Univerzita Karlova v Praze, Matematicko-fyzikální fakulta

Rudolf Rosa, studijní program Informatika – Programování

Dokumentace k zápočtovému programu na NPRG041 Programování v C++

Cvičící: Mgr. Jan Dědek

RURTextGapper

Program pro automatické nahrazování v textu

Obsah

1. O programu

RURTextGapper je konzolová aplikace pro automatické zpracování textu. Umožňuje vyhledávání a nahrazování slov v textu, buď podle slovníku nebo podle značek v textu. Vstupy jsou ve formě textových souborů, výstup(y) mohou být do souboru a/nebo na konzoli. Veškerá nastavení se provádějí pomocí textového souboru, předaného programu jako parametr.

Program je primárně určen k tomu, aby pomohl učitelům s přípravou cvičení a testů.

2. Programátorská část dokumentace

Program je napsaný objektově v jazyce C++, ve formě konzolové aplikace.

Program se ovládá pomocí textového souboru s volbami, který se předává jako parametr při spuštění; vstupy a výstupy jsou textové soubory, výstupy mohou být i na konzoli. Podrobnější informace o použití programu a jeho ovládání viz Uživatelská část dokumentace.

Zdrojový kód programu je rozdělený do tří souborů, většina kódu je umístěna v souboru `rtg.cpp`, zejména ve třídě `rtg`; soubor `rtg.h` je k němu příslušící hlavičkový soubor – tuto část programu lze tak využít i jako knihovnu. Soubor `main.cpp` obsahuje funkci `main()` – ta zajišťuje předání parametru do programu a jeho spuštění; dále ošetřuje případné chyby a ohlašuje ukončení běhu programu. Dále tedy popisuji pouze kód obsažený v souboru `rtg.cpp`.

Program popisuji po jednotlivých podtřídách třídy `rtg`. Datový model vyplývá z popisu tříd, podrobný popis je součástí kódu (komentáře).

Vyhledávání ze slovníku je implementací klasického algoritmu Aha a Corasickové. Nebudu zde popisovat, jak a proč algoritmus funguje, ale zaměřím se na to, jak je implementován. Algoritmus Aho-Corasick se skrývá především ve třídách `cUzel` a `cText`, dále pak ve třídě `cSlovnik`.

1. cMain

Třída, kterou vytvoří konstruktor třídy `rtg`. Jejím úkolem je spustit práci programu podle nastavení v options a po skončení práce korektně odalokovat paměť. Třídy, které spravují data alokovaná explicitně na haldě (klíčovým slovem `new`), mají definované destruktory, které tato data korektně odstraní.

2. cOptions

Tato třída reprezentuje a spravuje nastavení programu, je vytvořena při spuštění programu a odkaz na ni se předává napříč programem.

Nastavení se ukládá do struktury `map`, klíčem je název dané volby. Volby jsou silně typované, používám proto čtyři `map`y, jeden pro každý typ (`bool`, `char`, `int`, `string`). To umožňuje jednoduché a rychlé načtení voleb i přístup k nim. Pro každý typ se používá jako prefix první znak jeho názvu (tj. `b`, `c`, `i`, `s`). Podrobné informace o syntaxi a významu voleb viz Nastavení programu.

K volbám se (pouze pro čtení) přistupuje pomocí čtyř funkcí `*Opt`, kde `*` nahrazuje znak pro daný typ. Argumentem je název volby, funkce vrací hodnotu volby (díky silnému typování voleb je návratová hodnota již správného typu). Pokud není volba nalezena, hází funkce `string` výjimku, obsahující popis chyby. Ta je standardně zachycena v `main()` funkci programu; při použití programu jako knihovny je zachycení výjimky ponecháno na programátorovi.

3. cText

Hlavní třída v celém programu. Jednoduše řečeno reprezentuje vstupní text a operace nad ním – zajišťuje tedy vyhledávání a nahrazování v textu, včetně ukládání a výpisu vzniklých výstupů.

