

Argumenty II

V rámci tohoto úkolu naprogramujeme rozšíření funkcionality předchozího úkolu umožňujícího základní detekci argumentů předaných aplikaci z příkazové řádky. Rozšíření bude spočívat v tom, že nově začneme rozlišovat mezi jednotlivými povolenými typy navázaných hodnot u hodnotových přepínačů, stav přítomnosti a případné hodnoty všech očekávaných přepínačů uložíme do pomocné datové struktury a tu v závěrečném přehledu vypíšeme na standardní výstup. Všechny ostatní aspekty úkolu zůstávají beze změny.

Doposud jsme navázané hodnoty hodnotových přepínačů jen vypisovali, jejich konkrétní datový typ jsme proto vůbec řešit nepotřebovali. Jinými slovy pro nás vše bylo řetězcem `std::string`. Nově začneme rozlišovat hodnoty typu řetězec (tedy stávající chování), celé číslo (myslí se tím `int` s číslem zadaným v desítkové soustavě) a číslo s pohyblivou desetinnou tečkou (`float`).

Každý hodnotový přepínač má předem pevně dáno, jaký typ hodnoty předpokládá. Opět se očekává, že vše v tomto smyslu napevno zadrátujeme přímo do kódu. Konkrétně očekáváme následující chování: přepínače `-r`, `--red`, `-g`, `--green`, `-b` a `--blue` vyžadují celé číslo, přepínače `-a` a `--alpha` číslo desetinné.

Pro parsování příslušné číselné hodnoty z textového řetězce použijeme funkci `int std::stoi(const std::string& str, std::size_t* pos)`, resp. `std::stof`, obě mající 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. To se totiž v naší situaci nepřípouští.

V rámci procesu zpracování vstupních argumentů beze změny zachováme vypisování všech informačních zpráv, které jsme předpokládali v předchozím úkolu. Nově budeme vypisovat i následující zprávy:

- Value <value> is not a valid integer number!
- Value <value> is not a valid floating point number!

Platí, že vždy vypíšeme standardní hlášku o nalezení hodnoty navázané na daný hodnotový přepínač, teprve za ní případně jako další zpráva bude následovat výše uvedená informace o chybě. Pokud k parsovací chybě nedošlo, žádnou další hlášku nevypisujeme.

Informaci o stavu nalezení všech očekávaných přepínačů, navázaných hodnot v případě těch hodnotových, stejně jako hodnoty všech nalezených samostatných hodnot uložíme do instance struktury, kterou za tímto účelem navrhujeme. Použít k tomu opravdu musíme strukturu `struct`, ne třídu `class`. Všechny její datové položky budeme vnímat jako veřejné, a budeme k nim tedy přistupovat napřímo. Jinými slovy nebudeme přístup k nim schovávat za žádné členské funkce (metody). Samostatné hodnoty budeme ukládat se zachováním jejich vzájemného pořadí do vnořeného vektoru řetězců.

Instanci naší struktury vytvoříme přímo ve funkci `main`, naše hlavní funkce `process_arguments` pak převezme referenci na ní a všechny detekované informace do struktury průběžně během procesu zpracování argumentů bude zaznamenávat.

Platí přitom, že některé přepínače jsou párové. To znamená, že nám nezáleží na tom, která konkrétní varianta se objevila, stačí jen jedna z nich. Jinými slovy jde navzájem jen o aliasy. Podle očekávání budeme konkrétně pracovat s těmito propojenými dvojicemi: `-t` a `--transparent`, `-r` a `--red`, `-g` a `--green`, `-b` a `--blue`, `-a` a `--alpha`.

Současně platí, že se jednotlivé přepínače mohou ve stupních argumentech objevovat i opakovaně. Speciálně u těch hodnotových to pak znamená, že poslední požadavek vyhrává. A to ať už vedl k validní hodnotě, nebo nikoli. Pokud tedy např. nejprve detekujeme korektní přepínač `-r 255` a následně nevalidní `-r string`, výsledný stav bude takový, že přepínač `r` se bude jevit jako nedetekovaný, a tedy bez hodnoty.

Dodejme, že průběžně vypisované informační zprávy to nijak neovlivňuje, jde jen o to, jaké informace budou v naší struktuře uloženy. Nově nalezený hodnotový přepínač a jeho hodnotu si tedy zapamatujeme jen tehdy, když byla korektně naparsována. V opačném případě zajistíme, že případné již dříve detekované údaje o tomto přepínači z naší struktury vymažeme.

Z hlediska dekompozice problému do vhodně navržených funkcí nadále předpokládáme využití stávající funkce `process_value_option`, nově pak přidáme jednotlivé funkce `process_string_option`, `process_int_option` a `process_float_option`. Pojmenování je opět možné libovolně změnit.

Před koncem běhu programu ještě zařídíme, že veškerý obsah struktury pro kontrolu vypíšeme, a to opět na standardní výstup. Celkový výstup programu pak bude obsahovat postupné informační zprávy z detekce, následně uměle vložíme jeden prázdný řádek pro oddělení, teprve pak realizujeme zmíněný kontrolní přehled. Používat k tomu budeme následující šablony zpráv:

- Flag option <name|...> is <disabled>
- Flag option <name|...> is <enabled>
- Value option <name|...> is <disabled>
- Value option <name|...> is <enabled> and associated with value <value>
- Standalone value <value>

Každou zprávu jako obvykle ukončíme koncem řádku. Do lomených závorek se jménem uvedeme jediné jméno přepínače (např. `x`) nebo obě u těch párových (např. `r|red`). Pořadí výstupu bude následující: `x`, `y`, `grayscale`, `t|transparent`, `r|red`, `g|green`, `b|blue` a `a|alpha`. Při vypisování číselných hodnot předpokládáme použití konstruktu `std::to_string`. Na úplném konci pak vypíšeme samostatné hodnoty, pokud nějaké byly. Neočekávané přepínače se do tohoto výstupu vůbec nedostanou, ani ve vlastní struktuře jsme je nijak neukládali.

Opět odevzdejte všechny vytvořené zdrojové soubory (`*.cpp` a `*.h`). Složitější pasáže kódu doplňte o stručné komentáře (postačí jednořádkové `// ...`). Nadále kvůli `ReCodExu` vždy vrátíme jen návratový kód 0, mechanismus předávání chyb pomocí návratových hodnot jednotlivých funkcí však je i nadále nutné implementovat. Hlavním cílem úkolu je naučit se parsovat číselné hodnoty a pracovat se strukturami.

Opět si dejte pozor, abyste neopakovali podobné nebo dokonce stejné fragmenty kódu. To se týká i míst, kde naše funkce `process*_option` voláme, jde tedy i o eleganci jejich rozhraní. Nadále platí, že jména přepínačů můžeme napevno zadrátovat jen na jednom místě (případně odděleně pro krátké a dlouhé přepínače, ne však opakovaně). Jinými slovy je nutné zařídit, že samotná existence daného přepínače, určení jeho varianty z pohledu flagových a hodnotových, stanovení typu navázané hodnoty v případě těch hodnotových i specifikace místa k uložení detekovaných údajů v rámci naší struktury musí vše být pro každý jednotlivý přepínač řešeno jen na jednom jediném místě.

Pokud se rozhodnete rozvinout celý jmenný prostor nebo jen jeho část pomocí konstrukce `using`, nikdy tak nedělejte v hlavičkovém souboru.