**NPRG041** – 2022/23 ZS – Cvičení MS – **Malý úkol C04**

# Počítadlo

Cílem tohoto domácího úkolu je naprogramovat jednoduchou aplikaci, která dokáže spočítat základní statistiky nad textem v přirozeném jazyce, např. detekovat počet znaků, slov, vět atp. Vstupní text můžeme předat přes standardní vstup, stejně jako jej musíme umět načíst ze vstupních souborů. Spočítané statistiky naopak potřebujeme umět vypsat na standardní výstup nebo do výstupních souborů.

Požadované vstupy a výstupy, se kterými chceme pracovat, bychom mohli specifikovat např. pomocí argumentů předaných naší aplikaci přes příkazovou řádku. Této problematice se však v rámci tohoto úkolu už znovu věnovat nebudeme. Odevzdávat tedy budete jen kód umožňující vlastní zpracování vstupů a generování výstupů, nikoli `main` funkci a související kód, který toto zpracování bude nad zamýšlenými vstupy a výstupy vyvolávat a řídit.

Konkrétně se očekává implementace dvou tříd, a to třídy `Counter`, která nabídne všechny potřebné funkce na zpracování vstupů i výstupů, a třídy `Statistics`, která umožní uchování hodnot jednotlivých detekovaných statistik. V obou případech je potřeba přesně dodržet předepsané názvy tříd, stejně jako dále popsané chování a rozhraní.

Pro zpracování vstupních souborů použijeme rozhraní `std::ifstream` a funkci `std::getline`, pro práci s výstupními soubory použijeme `std::ofstream`. Náš kód však musíme napsat tak, abychom byli schopni pracovat i s jakýmkoli jiným vstupním resp. výstupním streamem v podobě `std::istream` resp. `std::ostream`. Vše potřebné najdeme ve standardních knihovnách `<iostream>`, `<fstream>` a `<string>`.

V praktické rovině budeme chtít implementovat následující dvě dvojice funkcí, všechny realizujeme jako veřejné statické členské funkce (metody) zmíněné třídy `Counter`. Při jejich návrhu využijeme mechanismu přetěžování, kdy odpovídající si funkce záměrně pojmenujeme stejně, lišit se pak budou jen počtem a/nebo typem parametrů.

- `static bool process(const std::string& filename, Statistics* data)`: načteme vstupní text ke zpracování z uvedeného vstupního souboru určeného jeho názvem, detekované hodnoty zapíšeme do již připravené instance třídy statistik
- `static bool process(std::istream& is, Statistics* data)`: totéž, jen vstupní text načteme z již otevřeného obecného vstupního streamu
- `static bool print(const std::string& filename, const Statistics* data)`: hodnoty všech detekovaných statistik zapíšeme do uvedeného výstupního souboru
- `static bool print(std::ostream& os, const Statistics* data)`: totéž, jen hodnoty zapíšeme do již otevřeného obecného výstupního streamu

Pokud jde o strukturu vstupního textu, máme následující předpoklady. Celý text se skládá z libovolného počtu vět. Ty jsou vždy povinně ukončeny právě jedním interpunkčním znakem tečka `.`, otazník `?` nebo vykřičník `!` a případně i odděleny libovolným počtem mezer. Libovolný počet mezer pak může být i před případnou první větou nebo za případnou poslední. Věta obsahuje libovolně namíchaná slova nebo čísla, minimálně však jedno z nich. Slova a čísla jako taková jsou vždy vzájemně oddělena alespoň jednou mezerou. Slovo obsahuje výhradně jen písmena anglické abecedy. Číslo pak obsahuje výhradně jen dekadické číslice a případně tečku `.`, pokud šlo o desetinné číslo. V tom případě musí být před i za desetinnou tečkou alespoň jedna číslice. Se zápornými čísly pracovat nebudeme. Text také může obsahovat libovolný počet konců řádků, umístěny mohou být jen mezi větami, slovy nebo čísly. Pro detekci jednotlivých písmen a číslic se očekává použití standardních funkcí `isdigit` a `isalpha`.

Cílem zpracování vstupního textu je detekovat základní statistiky, konkrétně celkový počet řádků, vět, slov, čísel, písmen, číslic, mezer a symbolů (otazník, vykřičník a tečka, a to včetně těch uvnitř desetinných čísel) a dále také celkový součet hodnot celých čísel a odděleně součet hodnot čísel desetinných. Pro tyto součty použijeme datové typy `long` resp. `double`, případné přetečení kontrolovat nebudeme. Jednotlivá nalezená čísla budou vždy v rozsahu nejvýše `int` resp. `float`.