Konkrétní detaily jsou podrobně komentovány v kódu, zde tedy popisuji především hlavní principy.

Vstupy a výstupy

Text se ve třídě postupně načítá z `ifstream`u soubor, upravený text se postupně ukládá do `stringstream`u `outbuffer`, ze kterého je průběžně vypisován na výstup (tj. dle nastavení do souboru a/nebo na konzoli). `Stringstream nalezBuffer` slouží k dočasnému uložení části vstupu, o kterém se ještě není rozhodnuto, co se s ním stane; význam podrobněji viz dále v textu.

Náhrady

Náhrady, tj. `string`y vyjmuté z textu, se ukládají do `map nahrady`; klíčem je samotné slovo, hodnotou je počet opakování (začíná tedy na 1 a při každém dalším nahrazení téhož slova se inkrementuje). V textu je možné číslovat vynechaná místa, poslední použité číslo určuje `pocitadloNahrad`. Je možné si zvolit, že stejné náhrady mají sdílet stejné číslo, pak se tato čísla ukládají do `map`u `nahradyCisla`, a pouze pokud pro dané slovo tato struktura dosud neobsahuje žádnou hodnotu, inkrementuje se počítadlo a hodnota se uloží pod klíčem odpovídajícím náhradě.

Jak přesně má náhrada vypadat se určuje v nastavení a implementace je poměrně přímočará.

Vyhledávání

Vyhledávání samotné probíhá výrazně odlišně podle toho, zda se vyhledává podle slovníku nebo z textu. Pomocné metody a metody realizující některé podružné volby jsou ale sdílené při obou způsobech vyhledávání – jde zejména o metody zajišťující práci s buffery a náhradami, jejich formátování a vypisování. Jsou sice důležité pro chod programu, ale nejsou nijak překvapivé a jsou bohatě komentované v kódu, nebuduje zde tedy podrobněji popisovat.

Pro spuštění vyhledávání se vždy volá metoda `provestHledani()`, která má ale několik odlišných přetížených variant. Pro vyhledávání z textu se volá varianta bez parametrů. Pro hledání podle slovníku se volá varianta s jedním parametrem, která je pouhou obálkou pro dvouparametrickou variantu, kterou vnitřně (i opakovaně) volá.

Vyhledávání z textu

Vyhledávání z textu je jednodušší variantou. Střídají se zde dva stavy, dané hodnotou `boolean`u `mameNalez`. Začínáme pochopitelně ve stavu `false`, kdy nemáme žádný nález a znaky ze vstupu rovnou kopírujeme do `outbuffer`u. Jakmile najdeme znak pro začátek slova, přejdeme do stavu `true` a znaky ze vstupu nyní kopírujeme do `nalezBuffer`u. Při nalezení znaku pro konec slova se vrátíme zpět do stavu `false`. Při přechodu mezi stavy vyprázdníme příslušný buffer a na konci vyhledávání vypíšeme výstupy. Provedení samotné náhrady je poměrně přímočarý proces.

Vyhledávání ze slovníku

Klíčová je zde položka `uzel`, která ukazuje na aktuální uzel (odpovídá tedy aktuálnímu stavu vyhledávání). Metoda `provestHledani()` postupně čte znaky ze vstupního proudu a předává je aktuálnímu uzlu, který určuje následující uzel (viz cUzel: Vyhledávání).

Významné jsou také položky `znak` a `znakTemp`, obě reprezentující znak načtený ze vstupu. Položka `znakTemp` je neupravený znak, přesně tak jak je na vstupu. Tato položka je v rámci třídy globální, neboť je nutné k ní přistupovat i z jiných míst. Naproti tomu `znak` je lokální pracovní položka – provádějí se na něm různé úpravy (např. převod na lowercase, pokud je nastaveno case-insensitive vyhledávání).

