Programátorská dokumentace

V repozitáři můžete najít 2 projekty, **CzeTex** a **CzeTex.Tests**, kde **CzeTex** představuje samotný program a **CzeTex.Tests** je projekt, který spouští testy.

CzeTex

Projekt i zdrojový kód je rozdělen do dvou částí. První se zabývá zpracováním vstupních souborů, vystavěním pomocných datových struktur nebo poskytováním pomocných metod. Druhá část se zabývá generováním výsledného .pdf souboru pomocí knihovny **iText**.

První část

Do první části bychom mohl zahrnout následující soubory **Commander.cs, Files.cs, SetupLoader.cs, Trie.cs** a **Util.cs**. Pojďme si tyto soubory a jejich obsah projít jednotlivě.

Commander.cs

V souboru **Commander.cs** můžeme najít stejnojmennou třídu, která představuje centrum celého programu, volá dílčí funkce a třídy a udržuje koherentnost celého běhu. Zpracovává obsah načtený ze vstupního souboru, volá třídu pro zpracování setup .json souboru.

Pro zpracování vstupního souboru se využívá metoda **ReadContent**, která detekuje speciální znak, v základu /, a následně extrahuje funkce a jejich parametry z textu, vyhledá v pomocné datové struktuře **trie** odkaz na příslušnou funkci a zavolají.

Třída obsahuje několik funkcí jako **ReadRestForParameter, ReadParameter atd.,** které se navzájem doplňují v různých případech pro korektní vyhledání parametrů textových CzeTex funkcí.

Files.cs

Dalším souborem je **Files.cs** obsahující stejnojmennou třídu, která zabezpečuje vyhledání souboru podle poskytnutého parametru od uživatele a následné korektní otevření, načtení obsahu a uzavření souboru.

Mimo těchto funkcionalit nabízí třída Files také možnost generování cesty k umístění výsledného .pdf souboru v metodě **GetPdfPath** nebo metodu **GetBaseName**, která vrací jméno bez přípony vstupního souboru a další.

SetupLoader.cs

Dalším souborem je **SetupLoader.cs** obsahující 2 třídy **JsonEntry** a **SetupLoader**.

Jak už název napovídá **JsonEntry** reprezentuje jeden záznam v setup .json souboru. Má atributy pro **addFunction, getFunction** zda má být funkce dynamicky generovaná a potom případně znak pro dynamicky generované funkce.

Třída **SetupLoader** načítá z setup JSON souboru mapování podle kterého bude následně přiřazovat

konkrétní textové CzeTex funkce k jejich kódovým ekvivalentům. Jak už bylo zmíněno, funkce si můžeme rozdělit na dynamicky generované a již předvytvořené. V metodě **AddFunctions** procházíme přes všechny záznamy v JSON souborů, tedy již vytvořené instance třídy JsonEntry. A následně rozhodujeme, zda má být funkce dynamicky generované nebo ne, dle příznaku v **dynamicallyGenerated**.

Když vyhodnotíme, že daná funkce musí být ještě vytvořená, tak přejdeme do větve programu v metodě **AddDynamicallyGeneratedFunction**, kde příslušné funkce pro vytvoření delegátů pro add a get funkce. A následně dané funkce přidáme to **trie** stromu. Pozn: Tyto dynamické funkce jsou využívané pouze pro funkce zobrazující nějaký znak, proto jsou silně spjaté s generací těchto funkcí. Ale možné rozšíření je jednoduché, pouze by bylo potřeba volat jinou funkci pro vytváření delegátů.

Pro funkce, které se nacházejí v kódu, vyhodnotíme zda má daná funkce mít pouze svou add a get funkci nebo pouze add, následně už pomocí funkce **CreateDelegate** a potřebnými parametry vytvoříme delegáty pro tyto funkce a obdobně jako při dynamických funkcí je přidáme do **trie** stromu.

Util.cs

Tento soubor obsahuje nemalé množství různých pomocných tříd/funkcí, které jsou využívané v ostatních částech kódu. Můžeme zde najít implementace generického zásobníku a k němu příslušné implementaci generického vrcholu spojového seznamu.

Také zde můžeme najít custom třídy pro chybové hlášky, které dávají spolu s textem chybové hlášky lepší představu uživateli, kde udělal chybu při psaní CzeTex textu. Následně se zde nachází také různé statické třídy, které obsahují funkce pro práci s řetězci nebo parametry.

