгоритми:

1. Етап 1: Разделяне на израза на token-и
   Входния израз идва като string, който трябва да бъде разделен на части, за да бъде обработен от програмата. За целта той се обхожда символ по символ, като се отделят числата (поддържа и числа с плаваща запетая), аритметичните операции и функциите (поддържат се функции с два аргумента разделени със запетая - например pow(2, 4)). Разделените token-и се връщат като масив и се подават за по-нататъшна обработка към другите алгоритми.

2. Етап 2: Превръщане на израза в постфиксен запис
   След разделянето на входния израз на token-и, резултатът от това се подава на функция, която използва Shunting Yard алгоритъма и превръща израза в постфиксен запис.



фиг. 2.1 - Визуализация на работата на Shunting Yard алгоритъма

За работата си алгоритъма използва стек, в който съхранява аритметичните операции и функциите и опашка, в която записва изходния резултат. Работата на алгоритъма се състои в обхождане на входа от token-и и тяхната оценка. Това се случва по следния начин:

- ако е число се слага в опашката
- ако е функция се слага в стека
- ако е запетая
    - ако в стека няма отваряща скоба (- изход с грешка
    - докато не намери отвяраща скоба (се pop-ват елементи от стека и се записват в опашката

- ако е аритметична операция
    - докато в стека има елементи и приоритета на текущата операция е по-малък или същия като приоритета на операцията на върха на стека, операциите се pop-ват и записват в опашката
    - текущата операция се вкарва в стека
- ако е отваряща скоба (се пъха в стека
- ако е затваряща скоба)
    - ако в стека няма отваряща скоба (- изход с грешка
    - докато не намери отваряща скоба (pop-ва резултатите от стека и ги записва в опашката. Когато намери отварящата скоба се вади от стека, но не се извършва нищо с нея.
    - ако в стека е останала функция, тя се pop-ва и записва в опашката

След като свършат token-ите елементите от стека се pop-ват и записват в опашката. Ако някой от тях е скоба изпълнението се прекъва със съобщение за грешка.

По подборно, работата на алгоритъма е описана тук, а съответно кода, който имплементира съответната функционалност може да бъде разгледан тук във функцията `convertToRPN`.

3. Етап 3: Пресмятане на постфиксния запис, генериран от Shunting yard алгоритъма

Stack	Unread expression	Comments
Empty	3 4 5 + x	
3	4 5 + x	Numbers are pushed on the stack
3 4	5 + x	
3 4 5	+ x	
3 9	x	Pop 4 and 5, push 4 5 +
27	No more input	Pop 3 and 9, push 3 9 x
Empty		Pop and display the result, 27

фиг. 2.2 - работа на RPN алгоритъма

След като израза е достигнал във форма за обратна полска нотация, неговата стойност може да бъде пресметната чрез така известния алгоритъм за това. За целта се използват стек и се обхождат елементите, който са разделени вече посредством Shunting Yard алгоритъма. При обхождането на елементите се правят следните действия и оценки:

- ако е число се пъха в стека
- ако е аритметична операция или функция

- прилага операцията или функцията в зависимост от това колко аргумента се нуждае тя, като ги `pop`-ва от стека. В зависимост от функциите се налагат допълнителни проверки. Например при операцията за делене се гледа дали втория аргумент е `0`, а при функцията за изчисляване на корен, се проверява дали аргумента е отрицателно число. В тези случай и в случай, че функцията се нуждае от аргументи, какъвто брой не е наличен в стека, операцията се прекъсва с изход за грешка.

Реализацията на алгоритъма може да бъде разгледане тук във функцията `calculateWithReversePolishNotation`.

3. Описание на програмния код

За изграждането на проекта е използван AngularJS, което обуславя използвания подход за разделение на логиката и структуриране на компонентите на проекта, спазвайки обособените от тях конвенции.

Разделението на файловата структура както и самият source код може да бъде разгледани в хранилището на проекта - тук. Важните за нас файлове и директории са:

- index.html - основна страница, която зарежда всички компоненти на приложението
- app.js - основен файл, обединяващ компонентите на приложението
- calculator/ - директория съдържаща контролер, отговарящ за калкулатора както и markup файл за визуализацията му. Calculator.js контролерът отговаря за връзка с cac service-a, чрез който се извършват изчисленията, както и за реализация на функционалностите свързани с потребителския интерфейс и обработка на настъпилите събития (клик върху бутон с число или функция от интерфейса на калкулатора). Също така този контролер съхранява списък с история от последните пресместнати операции и ги подава към интерфейса за визуализация.
- components/services/ - съдържа service използвани от приложението
- components/services/calc.js - основен service, който държи логиката използвана за работата на калкулатора. Използва се от Calculator контролера, като се извикват методите му. По този начин контролера се пази чист, а логиката се отделя и може да бъде преизползвана от различни контролери.
- bower_components/ - директория, съзържаща модули, необходими за функционирането на системата
- styles/ - съдържа стиловете необходими на приложението. В поддиректорията sass са поместени файловете, които се използват за прекомпилиране на тиловете до css
- bower.json - съдържа списък от пакетите необходими на приложението. Използва се от Bower
- packages.json - съдържа списък от пакетите необходими на приложението. Използва се от Grunt
- Gruntfile.js - конфигурационенн файл, използван от Grunt, съдържащ списък със задачи, които да изпълнява

4. Използвани технологии

Тъй като проекта представлява Single Page Application за реализацията му са използвани технологии предназначени за това. Основният език, който е използван при работата по проекта е JavaScript, като за стилизация и оформление е използван SASS(CSS).

За работата по проекта е използван AngularJS, които предоставя възможност за изграждане на SPA приложения. В комбинация с него се използват и средства за ускоряване на работата и автоматизация на процеса по разработка като Bower и Grunt. Допълнително за стилизацията на сайта се грижи SASS в комбинация с иконки от font-awesome.

5. Инсталация и настройки

За инсталацията и пускането на проекта е необходимо на системата да бъдат налични следните компоненти:

- Node.js и NPM
- Grunt и Bower (тези пакети са опционални, тъй като могат да бъдат изтеглени локално за проекта посредством npm)

Инструкции за инсталация:

1. Изтегляне на repository с кода. Може да се използва и github интерфейса.
   git clone https://github.com/valkirilov/Mini-Mathematica.git
2. Изтегляне на пакетите необходими за проекта
   npm install
   bower install

3. За да се стартира приложението трябва да се използва следната команда
   grunt

4. Отворете браузъра си на `http://0.0.0.0:8000` за да достъпите приложението

6. Приложимост на проекта

Проекта може да бъде използван както за пресмятането на прости математически изрази, така и за изчисляването на сложни задачи, състоящи се от тригонометрични, експоненциални и логаритмични функции. Приложението има и потребителски интерфейс, който е удобен и познат от потребителите, което допринася за по-лесната му употреба.

Проектът е достъпен на адрес: http://valkirilov.github.io/Mini-Mathematica

Хранилище с кода на проекта: https://github.com/valkirilov/Mini-Mathematica