Potud samotné vyhledávání – zajímavější je nahrazování, které je složitější.

Nahrazování podle slovníku

Datová položka `posledniNalezeneUzel` ukazuje na uzel, ve kterém jsme naposledy zaznamenali výskyt hledaného slova. Pokud má nulovou hodnotu, znamená to, že poslední výskyt byl již zpracován (anebo že jsme dosud nic nenašli). Aktualizuje se, pokud v aktuálním uzlu končí nějaké slovo, případně když v něm končí nepřímo (viz cUzel, položka `zpatkyNalez`).

Nahrazovaní se spustí v okamžiku, kdy máme nalezené nějaké slovo, ale měli bychom se ve vyhledávacím stromě vrátit – v tu chvíli se volá hlavní nahrazovací metoda `nahradSlovo()`, která podle aktuální situace a nastavení provede nahrazení (volá ji přímo aktuální uzel). Tato metoda se také zavolá při nalezení slova, pokud je nastavené dohledávání konce slov – v takovém případě nezáleží na tom, co následuje na vstupu, dokud to není znak pro konec slova.

Metoda nejprve rozdělí znaky v `outbufferu` (tj. znaky, které byly načtené od posledního výstupu = výpisu tohoto bufferu). Začátek bufferu (až do začátku nalezeného slova) jde bez dalších úprav na výstup, nad ním není třeba provádět žádné operace. Následuje nalezené slovo – je také vzato z bufferu a ne z uzlu, který na něj ukazuje (odtud se bere pouze jeho délka), neboť náhrady mají obsahovat slova tak, jak jsou na vstupu, což může obnášet např. jinou velikost písmen. Zbytek pak tvoří `nalezBuffer`, jehož obsah je uložen do `stringu` `nalezB`. Jde o již načtené znaky, které následují za koncem nalezeného slova (a tedy je často prázdný, při některém nastavení dokonce vždy) – tyto znaky je třeba dále zpracovat.

Nyní může přijít na řadu dohledání konce slova (je-li tato volba aktivní). To znamená, že nález nebude tvořit jen slovo, nalezené podle slovníku, ale bude končit až tam, kde se v textu nachází nejbližší znak pro konec slova (tedy ve vnitřní reprezentaci mezera, na kterou se přeloží znaky znamenající začátek nebo konec slova). Toto zajišťuje metoda `dohledatKonec()`, která ze zdroje do bufferu `buf` dočte znaky až do konce slova.

Následuje vytvoření a vypsání náhrady (včetně uložení do seznamu provedených náhrad `nahrady`), které má na starosti metoda `nahradaText()` a několik dalších metod, jejichž funkce je zřejmá a není komplikovaná. Využívá se zde mnoho voleb z nastavení, pomocí kterých je možné určit parametry náhrady.

Pokud byl `nalezBuffer` prázdný, je zpracování nálezu hotové a řízení se vrací do volající metody, které pokračuje se ve čtení vstupního textu a vyhledávání, které opět začne ze `startu` (po nahrazení slova nedává smysl, aby vyhledávání bylo v jiném než startovním stavu).

Pokud ale `nalezBuffer` prázdný nebyl, je třeba ještě zpracovat znaky, které obsahoval. Ty již nejsou ve vstupním proudu, ale ani ještě nebyly zpracovány. Nemohu ani použít uzel, ve kterém naposledy skončilo vyhledávání, neboť ten označuje stav po načtení nalezeného slova a až poté těchto znaků, který nemusí být shodný se stavem při načtení těchto znaků se začátkem ve `start`u. Používám se zde proto takovýto trik: z těchto znaků vytvořím nový proud, nad kterým rekurzivně zavolám metodu `provestHledani()`. Protože již nad těmito znaky jednou běžela, nesmí se znovu zapisovat návštěvy uzlů ve vyhledávacím stromě, což zajišťuje parametr `hlavniCyklus`, nastavený tentokrát na `false` (propaguje se do metody `cUzel::pismeno()`). Takto získám správný uzel, ze kterého se má v hlavním cyklu pokračovat ve vyhledávání, a ten metoda vrátí jako výstupní hodnotu. (Samozřejmě je teoreticky možné, aby rekurze proběhla do větší hloubky, ale její konečnost je zajištěna konečností znaků v `nalezBuffer`u.)