V neposlední řadě tu také najdeme třídy **FunctionGeneratorForPDF** která obsahuje metody pro generování delegátů pro jak add tak get dynamicky generované funkce, jejichž využití jsme již popsali výše.

Trie.cs

Posledním souborem v první sekci je **Trie.cs** obsahující implementaci **trie** stromu, také známý jako písmenkový strom, který se využívá pro vytvoření bijekce mezi CzeTex textovými funkce a funkce provádějící konkrétní kód.

Mimo samotného stromu obsahuje třída také 2 pole, jedno pro add funkce a druhé pro get funkce. Bijekce na stromě je tvořena následovně: posloupnost písmen od kořene stromu do vrcholu musí odpovídat názvu CzeTex funkce. Z toho nám plynou určitá omezení pro názvy CzeTex funkcí. Strom je budovaný pouze z malých písmen anglické abecedy, tedy funkce nemohou používat jinou množinu znaků. Aby se jednalo o bijekci, tak musí platit, že každá funkce má unikátní jméno, což nám vyřeší už vstupní setup JSON soubor, poněvadž v něm nemůžou existovat 2 záznamy se stejným klíčem a v našem případě klíče odpovídají názvu CzeTex funkce.

Když najdeme vrchol jehož posloupnost znaků na cestě od kořene je stejná jako jméno CzeTex funkce, pak bude v daném vrcholu uložený unsigned číslo odpovídající indexu v polích add a get funkcí. Pozn: I

když nějaká funkce nemusí mít get funkci, tak je dané políčko nastavené na null, tedy můžeme mít pouze jeden index pro 2 pole a vše bude fungovat korektně.

Volba této metody ukládání delegátů na funkce nám zabezpečí, že dokážeme nalézt potřebné mapování v lineární čase vůči délce názvu CzeTex funkce. Na rozdíl od lineárního času vůči počtu prvků, pokud bychom vždy museli projít všechny funkce. A zároveň se vyhneme nepříznivé horní asymptotické hranici, se kterou bychom se mohli setkat pokud bychom využívali hashování a také díky tomu, že známe universum ze kterého všechny názvy funkcí pochází, tak by bylo i zbytečné.

Poznámka: rozšíření jména CzeTex funkcí na i velké znaky Anglické abecedy by nebyl žádný velký problém v kódu, pouze by bylo potřeba změnit konstanty ve třídě **TrieConstants** kde by se **smallestAvailableCharacter** změnil na 'A' a **numberOfChildren** na 58. Což by bohužel mimo velkých znaků přidalo i speciální znaky, což jsem považoval za nežádoucí a zároveň by strom zbytečně rychle zvyšoval svou velikost, poněvadž při přidání pouze jednoho potomka k jednomu vrcholu by bylo v paměti vyhrazeno místo pro dalších 57. Což u řídkého stromu je zbytečné.

Druhá část

Druhá část zdrojového kódu se zabývá generováním PDF a souvisejícími operacemi. A znovu se skládá s několika souborů.

Fonts.cs a Signs.cs

Tyto 2 soubory jsou si dost podobné poněvadž každý z nich obsahuje jednu třídu, která obsahuje konstanty. Tech třída Fonts obsahuje různé fonty, výchozí velikost textu nebo velikost odsazení mezi odstavci či řádky. Třída Signs na druhou stranu obsahuje konstanty reprezentující různé speciální znaky jako je např. znak pro děleno, implikace, kvantifikátory atd.

PDFSupport.cs

Soubor **PDFSupport.cs** obsahuje, jak už název naznačuje, shluk podpůrných tříd a metod, tedy hlavně třídy pro různé vrstvy testu nebo wrapper pro iText třídy pro text, který rozšiřuje funkcionalitu knihovní třídy.

Máme jednu rodičovskou třídu **TextCharacteristics**, která představuje takovou obecnou charakterizaci textu, font, velikost textu atd. Dále existuje několik tříd, které z této třídy dědí a následně implementují jednu nebo i více z virtuálních metod, jako je například metoda Special, Add atd. Následně tyto metody pozměňují nějakým způsobem vzhled výsledného textu. Např. třída UnderLineText si uzpůsobuje metodu Special tak, aby přidávala podtržení textu.

