NPRG041 – 2024/25 ZS – Cvičení MS – **Malý úkol C03**

Počítadlo

Předmětem tohoto domácího úkolu je jednoduchá aplikace, která dokáže spočítat základní statistiky nad textem v (téměř) přirozeném jazyce, např. detekovat počet znaků, slov nebo vět. Vstupní text můžeme získat přes standardní vstup nebo vstupní soubory. Spočítané statistiky naopak potřebujeme umět vypsat na standardní výstup nebo do výstupních souborů.

Požadované vstupy a výstupy, se kterými budeme chtít 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. Soustředit se totiž chceme jen na kód umožňující vlastní zpracování vstupních textů a generování výstupů, nikoli na `main` funkci a související kód, který toto zpracování bude nad zamýšlenými vstupy a výstupy vyvolávat a řídit.

Konkrétně se očekává implementace třídy `Counter`, která nabídne všechny potřebné funkce na zpracování vstupů i vytvoření výstupů, a struktury `Statistics`, která umožní uchování jednotlivých detekovaných statistik. V obou případech je potřeba přesně dodržet předepsané názvy typů, 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 void 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 struktury statistik
- `static void process(std::istream& stream, Statistics* data)`:
  totéž, jen vstupní text načteme z již otevřeného obecného vstupního streamu
- `static void print(const std::string& filename, const Statistics* data)`:
  hodnoty všech detekovaných statistik zapíšeme do uvedeného výstupního souboru
- `static void print(std::ostream& stream, 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 rozpoznání písmen a číslic se očekává použití standardních funkcí `int std::isdigit(int ch)` a `int std::isalpha(int ch)` z knihovny `<cctype>`. Pro parsování příslušné číselné hodnoty z textového řetězce pak použijeme funkci `int std::stoi(const std::string& str, std::size_t* pos)`, resp. `std::stof`. Obě jsou z knihovny `<string>` a mají, alespoň pokud jde o první dva pro nás potřebné parametry, stejné rozhraní i stejné principy chování. Musíme si dát pozor na to, abychom správně detekovali všechny možné chybové situace, pokud by parsování neproběhlo úspěšně. To znamená, že musíme ochytávat výjimky `std::invalid_argument` i `std::out_of_range` a za problém považujeme i situaci, kdy by za jinak legitimní číselnou hodnotou následovalo ještě cokoli dalšího.

Cílem zpracování vstupního textu je detekovat základní statistiky, konkrétně celkový počet řádků (`lines`), vět (`sentences`), slov (`words`), čísel (`numbers`), písmen (`letters`), číslic (`digits`), mezer (`spaces`)

a symbolů (`symbols` pro otazník, vykřičník a tečku, a to včetně těch uvnitř desetinných čísel) a dále také celkový součet hodnot celých čísel (`integers`) a odděleně součet hodnot čísel desetinných (`floats`). 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 zjišťovaných statistik navrhneme již zmíněnou strukturu `Statistics`. Obsahovat bude výhradně jen potřebné datové položky (ty navíc budou přístupné veřejně a musí být respektovány jejich výše uvedené názvy a dokonce i pořadí), obejdeme se tedy bez jakýchkoli metod. Nově zkonstruovaná instance této struktury musí mít všechny datové položky implicitně inicializovány na nulové hodnoty. Při práci s nimi pak budeme jejich hodnoty aktualizovat inkrementálně, nikoli nahrazovat hodnotami novými. To umožní sdílet jednu instanci struktury (bude např. vytvořena lokálně ve funkci `main`) napříč vícero vstupy, a tedy detekované hodnoty akumulovat i přes více postupně zpracovaných vstupů dohromady.

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. Uvedené pořadí a názvy 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
```

V obecné rovině můžeme předpokládat, že vstupní texty budou z hlediska popsané struktury korektní. Explicitně však musíme kontrolovat špatně formovaná slova a čísla, tedy že by započaté slovo obsahovalo číslice (např. `a15`) nebo naopak započaté číslo písmena (např. `3.14a`). Stejně tak ohlídáme situace, kdy není možné získat hodnotu jinak korektního čísla kvůli přetečení (např. `9876543210`). Detekovat a ošetřovat dále musíme všechny chybové situace týkající se práce se samotnými soubory, konkrétně při neúspěšném pokusu o otevření vstupního nebo výstupního souboru.

Ve všech popsaných chybových situacích vyhodíme textovou výjimku (tedy výjimku typu `const char*`, nikoli `std::string`) s odpovídající hláškou podle následujícího přehledu:

- `Unable to open input file`
- `Unable to open output file`
- `Invalid word detected`
- `Invalid integer number detected`
- `Invalid floating point number detected`

Veškerý kód rozdělte do jednotlivých modulů s případnými hlavičkovými soubory. Definice předepsané třídy `Counter` a struktury `Statistics` je potřeba umístit do hlavičkového souboru s názvem `Counter.h`. Odevzdejte všechny vytvořené zdrojové soubory (`*.cpp` a `*.h`) kromě 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 v ReCodExu, obsahuje direktivu `#include "Counter.h"`, bude zkompilován společně se všemi zbývajícími odevzdanými soubory projektu a průběh celého testu bude řídit.

Hlavním záměrem úkolu je ověřit schopnost práce se soubory a streamy obecně. Předpokládá se dodržení všech postupů, které jsme se už naučili, stejně jako obecných požadavků, které na úkoly máme. Specificky se zaměřte zejména na následující aspekty. Opět si dejte pozor na to, abyste neopakovali stejné nebo podobné fragmenty kódu. Pro nejrůznější počty i nadále 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. Předpokládá se použití funkcí `std::stoi` a `std::stof`, nevolejte je však z hlavního kódu přímo, zabalte je do vlastních funkcí. Jednoduše totiž nechcete odpovědnost za ošetřování těchto nízkoúrovňových 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.

Proces zpracování každého řádku je potřeba zvládnout na jeden průchod. S návrhem případných dalších pomocných funkcí spíše šetřete, přehnaná dekompozice by celé věci jen uškodila. Třída počítadla nesmí mít žádné datové položky (protože jejich trvalé uchování by nedávalo smysl), struktura statistik naopak nesmí mít žádné metody.

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`.

Pokud během ladění programu v ReCodExu narazíte na chybový signál (což je něco jiného než návratový kód), došlo při běhu programu k natolik závažnému problému, že musel být ukončen. Příčina v našem případě nejspíše bude v tom, že se pokoušíte přistoupit před začátek nebo 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.