4. cSlovo

Reprezentuje jedno slovo slovníku. Je uloženo přesně tak, jak bylo na vstupu; výjimkou je případný znak pro začátek slova, přidaný na konec slova – v případě, že se mají vyhledávat pouze celá slova (resp. slova končící koncem slova) a konce slov se nemají dohledávat.

Slovo se umí vypsat včetně počtu výskytů (pochopitelně až po provedení vyhledávání); počet výskytů do slova zapisuje výhradně uzel, ve kterém slovo končí.

5. cSlovnik

Reprezentuje slovník, obsahuje slova a slovníkový strom.

Na kořen slovníkového stromu ukazuje ukazatel `koren`; `start` ukazuje buď na kořen, nebo na přidaný vrchol, který je potomkem kořene a do kterého se z kořene přechází znakem pro začátek slova (pokud je nastavena volba `b_zacatek_zacslova`). Vyhledávání začíná vždy v uzlu `start`, neboť například začátek textu je zároveň začátkem slova.

Na všechny uzly stromu také vede odkaz ze struktury `uzlyPodlePater`, která pro každé patro stromu (nechť strom roste nahoru) obsahuje `vector` se všemi uzly, které na daném patře leží. To je důležité pro průchod stromem při počítání výskytů a při dealokaci uzlů.

Stavba stromu probíhá postupným čtením slov (do šířky). Začne se ve startovacím uzlu a postupně se do něj přidávají první písmena všech slov; každé slovo má v sobě uloženou informaci o tom, kolik znaků z něj je již zpracováno a ve kterém uzlu momentálně končí. Jakmile je celé načteno, je připojeno k uzlu, ve kterém končí.

Slovník má dále na starosti napočítání výskytů slov po skončení vyhledávání. Nejprve projde strom od nejvyšších pater (opět do šířky) a zaznamená počty navštívení jednotlivých uzlů. Aby byly korektně zaznamenány i nálezy slov, která jsou podřetězci jiných slov, při počítání návštěv se tento počet zároveň přičte do uzlu `zpatky` – protože podřetězec je vždy kratší než řetězec, má uzel při zaznamenávání počtu nálezů již definitivní informaci o svém počtu navštívení.

6. cUzel

Objekt této třídy reprezentuje jeden uzel ve slovníkovém stromě. V uzlech je vyjádřena struktura stromu, zajišťují také hlavní úkony pří vyhledávání.

Pro dopředný postup ve stromě je zde struktura `naslednici`, které ukazuje na všechny uzly, do kterých je možné z daného uzlu postoupit (přes písmeno, které je klíčem v `mapu`). Pokud přijímáme písmeno, kterým nepokračuje žádné slovo procházející tímto uzlem, pošle uzel toto písmeno do nižších pater stromu, konkrétně uzlu `zpatky`.

Pro vyhledávání je důležitý také ukazatel `zpatkyNalez` – je-li nenulový, ukazuje na uzel, ve kterém končí slovo, které je prefixem slova vedoucího do aktuálního uzlu. Pokud jsme tedy v aktuálním uzlu, můžeme zároveň ohlásit výskyt slova v uzlu `zpatkyNalez`.

Stavba stromu

Uzly vznikají při stavbě stromu. V případě potřeby je uzel vytvořen, přičemž získává informaci o svém uzlu `zpatky`. Uzel `zpatkyNalez` si uzel dohledá sám (stačí jít postupně směrem ke kořeni stromu podle ukazatelů `zpatky`).