Aby byla struktura dědění více přehlednější, tak potomci, kteří mají podobnou charakteristiku tak jim je nastaven předek, který tuto charakteristiku na stavuje a následně je možné je podle tohoto předka v kódu filtrovat. Pro názornost předvedu příklad na třídě UnderLineText: **TextCharacteristics** ⇒ **SpecialTextCharacteristics** ⇒ **UnderLineText** neboli z TextCharacteristics dědí SpecialTextCharacteristics, která shlukuje všechny funkce s implementovanou speciální metodou a následně až potom z ní dědí třída UnderLineText.

Jak už bylo zmíněno výše v tomto souboru můžeme také najít třídu, která dědí z knihovní třídy Text, a rozšiřuje její funkcionalitu. A to zejména kvůli nastavení "pozdvihnutí" textu, poněvadž knihovní třída má tuto funkcionalitu ne úplně nejlépe implementovanou.

PDFStack.cs

V souboru **PDFStack.cs** najdeme třídu CharacteristicsStack, tedy implementaci zásobníku, která jako své prvky uchovává charakteristiku textu. Kde se následně pro charakterizaci aktuálního textu využívá vrchní vrstva.

Zásobník je implementován tak, že není potřeba vždy definovat plně každou vrstvu poněvadž pokud například nová vrstva definuje pouze font, potom se velikost písma kopíruje z předchozí vrstvy (pokud bychom přidávali neúplnou charakteristiku do prázdného zásobníku, tak se nastaví výchozí parametry pro chybějící části).

PDF.cs a PDFMath.cs

Posledními soubory jsou **PDF.cs** a **PDFMath.cs**, které obsahují třídu **PDF** obsahující funkce pro generování .pdf souboru. Třída je rozdělena tak, že v souboru PDFMath.cs jsou matematické funkce a v PDF.cs jsou všeobecné funkce k generování.

Tedy v souboru **PDF.cs** a následně v třídě **PDF** dochází již při volání konstruktoru k vytvoření kostry .pdf souboru a následně pomocí volání funkcí namapovaných na textové CzeTex funkce dochází k upravování .pdf souboru. Můžeme tady najít metody jak pro vytvoření textu, přidání nadpisu, vytvoření seznamu atd.

V soubory **PDFMath.cs** jak už bylo zmíněno najdeme matematicky zaměřené textové funkce. Tedy najdeme tu metody, které slouží k přidávání matematické notace a využívají je např. delegáti pro znakové dynamicky generované funkce.

PODROBNĚJŠÍ INFORMACE KE VŠEM TŘÍDÁM, METODÁM ATD. SI MŮŽETE PŘEČÍST PŘÍMO V KOMENTOVANÉM ZDROJOVÉM KÓDU.

Testy

V projektu **CzeTex.Tests** v soubor **Tests.cs** můžeme najít několik desítek testovacích procedur, které testují jak konkrétní funkce, tak správnou interpretaci souborů s různými testovacími příklady.

Pro testování správné interpretace celých souborů se využívá funkce **RunScript**, která simuluje chování uživatele a zadávání příkazů do terminálu. Tedy spouští příslušné **dotnet run** příkazy v návaznosti na vstupní parametry funkce. Dokáže tedy testovat jak na pouze input soubory tak na kombinaci input a setup souborů.

Spuštění testů,

Pro spuštění testů musíte nejdříve přejít do složky **CzeTex.Tests** a následně spustit příkaz **dotnet tests**.

Pozn: Funkce **RunScript** nepropouští k volající funkci chybové hlášky, tedy vyhodnocení chování se provádí pomocí **exit kódu**. Ten pro **CzeTexExceptions** může být na různých zařízeních a operačních systémech jiný, poněvadž se automaticky generuje. Ve výchozím natavení je **exit kód** roven 134, pokud spustíte testy a uvidíte že se všechny testy liší od 134 vždy se stejnou hodnotou, tak to znamená, že byl výjimkám přiřazen jiný **exit kód**, tedy pro správné fungování musíte v třídě, kde se nachází testy zaměnit hodnotu původního **exit kódu** za vaši, nahrazení konstantní proměnné na řádku **15** v souboru **Tests.cs**.