Pro uložení hodnot všech zjišťovaných statistik navrhneme již zmíněnou třídu `Statistics`. Obsahovat bude výhradně jen potřebné datové položky, ty budou přístupné veřejně. Obejdeme se tedy bez jakýchkoli metod. Po celou dobu běhu programu budeme mít k dispozici jen jedinou instanci této třídy (bude vytvořena

lokálně ve funkci `main`), detekované hodnoty v ní budeme postupně inkrementovat s každým zpracovaným vstupem. Jinými slovy detekované hodnoty budeme akumulovat přes všechny vstupy dohromady. Kdykoli se objeví požadavek na vypsání statistik na výstup, vypíšeme v daný okamžik aktuální obsah.

Nalezené statistiky vypíšeme ve formátu ilustrovaném v následujícím ukázkovém výstupu. Na každý řádek uvedeme jeden konkrétní údaj, a to jeho stručný název, dvojtečku, mezeru a vlastní hodnotu. Každý řádek ukončíme pomocí konce řádku `std::endl`. Pro vypsání každé dílčí hodnoty použijeme operátor `<<` pro přímý zápis do streamu (jinými slovy tentokráte funkci `std::to_string` používat nebudeme). Uvedené pořadí údajů je nutné dodržet.

```
Lines: 3
Sentences: 4
Words: 30
Numbers: 7
Letters: 163
Digits: 20
Spaces: 37
Symbols: 7
Integers: 85
Floats: 23.5
```

Podobně jako v minulých úkolech, i tentokráte bude každá naše funkce vracet hodnotu typu `bool` pro informování o případných chybách. Konkrétně budeme vracet `true` v případě úspěchu, naopak `false` v případě chyby, aniž bychom jakkoli jednotlivé příčiny chyb chtěli rozlišovat nebo o nich vypisovat jakékoli chybové hlášky. V obecné rovině můžeme předpokládat, že vstupní texty budou z hlediska popsané struktury korektní. Detekovat a ošetřovat ale musíme všechny chybové situace týkající se práce se samotnými soubory.

Veškerý kód samozřejmě rozdělte do jednotlivých modulů s hlavičkovými soubory. Obě předepsané třídy `Counter` a `Statistics` musíte umístit do hlavičkového souboru s názvem `Counter.h`. Odevzdejte všechny vytvořené zdrojové soubory (`*.cpp` a `*.h`) kromě hlavního souboru `Main.cpp` obsahujícího implementaci funkce `main` a související kód, který jste si za účelem experimentálního otestování předepsané funkcionality patrně vytvořili. Tento soubor totiž už je součástí předem připraveného testu, bude zkompilován společně se všemi zbývajícími soubory projektu a průběh celého testu bude řídit.

Hlavním záměrem úkolu je ověřit schopnost práce se soubory a streamy obecně. Předpokládáme dodržení všech postupů, které jsme se už naučili, stejně jako požadavků, které na úkoly máme. Specificky se však zaměřte zejména na následující aspekty. Opět si dejte pozor na to, abyste neopakovali stejné nebo podobné fragmenty kódu. Pro nejrůznější počty samozřejmě používáme typ `size_t`. Zpracování vstupního textu je potřeba zvládnout na jediný průchod. Nepoužívejte žádné kontejnery (kromě samotného řetězce `std::string`).

Při parsování hodnot celých a desetinných čísel nezapomeňte na detekci všech možných chybových situací, jakkoli na ně nenarazíme. Funkce `std::stoi` a `std::stof` však nevolejte z hlavního kódu přímo, ale zabalte je do vlastních funkcí, protože nechcete ošetřování těchto chybových situací přenášet na volajícího.

Pokud budete chtít navrhnout nějaké další vlastní interní funkce a nelze čekat, že by se daly využít univerzálně (jinými slovy jsou svázané čistě s námi řešeným problémem počítadla), nenavrhujte je jako globální funkce, ale jako privátní statické členské funkce naší třídy `Counter`. Přesně z tohoto důvodu totiž celá tato třída existuje, tedy aby na jednom místě zapouzdřila vše, co k realizaci našeho počítadla potřebujeme. Naopak dvojice funkcí pro parsování číselných hodnot v podobě globálních funkcí klidně zůstat může, protože univerzálně použitelné jsou.

Na úplný závěr dodejme, jak v rámci ladění ukončit zadávání vstupního textu přes standardní vstup. V případě Linuxu postačí kombinace kláves `CTRL+D`, na Windows je potřeba na samostatném řádku `CTRL+Z` a následně ještě `Enter`. Program jako celek naši interní logickou chybovou hodnotu bude opět kvůli ReCodExu ignorovat a vždy vrátí kód `0`.

Pokud během ladění programu narazíte na nečekané pády, příčina nejspíše bude v tom, že se pokoušíte přistoupit za konec pole nebo kontejneru (což platí i pro řetězce `std::string`) nebo chcete použít reference nebo ukazatele na objekty nebo položky, které už v daném okamžiku neexistují.