Uzel je vytvořen svým předchůdcem – tj. uzlem, ze kterého do něj vede cesta přes některé písmeno. Pokud se má v uzlu přijímat písmeno, které se zde ještě nepřijímá, vytvoří si uzel nového následníka, do kterého přes toto písmeno povede cesta; korektně mu nastaví jeho datové položky a zařadí ho mezi své následníky.

Vyhledávání

Většina metod v části označené v kódu jako „vyhledávání“ pouze zajišťuje read-only přístup k datovým položkám uzlu. Metoda `pismeno()` je tou metodou, která se na uzlu volá během vyhledávání – slouží pro započítání návštěvy uzlu (je-li toto žádoucí, což není například v případě, že neprobíhá vlastní vyhledávání, ale je teprve stavěn strom) a především jako brána k hlavní vyhledávací metodě, `pismenoInterni()`.

Metoda `pismenoInterni()` je tou metodou, která implementuje hlavní část vyhledávacího algoritmu. Uzel, který odpovídá aktuálnímu stavu vyhledávání, dostane jako vstup této metody následující písmeno na vstupu a má vrátit následující stav. Pokud toto písmeno přijímá (tedy existuje slovo, které prochází tímto uzlem a pokračuje tímto písmenem), přejde se do příslušného uzlu. Pokud jej nepřijímá, standardní algoritmus vyhledá uzel, do kterého lze přejít – postupně se vrací po ukazatelích `zpatky`, dokud nenajde uzel, který písmeno přijímá; ten pak dává (pomocí téže metody) následující stav (tato cesta stromem se zastaví nejpozději v kořeni stromu). Pokud ale chceme v textu nahrazovat, volá se metoda `nahradSlovo()`, která v závislosti na nastavení provede různé operace (viz `nahradSlovo()`) a vrací následující stav, který se propaguje jako návratová hodnota metody.

Finální napočítání výskytů jednotlivých slov bylo popsáno v popisu třídy `cSlovnik`.

7. cUzelKoren

Potomek třídy `cUzel`, představuje kořen stromu složeného z objektů třídy `cUzel`. Liší se od nich tím, že při vyhledávání (metoda `pismenoInterni()`) nepřeposílá nenalezený znak dál po stromě (uzlu `zpatky`), ale zastaví rekurzi v sobě (neboť ve skutečnosti žádné další uzly ve stromě nejsou, i když formálně ukazatel `zpatky` v kořeni ukazuje na kořen samotný).

3. Uživatelská část dokumentace

1. Funkce programu

Základní informace

Program je určený především pro učitele, kteří chtějí pro studenty připravit cvičení typu „gapped text“ – text, ve kterém jsou vynechaná slova či jiné úseky (pokud zde mluvím o slovech, má vždy na mysli obecně jakékoli úseky textu). Místo nahrazování slov lze provést i pouhé vyhledávání bez nahrazování (s počítáním výskytů).

Program pracuje pouze s obyčejnými textovými soubory, text je proto potřeba uložit jako prostý text (obvyklou příponou je `txt`). Zpracovaný text bude také vrácen do textového souboru; případné formátování poté proveďte v textovém editoru.

Nastavení se provádí pomocí textového souboru – více informací najdete ve druhé kapitole. Vzorové nastavení je pro příklad přibaleno k programu včetně souborů s textem pro nahrazování.

Slova pro nahrazování lze určit dvěma způsoby – z textu anebo ze slovníku.

Vyhledat slova z textu

Prvním způsobem je označení slov přímo v textu. Ve volbách si vymyslíte znak, kterým označíte začátek slova pro vynechání (dobře se píše například paragraf – „§“ – bývá umístěn vedle klávesy Enter). Před každé slovo pak napíšete tento znak – například takto:

```
§Chci vynechat první, §čtvrté a osmé slovo §tohoto textu.
```

Program automaticky vyhledá tato slova a nahradí je podle nastavení – například číslem a libovolným textem (obvyklé bývá například „_____“), nebo přesmyčkou původního slova; je také možné zachovat původní slovo a jen přidat kolem něj další znaky, což se může hodit například pro vytvoření textu v HTML formátu.

