**ŽILINSKÁ UNIVERZITA V ŽILINE**

**FAKULTA RIADENIA A INFORMATIKY**

**ALGORITMY A ÚDAJOVÉ ŠTRUKTÚRY 1**

Semestrálna práca

Pavol Brišák 5ZYI25

**Školský rok 2022/2023**

**Cvičiaci: Michal Mrena, Streda 11:00**

# Návrh aplikácie

Po spustení aplikácie, sa používateľovi otvorí konzolové okno kde má na výber pracovať s územnými jednotkami (1. úroveň), prechádzať hierarchiou a triediť dané výstupy (2. a 4. úroveň) a vyberať z tabuľky (3. úroveň). Hlavný kód je v triede *main*, ktorý skončí kým používateľ neukončí aplikáciu. Jednotlivé úrovne sú popísané nižšie a majú svoje vlastné triedy. V triede main sa vytvoria inštancie daných tried a v cykle sa pýtame používateľa, čo si želá robiť. V aplikácii využívam údajové štruktúry z cvičení (sú uvedené v UML diagrame). ImplicitSequence využívam hlavne pre rýchlu operáciu sprístupni a vlož, pričom si nepotrebujem pamätať referencie na nasledovníka a predchodcu. MultiWayExplicitHierarchy využívam predovšetkým na to aby si každý údaj pamätal svojich synov a viac-cestná preto, lebo každý vrchol môže mať viacero potomkov. Treap som si zvolil preto, aby práca s tabuľkou bola náročnejšia než pri použití utriedenej implicitnej tabuľky.

# používateľská príručka

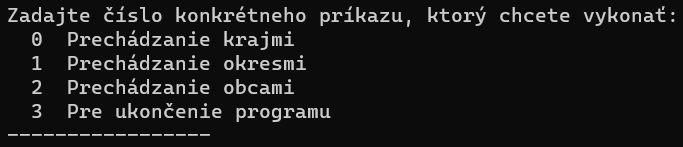
Celý program sa spúšťa v triede *main*, ktorá si vytvára inštancie ostatných tried. Po spustení aplikácie je používateľ uvítaný a má na výber z možností používania aplikácie. Používateľ zadáva celé čísla, tak ako má uvedené pri výbere z daných možností. Čísla neuvedené v možnostiach je používateľ presunutý naspäť na začiatok.

Obrázok, na ktorom je text, písmo, snímka obrazovky

Automaticky generovaný popis

**Obrázok 1 Výber z možností aplikácie**

Vysvetlenie každej z možností postupne: 1. úroveň, prechádzanie načítanou sekvenciou; 2. a 4. prechádzanie hierarchiou; 3. úroveň, vyberanie z tabuľky; ukončenie aplikácie.



**Obrázok 2 Výber danej sekvencie a filtrovanie pomocou predikátu**

Zadaním konkrétneho príkazu má užívateľ na výber z nižšie uvedených predikátov.

Obrázok, na ktorom je text, písmo, snímka obrazovky

Automaticky generovaný popis

**Obrázok 3 Výber predikátu nad 1. úrovňou**

Po vybraní daného predikátu musí používateľ zadať reťazec ktorý sa má spracovať, pri startsWithString musí reťazec začať veľkým písmenom, pri containsStr nemusí.

Obrázok, na ktorom je text, snímka obrazovky, písmo

Automaticky generovaný popis

**Obrázok 4 Prechádzanie hierarchiou**

Po vybraní možnosti prechádzania hierarchiou je používateľ oboznámený s tým kde sa momentálne nachádza a akí sú dostupní synovia na ktorých sa môže presunúť. Používateľ má na výber presun na syna, kde zadá index syna (index je to prvé číslo odľava pri názvu syna), presun na otca alebo zavolanie predikátu nad daným vrcholom, ktorý používateľ vidí úplne na začiatku („Nachádzaš sa na..“).

Obrázok, na ktorom je text, písmo, snímka obrazovky, čierny

Automaticky generovaný popis

**Obrázok 5 Možnosti predikátov pri prechádzaním hierarchiou**

Používateľ má na výber zo všetkých predikátov. Pri výbere predikátu hasType musí používateľ zadať reťazec „kraj“, „okres“ alebo „obec“.

Obrázok, na ktorom je text, písmo, snímka obrazovky

Automaticky generovaný popis

**Obrázok 6 Zvolenie triedenia vyfiltrovaných údajov**

Po vybraní predikátu má používateľ na výber triediť dané údaje podľa vyššie uvedených možností.

Obrázok, na ktorom je text, písmo, snímka obrazovky, čierny

Automaticky generovaný popis

**Obrázok 7 Práca s tabuľkou**

Pri výbere prvej možnosti musí najprv používateľ zadať typ územnej jednotky („kraj“, „okres“, „obec“) a následne celý názov danej územnej jednotky, podľa ktorého sa vyberie daný údaj z tabuľky.

