Implementační dokumentace k projektu do IPP 2017/2018
Jméno a příjmení: Martin Studený
Login: xstude23

Analyzátor kódu v IPPcode18:

Parser je implementován jako jeden obrovský automat, ve kterém jsou další vnořené automaty. Zpracování argumentů je provedeno pomocí funkce `getopt`. Tahle funkce umožňuje poměrně jednoduché ošetření různých kombinací parametrů a počtu argumentů. Po tomto ošetření se volá funkce `load_input`, která načte vstupní text ze standartního vstupu a prochází jej řádek po řádku. Na jednotlivé řádky je pak volána funkce `automat`.

Tato funkce představuje hlavní automat. Dochází v něm k rozlišování jednotlivých instrukcí – prochází pouze první část řádku. V momentě, kdy "přečte" nějakou instrukci, jsou zavolány další automaty. Neexistující instrukce je ošetřena právě v tomto hlavním automatu

Další automaty jsem rozdělil podle toho, jaké argumenty očekávají. Podle toho jsem poté například určoval, jestli má instrukce správný počet argumentů, jestli jsou správného typu a v podstatě všechny kontroly, které byly potřeba udělat.

Následně se zavolají funkce pro výpis XML. Zde se výpis opět rozlišuje v závislosti na tom, jaké parametry má jednotlivá instrukce obsahovat. Jelikož jedna taková funkce je společná pro více instrukcí, musel jsem těmto funkcím předávat parametrem jednak jednotlivé argument, tak i jakého typu jsou proměnné a kód instrukce, abychom věděli, jakou instrukci zrovna vypisujeme.

Pro výpis XML používám knihovnu `xml.etree.ElementTree`. Nevýhodou je, že kvůli tomu musím procházet vstupní soubor dvakrát. Nepřišel jsem totiž na způsob, jak v případě chyby nevypisovat XML. To sice nevadí, ale přesto jsem to implementoval tímhle způsobem.

V prvním běhu tedy kontroluju, jestli je správná syntaxe a až poté, když už vím, že XML budu vypisovat, tak začnu.

Jelikož některé znaky v XML nejsou povoleny, musel jsem každý string, který by se v XML měl objevit, předělat tak, aby místo nepovolených znaků byly escape sekvence. To řeším tím způsobem, že pokaždé, když je to potřeba, tak zavolám funkci `change_string`. Ta tento string již převede.

Součástí implementace je i rozšíření, které vypisuje i statistiky. To ale v závislosti na délce mého pravděpodobně nefunguje úplně přesně. Musel jsem totiž kód upravovat na hodně místech a dost možná jsem na něco zapomněl.

Interpret XML reprezentace kódu:

Argumenty interpretu jsou ošetřeny pomocí funkce `parse_args`. Pro načítání XML souboru jsem použil knihovnu `xml.etree.ElementTree`.

Nejprve zkontroluji kořen, jestli má správný počet atributů, jestli jsou všichni jeho potomci instrukce a zda mají správné atributy.

V skriptu opět využívám automat, který po zjištění `opcode` rozliší, na kterou funkci máme dále pokračovat. Implementace je podobná analyzátoru, protože opět máme k dispozici několik funkcí, které jsou společné pro více instrukcí, ale očekávají jiné argumenty. To ale platí pouze pro analýzu vstupního XML souboru. V druhém průchodu již má každá instrukce vlastní funkci.

V zadaném XML souboru nezáleží na pořadí `order` ani argumentů u jednotlivých instrukcí. Tahle část byla v mém kódu dost náročná. Především u skoků na návěští. V těchto místech vznikaly problémy, jelikož procházím XML tím způsobem, že mám jeden velký cyklus, který hledá instrukce od první až po poslední.

Problém nastal v tom, že v momentě, kdy jsem potřeboval někam skočit, například z instrukce číslo na číslo 2, musel jsem ukončit současný cyklus a vytvořit nový, který již bude počítat od instrukce následující za návěštím, na které jsme skočili. Tuto problematiku jsem však, s využitím různých přepínačů, úspěšně vyřešil.

V interpretu procházím vstupní soubor opět dvakrát. Jelikož interpretu může být nevalidní XML soubor, musel jsem v prvním průchodu zkontrolovat, zde XML je ve správném formátu.

V druhém průchodu už máme pro každou instrukci speciální funkci, která všechno potřebné kontroluje. Funkce jsou opět zbytečně dlouhé – často se v kódu totiž vyskytují úseky kódu, které jsou pro spoustu instrukcí téměř totožné. Jelikož jsem to už měl tímhle způsobem rozdělané, tak jsem to i dokončil.

Testovací rámec:

V testovacím skriptu není implementovaný přepínač `--recursice`. Vše ostatní by ve skriptu mělo fungovat správně.

Nejprve se zkontroluji argumenty, kde počítám i s přepínačem `--recursice`, ale v případě jeho použití skript ukončuji s chybou.

Ve složce, kde máme hledat testy, hledám jakékoliv soubory s příponou `.src`, další přípony zatím nekontrolujeme. Nejprve si domyslíme, že opravdu existují a pokusíme se je najít. V případě, že se to nepovede, soubory vytvoříme přesně tak, jak máme.

HTML se vypisuji až na konci skriptu. Hlavička se vypisuje podle speciální funkce. Ta je stále stejná a mění se jen počty testů. Pro výpis podrobnějších informací o testech slouží druhá funkce, která se zavolá při každém provádění testu, ale zatím nic nevypisuje. Pouze přidává html kód do proměnné. Výpis nastane až úplně na konci za hlavičkou.

Skript nevytváří žádný XML soubor. Výstup je třeba si přesměrovat.