Pokud nechcete nahrazovat jednotlivá slova, nastavte si ve volbách i znak pro konec slova – program pak bude nahrazovat celý text mezi znakem pro začátek a znakem pro konec.

Vyhledat slova podle slovníku

Druhou možností je dát programu ještě jeden soubor – slovník. Jde o obyčejný seznam slov, která se mají v textu najít a nahradit, jedno slovo na jeden řádek. To se hodí třeba v případě, že máte seznam nepravidelných sloves a chcete je v textu všechna najít a vynechat, nebo třeba i jen nějak označit.

Je možné zadat do slovníku jen začátky slov a zapnout příslušnou volbu – program pak vyhledá všechna slova, které takto začínají.

Při vyhledávání podle slovníku je samozřejmě možné nastavit nahrazování stejným způsobem jako při vyhledávání z textu.

Výstupy

Veškeré výstupy lze poslat do textového souboru, anebo na konzoli (to se hodí například při práci v konzoli na unixu). Samozřejmě, že pokud Vás některý výstup nezajímá, nemusíte ho posílat nikam a prostě ho nedostanete.

Pokud provádíte nahrazování v textu, hlavním výstupem asi bude text s provedenými náhradami. Jak mají náhrady vypadat se určuje v nastavení (viz níže).

Dalšími možný výstupy jsou slovník a seznam provedených náhrad. Oba výstupy jsou podobné. Slovník lze získat pouze při vyhledávání podle slovníku a na výstupu lze získat i slova, která nebyla v textu vůbec nalezena; slova jsou vypisována ve stejném pořadí jako ve vstupním slovníku. Náhrady naproti tomu lze vypsat vždy, když bylo provedeno nahrazování, a obsahují právě ta slova, která byla v textu skutečně nahrazena. Formát je opět ve tvaru jedno slovo na řádek, za ním je v závorce uvedený počet opakování. Náhrady jsou seřazené podle abecedy[1], volitelně je lze vypsat náhodně setříděné.

Spuštění programu

Program lze spustit například tak, že v něm otevřete soubor s nastavením (Soubor – Otevřít v programu – najít `rtg.exe`). Podrobnější informace jsou uvedené dále v textu.

2. Nastavení programu (volby, options)

Nastavení programu se provádí pomocí souboru „options“. Soubor options s výchozím nastavením (`doptions.txt`) je součástí programu. Soubor má jednoduchou syntaxi:

- `;cokoliv` – řádek začínající středníkem se chápe jako komentář; za středníkem musí být aspoň jeden další znak, jeden samotný středník se chápe jako zakomentování celé následující řádky
- `b_cokoliv=0` – boolean volba, hodnotou je `0` (ne) nebo `1` (ano) (zde `0`)
- `c_cokoliv=a` – char volba, hodnotou je jeden znak (zde `a`)
- `s_cokoliv=ahoj` – string volba, hodnotou je řetězec znaků (zde `ahoj`)
- `i_cokoliv=42` – int volba, hodnotou je integer (zde `42`)

Nastavení je rozdělené na několik oddílů, nicméně na pořadí ani na komentářích nezáleží. Záleží na velikosti písmen, soubor nesmí obsahovat žádné znaky navíc. Všechny volby by měly být explicitně uvedeny, jinak program může zahlásit chybu („`Neni definovana hodnota parametru`“). Uvedení neexistující volby není chápáno jako chyba a nemá vliv na běh programu.

Volby, které mají až na prefix shodný název, spolu vždy úzce souvisí. V původním souboru nastavení je u všech voleb uveden krátký komentář s jejich významem.