# UML diagram tried

# úroveň

Prvou úlohou bolo načítať údaje o obciach, okresoch a krajoch Slovenskej republiky z priložených súborov do nami zvolených údajových štruktúr. Najskôr bolo nutné si vytvoriť triedu pre samostatnú územnú jednotku. Spomenutá trieda ***UzemnaJednotka*** má nasledujúce atribúty: sortNumber, code, officialTitle, mediumTitle, shortTitle, note a type. V konštruktore sa všetky tieto atribúty inicializujú. Ďalej sú v triede gettery a settery na spomenuté atribúty, metóda **vypis()** vypíše na konzolu kód územnej jednotky a jej názov, metóda **toString()** prevedie všetky atribúty na jeden reťazec a metóda **dajPocetSamohlasok()** vráti počet samohlások názvu územnej jednotky (toto je využité v 4. úrovni). Ďalej som si vytvoril triedu ***Vstup***, ktorá má atribúty *file* typu fstream a *sekvencia* typu ImplicitSequence<UzemnaJednotka>, do ktorého sa uložia vytvorené objekty typu UzemnaJednotka a uložia sa do atribútu. Zvolil som ImplicitSequence pretože si nepotrebujem pamätať referencie na predchodcov alebo nasledovníkov, tým som pamäťovo menej náročný. Daný atribút sa vytvára cez kľúčové slovo new, čiže ho v deštruktore musím vyčistiť metódou clear() a pomocou delete vymazať. V metóde **nacitajUdaje()** si cez parameter pošlem názov súboru z ktorého idem územné jednotky načítavať, následne si otvorím súbor, pokiaľ je otvorený, cyklom si pomocou getline() načítam celý riadok do pomocnej premennej a pomocou stringstream-u si rozdelím daný riadok na reťazce, ktoré sú oddelené bodkočiarkou. Do sekvencie si uložím celý objekt UzemnaJednotka s načítanými a rozparsovanými parametrami. Keď narazíme na koniec súboru, cyklus skončí, súbor sa zatvorí a vrátim si adresu na atribút *sekvencia*. Túto metódu volám v hlavnej triede main a posielam si túto štruktúru do ďalších úrovni aby som si vytváral len ukazovatele na už načítané územné jednotky.

Trieda ***SekvenciaUzemnychJednotiek*** si v atribúte sekvencia typu ImplicitSequence<UzemnaJednotka\*>\* ukladá ukazovatele na už načítané údaje. V metóde nacitajSekvenciu() si ju načíta a vráti.

Trieda ***Algoritmus*** obsahuje metódu **dajDataKtoreSplnajuPredikat()** ktorý má šablónový parameter **Iterator**, aby mu mohli prísť akékoľvek typy iterátorov. Ako parameter mu príde begin a end Iterator a predikát. Tu by si používateľ vedel poslať akýkoľvek predikát a fungovalo by to. Stačí teda poslať metóde začiatočný a koncový iterátor, a predikát, ktorý má daný údaj spĺňať.

Obrázok, na ktorom je diagram, snímka obrazovky, rad, text

Automaticky generovaný popis

Obrázok 8 Diagram aktivít

# úroveň

Trieda ***HierarchiaSlovenska*** reprezentuje hierarchiu územných jednotiek, načítaných z prvej úrovne. Atribúty hierarchia typu MultiWayExplicitHierarchy<UzemnaJednotka\*>\*, ktorá si ukladá ukazovatele na načítané územné jednotky a slovensko typu UzemnaJednotka\* slúži ako pomocný údaj, ktorý reprezentuje vrchol hierarchie. Hierarchia sa načítava v metóde **nacitajHierarchiu()**, kde si na začiatku vložím koreň a následne v cykle vkladám kraje, okresy a obce. Používam na to konkrétne iterátory sekvencií. Pri každom vkladaní kraju si prejdem iterátorom okresy a kontrolujem, či nepatrí daný okres do vloženého kraja, tu sa môžeme spoľahnúť na poradie údajov v vstupných súborov. Kontrolujem, či posledných 5 znakov v atribútu note kraja sa nenachádzajú v atribúte code okresu. Pokiaľ nájdem taký okres, vložím ho ako syna daného kraja a idem obdobným spôsobom kontrolovať obce. Zložitosť načítavania je **O(n)** pretože každý údaj musím zo sekvencie prejsť raz.

