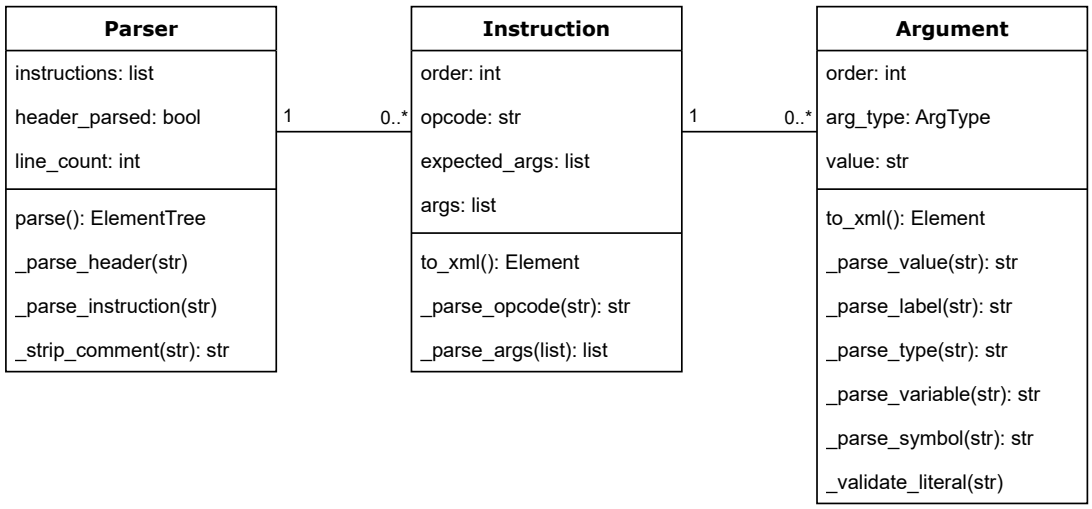
Implementační dokumentace k 1. úloze do IPP 2023/2024

Jméno a příjmení: Milan Vodák
Login: xvodak07

# 1 Diagram tříd




# 2 Popis částí

## 2.1 `Parser`

Třída `Parser` reprezentuje samotný syntaktický analyzátor. Většinu analyzační logiky zastřešuje metoda `parse()`, která čte řádek po řádku ze standardního vstupu a zpracovává je. Jako první z řádku prostřednictvím metody `_strip_comment()` odstraní komentáře a také přebytečné bílé znaky a následně řádek zpracuje jako hlavičku, nebyla-li tato nalezena; nebo jako instrukci. Prázdné řádky či řádky obsahující pouze komentář jsou ignorovány. Dostane-li se analyzátor na konec standardního vstupu zatímco stále nenašel hlavičku, ukončí program s návratovým kódem 23.

O zpracování hlavičky a instrukce se starají pomocné metody analyzátoru `_parse_header()`, resp. `_parse_instruction()`. Po nalezení správné hlavičky je nastaven příznak `header_parsed` a na další řádky už je nahlíženo jen jako na instrukce. Zpracování instrukce je delegováno na třídu `Instruction` (viz Sekce 2.2) a získaná instance se vloží do seznamu doposud načtených instrukcí, tj. instančního atributu `instructions`. V této fázi se analyzátor také stará o ošetření výjimek, jež mohou být vyvolány během konstrukce instrukcí a jejich argumentů. To spočívá ve výpisu hlášky s číslem dotčeného řádku na standardní chybový výstup a v ukončení programu s odpovídajícím návratovým kódem.

Pro generování výstupní XML reprezentace jsem použil knihovnu `xml.etree.ElementTree`. Po zpracování celého standardního vstupu se vytvoří kořenový element `<program>` a iterativně se instrukce ze seznamu `instructions` převádějí na XML elementy, které jsou dovnitř kořenového elementu vkládány. Nakonec metoda `parse()` vrátí XML strom zpracovaného programu.

## 2.2 `Instruction`

Instance třídy `Instruction` odpovídá jedné instrukci vstupního programu a uchovává v sobě pořadí instrukce, operační kód, očekávané typy argumentů a seznam skutečně načtených argumentů. Konstruktor třídy předá postupně kontrolu pomocným metodám `_parse_opcode()` a `_parse_args()`. Hotovou instrukci je možné převést na XML element voláním metody `to_xml()`.

Instrukční sada je reprezentována slovníkem `INSTRUCTION_SET`, jehož záznamy se skládají z operačního kódu jako klíče a ze seznamu typů argumentů, které instrukce očekává.

Po validaci obdrženého operačního kódu za použití regulárního výrazu[1] je z instrukční sady vybrán záznam odpovídající instrukce a následně je instanci nastaven operační kód (velkými písmeny) a očekávané typy argumentů. Na základě těchto typů se následně provádí konstrukce argumentů ze slov načtených za operačním kódem (viz Sekce 2.3) a ty jsou přidávány do seznamu `args`.

## 2.3 `Argument`

Argument sestává z jeho pořadí v rámci instrukce, typu vyjádřeného výčtovým typem `ArgType` a hodnoty, která bude vypsána do výstupního XML. Obdobně jako u instrukce, převod na XML reprezentaci zajišťuje metoda `to_xml()`. Konstruktor třídy `Argument` požaduje na vstupu typ argumentu, který je očekáván, a na základě tohoto typu je provedeno ověření správnosti načtené hodnoty a případně její úprava do výstupního formátu.

Očekává-li se návěští, typ, nebo proměnná, validace probíhá jednoduše pomocí regulárních výrazů. V případě symbolu může být argument buď proměnná nebo konstanta, což se rozhoduje podle části identifikátoru před znakem `@` a následně je patřičně upřesněn typ atributu. Jedná-li se o konstantu, metoda `_validate_literal()` regulárními výrazy ověří správnost druhé části zápisu. V případě nesprávně zapsaného argumentu či výskytu argumentu neočekávaného typu se vyvolá výjimka `ParserError` s konkrétní chybovou zprávou.

[1] Výskyt jiného znaku než velkých či malých písmen na místě, kde se očekává operační kód, je považován za syntaktickou chybu, která vede na ukončení programu s návratovým kódem 23. Neúspěšné vyhledání operačního kódu v instrukční sadě vyvolá výjimku `InvalidOpcodeError`.