Následuje popis jednotlivých nastavení, rozdělený podle jednotlivých oddílů. Výchozí hodnoty viz soubor `doptions.txt`.

1 Nejde o zcela korektní abecední řazení, ale o výchozí C++ řazení, které se řídí ASCII kódy znaků – proto jsou velká písmena před malými a neanglické znaky za anglickými.

Soubory

Vstupní a výstupní soubory. Vstupní soubory musí existovat, výstupní budou vytvořeny nebo přepsány. Soubory lze zadávat relativní i absolutní adresou – tedy např. `text.txt` nebo `C:\Documents and Settings\R\Dokumenty\text.txt`

- `s_in_text` – vstupní text, se kterým má program pracovat
- `s_in_slovnik` – vstupní slovník (pokud se má vyhledávat podle slovníku); slova (obecně libovolné řetězce, které se mají vyhledávat) oddělená prázdnými řádky
- `s_out_text` – soubor, do kterého se vypíše text po provedení případných nahrazení
- `s_out_slovnik` – soubor, do kterého se vypíše výsledek vyhledávání
- `s_out_nahrady` – soubor, do kterého se vypíše výsledek nahrazování

Slovník

Nastavení týkající se vstupního slovníku, který lze předat jako soubor a podle kterého lze hledat slova, a výstupního slovníku (= výsledek vyhledávání podle vstupního slovníku). Výsledek vyhledávání obsahuje na každém řádku nalezené slovo a za ním v závorce počet výskytů.

- `b_slovnik` – vyhledávat podle slovníku (jinak: vyhledávat v textu slova ohraničená speciálními znaky, viz níže). Následující volby mají smysl, pouze pokud je zde nastaveno ano (1)
- `b_cout_slovnik` – vypsat výsledek vyhledávání na konzoli
- `b_out_slovnik` – vypsat výsledek vyhledávání do souboru
- `b_vypsat_slova_bez_vyskytu` – vypsat i slova s počtem výskytu 0

Text

Nastavení týkající se výstupního textu (pokud bylo provedeno nahrazovaní, je výstupem text po provedení nahrazování; v opačném případě je totožný se vstupním textem).

- `b_cout_text` – vypsat text na konzoli
- `b_out_text` – vypsat text do souboru

Vyhledávání

Nastavení týkající se vyhledávaní je pouze jedno:

- `b_case_insensitive` – provést vyhledávání bez ohledu na velikost písmen; tato volba pracuje spolehlivě pro anglické znaky v ASCII kódování, pro ostatní znaky a kódování s velkou pravděpodobností nebude fungovat správně[2]

2 Nicméně to lze obejít vypnutím této volby a ručním vložením všech očekávaných podob daného slova do slovníku – pokud tedy místo jediného slova „čau“ vložíte „čau“, „Čau“ a „ČAU“, bude to fungovat správně s téměř 100% pravděpodobností (problém pochopitelně nastane při nekonzistenci v kódování vstupních souborů).

Nahrazování

Jde o hlavní úkol programu, z toho důvodu je zde nejvíce voleb. Rozdělím je proto do několika skupin.

Zda nahrazovat

Jediná volba, určující, zda se vůbec provede nahrazování. Pokud má hodnotu ne (`0`), následující volby nemají význam.

- `b_nahradit`

Začátky a konce slov

Několik voleb, určujících, jak se chovat k začátkům a koncům slov. Znaky vyjadřující začátek/konec slova jsou interně převedeny na mezery; je třeba s tím počítat, pokud se snažíte vyhledávat řetězce obsahující některý z těchto znaků. Pokud mají všechny boolean volby hodnotu `0`, na začátky a konce slov se nebere ohled. Různé volby se logicky uplatní jen v některých situacích – například znaky vymezující začátek a konec slova se pochopitelně neuplatní při hledání podle slovníku.