Iterátor na prechádzanie hierarchiou je implementovaný ako samostatný Iterator v triede ***MultiWayExplicitHierarchyIterator*** v súbore hierarchy.h z cvičení (viď obrátok 9). V konštruktore sa inicializuje atribút hierarchia zo zaslanej hierarchie z parametra a atribút vrchol od ktorého bude iterátor iterovať. Iterovanie funguje tak, že sa zavolá PostOrderIterator alebo PreOrderIterator z už implementovaných tried z cvičení. Prechod na syna a na otca spočíva len v zmene aktuálneho vrcholu iterátora pomocou metódy accessSon() a accessParent(). Metóda **dajAktualnyVrchol()** vráti aktuálnu pozíciu iterátora (atribút vrchol) a metóda dajData() sprístupni dáta aktuálneho vrcholu.

Obrázok, na ktorom je text, snímka obrazovky, písmo

Automaticky generovaný popis

**Obrázok 9 Iterátor viac-cestnej hierarchie**

Nový predikát hasType malo jednoduchú implementáciu. Bolo treba doplniť triedu UzemnaJednotka o atribút type, v ktorom si pamätám typ územnej jednotky (kraj/okres/obec). Do metódy dajDataKtoreSplnajuPredikat() posielam nový predikát, ktorý vráti true ak je údaj daného typu. Tu je vidno, že doplnenie algoritmu o nový predikát nie je náročné. Treba len doplniť v triede main možnosť výberu daného predikátu.

Do týchto metód stačí poslať Iterator typu MultiWayExplicitHierarchyIterator a algoritmus správne vráti údaje, ktoré spĺňajú daný predikát. Nebolo treba teda meniť nič.

# úroveň

Do aplikácie som pridal tabuľku **Treap** pre uchovávanie ukazovateľov na územné jednotky. V triede ***TabulkaUzemnychJednotiek*** sú atribúty *pomocna* typu ImplicitSequence<UzemnaJednotka\*>\* a *tabulkaTreap* typu Treap<string, UzemnaJednotka\*>\*, ktoré sa v konštruktore inicializujú a v deštruktore vyčistia a vymažú.

Metóda **nacitajTreap()** iterátormi prejde sekvenciu ktorú ideme vložiť do tabuľky, ako kľúč sa vyberie officialTitle getterom UzemnejJednotky a pomocou metódy insertUnsafe() sa vloží do tabuľky. Metóda **insertUnsafe()** je upravené v tom, že nevyhodí výnimku pokiaľ sa nájde údaj s rovnakým kľúčom a taktiež v tom, že kľúče väčšie alebo rovné ako parent idú napravo a menšie naľavo. V metóde **najdiPodlaKlucaVTreap()** vyberám z tabuľky metódou removeUnsafe(), ktorá mi vráti výnimku pokiaľ sa údaj s daným kľúčom v tabuľke nenachádza. Túto výnimku si odchytím a cyklus skončí.

# úroveň

Vytvoril som si triedu ***TriediaciAlgoritmus***, ktorý okrem konštruktora a deštruktora obsahuje metódu **utriedData()**, ktorý slúži ako univerzálny algoritmus, parametrami sú ImplicitSequence šablónového typu DataType a std::function komparátor. V metóde sa zavolá sort() QuickSortu s danými parametrami.

Metóda **porovnajPocetSamohlasok()** pošle metóde utriedData() danú sekvenciu a komparátor, ktorý vráti true ak prvý údaj ma menší počet samohlások ako druhý. Využíva sa tu metóda dajPocetSamohlasok() z triedy UzemnaJednotka. V danej metóde sa prejde každý znak z atribútu officialTitle, premení sa na malé písmeno a skontroluje sa či nepatrí do množiny samohlások.

Metóda **porovnajAbecedne()** si najprv pripraví lokálnu premennú local typu std::locale, ktorý predstavuje lokalizáciu (jazyk a kultúru) pre porovnávanie reťazcov. V tomto prípade je použitá slovenská lokalizácia so znakovou sadou "Slovak\_Slovakia.1250". Do premenných retazecA a retazecB si uložím atribúty officialTitle oboch údajov. Následne sa vytvorí referencia **coll** na šablónový facet **std::collate<char>** získaný z objektu **local**. Facet **std::collate** sa používa na porovnávanie reťazcov v rôznych lokalizáciách. Potom sa volá metóda **compare** objektu **coll** s cieľom porovnať reťazce **retazecA** a **retazecB**. V tomto prípade sa porovnáva oblasť pamäti reťazcov od ich začiatku po ich koniec. Ak je **porovnanie** menšie ako 0, vráti sa **true**, čo znamená, že prvý prvok je "menší" než druhý prvok a má byť radený skôr. Ak je **porovnanie** väčšie alebo rovné 0, vráti sa **false**, čo znamená, že prvý prvok je "väčší" alebo "rovný" druhému prvku a má byť radený neskôr alebo na rovnakej pozícii.