- `s_konec_slova` – znaky, které v textu znamenající konec slova (mezera a znaky pro konec řádku jsou dodefinovány napevno)
- `b_zacatek_zacslova` – nález musí začínat na začátku slova, jinak nález shody není ohlášen
- `b_konec_konslova` – nález (náhrada) musí končit na konci slova
- `b_konec_dohledat` – dohledat konec slova (co se má udělat, pokud nález nekončí na konci slova – zda se má konec slova dohledat, anebo zda se má nález ignorovat)
- `c_nahrada_zac` – znak uvozující v textu slovo pro nahrazení (nezbytné při vyhledávání slov pro nahrazení z textu, tedy nikoli ze slovníku – označuje slova, která se mají nahradit)
- `c_nahrada_kon` – znak ukončující v textu slovo pro nahrazení (při vyhledávání z textu, pokud nechci slovo automaticky ukončovat koncem slova)

Čím nahrazovat

Řetězec, kterým má být nález nahrazen, lze určit pomocí této skupiny voleb. Formát náhrady lze symbolicky popsat takto:

<s_nahrada_pred><číslo><s_nahrada_mezi><původní slovo><s_nahrada_za>

Libovolná z těchto částí může být prázdná: části před, mezi a za lze ponechat nevyplněné, číslo a původní slovo je určeno boolean volbami.

- `s_nahrada_pred` – první část řetězce
- `b_nahrada_cislovat` – zda má být součástí řetězce číslo
- `nahrada_cislovat_pismenkem` – místo čísla použít písmenko
- `b_nahrada_cislovat_slova` – číslování se týká slov, nikoli prázdných míst; opakované výskyty stejného slova tedy budou mít shodná čísla

- `s_nahrada_mezi` – druhá část řetězce
- `b_nahrada_slovo` – zda se má v náhradě zachovat původní slovo
- `b_nahrada_presmycka` – zda se má ze slova udělat jeho přesmyčka (bylo-li zachováno)
- `s_nahrada_za` – poslední část řetězce

Výstupy

Náhrady se vypisují ve stejném formátu jako slovník; jsou ale abecedně seřazené a mohou být ovlivněny např. zapnutým dohledáváním konce slova – v takovém případě slovník vypíše prefix nahrazených slov, zatímco náhrady vypíšou skutečně nahrazená slova.

- `b_out_nahrady` – vypsat náhrady do souboru
- `b_cout_nahrady` – vypsat náhrady na konzoli
- `b_nahrady_random` – při výpisu náhodně setřídit náhrady

3. Spuštění programu

Program se spouští s jedním parametrem, kterým je adresa souboru s volbami, které se mají použít. Bude tedy fungovat správně, pokud tento soubor otevřete tímto programem (v OS Windows například takto: Soubor – Otevřít v programu – najít program `rtg.exe`). Můžete si také soubor s volbami ukládat s vlastní příponou, pro kterou si zadefinujete jako výchozí program RURTextGapper, pak bude použití ještě pohodlnější (standardní způsob otevření, např. dvojklikem).

Program je samozřejmě možné spouštět z příkazového řádku, pokud mu jako první parametr předáte soubor s volbami. Dobře lze také využít soubory dávkových příkazů (batch, sh).

Program po spuštění vypíše na konzoli případné vyžádané výstupy, text „hotovo“ a čeká na stisknutí klávesy Enter. Výstupy lze ovlivnit pomocí voleb, ostatní výstupy na konzoli a čekání na Enter lze potlačit použitím druhého parametru; jako druhý parametr lze použít libovolný řetězec.

Pokud program korektně skončí, vrací návratovou hodnotu `0`.

Pokud dojde k chybě, vrací `1` a na konzoli vypíše informace o chybě (pokud nebyl použit druhý parametr, ten potlačuje i tyto výstupy).

Pokud je program spuštěn se špatným počtem parametrů, vždy vypíše na konzoli hlášení o chybě a čeká na stisk klávesy Enter; jako návratový kód vrací `2`.