

Zadání projektu z předmětu IPP 2023/2024

Zbyněk Křivka, Radim Kocman

E-mail: krivka@fit.vut.cz

1 Základní charakteristika projektu

Navrhněte, implementujte a dokumentujte dva programy pro analýzu a interpretaci nestrukturovaného imperativního jazyka IPPcode24. K implementaci vytvořte odpovídající stručnou programovou dokumentaci. Projekt se skládá ze dvou úloh a je individuální.

První úloha se skládá z programu v jazyce Python 3.10 (viz sekce 3) spouštěného pomocí skriptu `parse.py` a dokumentace k tomuto programu (viz sekce 2.1). Druhá úloha se skládá z programu v jazyce PHP 8.3 (viz sekce 4) integrovaného do poskytnutého rámce (spouštěného pomocí `interpret.php`) a dokumentace k vaší implementaci (viz sekce 2.1 a upřesnění v sekci 4).

2 Požadavky a organizační informace

Kromě implementace obou programů a vytvoření dokumentace je třeba dodržet také řadu následujících formálních požadavků. Pokud některý nebude dodržen, může být projekt hodnocen nula body! Pro kontrolu alespoň některých formálních požadavků lze využít skript `is_it_ok.sh` dostupný v *Module* předmětu IPP.

Termíny:

připomínky¹ k zadání projektu do 13. února 2024;

fixace zadání projektu od 14. února 2024;

odevzdání první úlohy v úterý **12. března** 2024 do 23:59:59;

odevzdání druhé úlohy v úterý **16. dubna** 2024 do 23:59:59.

Dodatečné informace a konzultace k projektu z IPP:

- *E-Learning* (Moodle) předmětu IPP včetně Často kladených otázek (*FAQ*) a Souborů k projektu.
- *Fórum* v *E-Learning* (Moodle) IPP pro ak. rok 2023/2024, témata *Projekt.**.
- Zbyněk Křivka (garant projektu): dle konzultačních slotů (viz webová stránka) nebo po dohodě e-mailem (uvádějte předmět začínající "IPP:"), viz <http://www.fit.vut.cz/person/krivka>. Další cvičící najdete na kartě předmětu.
- Dušan Kolář (garant předmětu; jen v závažných případech): po dohodě e-mailem (uvádějte předmět začínající "IPP:"), viz <http://www.fit.vut.cz/person/kolar>.

¹Naleznete-li v zadání nějakou chybu či nejasnost, dejte prosím vědět na *Fóru* předmětu nebo e-mailem na krivka@fit.vut.cz. Validní připomínky a upozornění na chyby budou oceněny i bonusovými body.

Pokud máte jakékoliv dotazy, problémy či nejasnosti ohledně tohoto projektu, tak po přečtení *FAQ* a *Fóra* (využívejte i možnosti hledání na *Fóru*), neváhejte napsat na *Fórum* (u obecného problému, jenž se potenciálně týká i vašich kolegů) či kontaktovat garanta projektu, cvičícího (v případě individuálního problému), nebo nouzově garanta předmětu, kdy do předmětu vždy uveďte na začátek řetězec "IPP:". Na problémy zjištěné v řádu hodin až jednotek dní před termínem odevzdání některé části projektu nebude brán zřetel. Začněte proto projekt řešit s dostatečným předstihem.

Forma a způsob odevzdání: Každá úloha se odevzdává individuálně prostřednictvím StudIS do předmětu IPP (odevzdání e-mailem je bodově postihováno) a odevzdáním stvrzujete výhradní autorství skriptů i dokumentace.

V aktivitě „Projekt - 1. úloha v Pythonu 3.10“ odevzdáte archiv pro první úlohu (skript `parse.py`; včetně dokumentace tohoto skriptu). Po termínu odevzdání 1. úlohy bude otevřeno odevzdání archivu v aktivitě „Projekt - 2. úloha v PHP 8.3“ pro druhou úlohu (obsah adresáře `student` v rámci *ipp-core*; včetně dokumentace).

Každá úloha bude odevzdána ve zvláštním archivu, kde budou soubory k dané úloze zkomprimovány programem ZIP, TAR+GZIP či TAR+BZIP do jediného archivu pojmenovaného `xlogin99.zip`, `xlogin99.tgz`, nebo `xlogin99.tbz`, kde `xlogin99` je váš login. Velikost každého archivu bude omezena informačním systémem (pravděpodobně na 2 MB). Archiv nesmí obsahovat speciální či binární spustitelné soubory, ani námi poskytnutý rámec *ipp-core*. Názvy všech souborů a adresářů mohou obsahovat pouze písmena anglické abecedy, číslice, tečku, pomlčku a podtržítka (nepřibírajte skryté adresáře, typicky začínající tečkou). **Skripty úlohy (resp. u 2. úlohy obsah složky `student`) budou umístěny v kořenovém adresáři odevzdaného archivu.** Po rozbalení archivu na serveru *Merlin* (v případě 2. úlohy rozbalíme archiv do adresáře `student` našeho rámce *ipp-core*) bude možné skript(y) spustit. Archiv smí obsahovat v rozumné míře adresáře (typicky pro vaše moduly/jmenné prostory, vlastní testy a pomocné skripty nebo povolené knihovny², které nejsou nainstalovány na serveru *Merlin*).

Hodnocení: Výše základního bodového hodnocení projektu v předmětu IPP je **maximálně 20 bodů**. Navíc lze získat maximálně 5 bonusových bodů za kvalitní a podařené řešení některého z rozšíření nebo kvalitativně nadprůměrnou účast na *Fóru* projektu apod.

Hodnocení jednotlivých skriptů: `parse.py` až 7 bodů (z toho až 1 bod za dokumentaci a úpravu zdrojových kódů skriptu `parse.py`); `interpret.php` až 13 bodů (z toho návrh, dokumentace a úprava zdrojových textů skriptů až **4 body**). Dokumentace je však hodnocena maximálně 50 % ze sumy hodnocení funkčnosti dané úlohy (tedy v případě neodevzdání žádného funkčního skriptu v dané úloze bude samotná dokumentace hodnocena 0 body). Body za každou úlohu včetně její dokumentace se před vložením do StudIS zaokrouhlují na desetiny bodu.

Skripty budou spouštěny na serveru *Merlin* příkazem: `interpret skript parametry`, kde `interpret` bude `python3.10` nebo `php8.3`, `skript` a `parametry` závisí na dané úloze. Hodnocení většiny funkčnosti bude zajišťovat automatizovaný nástroj. Kvalitu dokumentace, komentářů, návrh a strukturu zdrojového kódu budou hodnotit cvičící.

Vaše skripty budou hodnoceny nezávisle na sobě, takže je možné odevzdat například jen 2. úlohu bez odevzdání 1. úlohy.

Podmínky pro opakující studenty ohledně případného uznání hodnocení loňského projektu najdete v *FAQ* na *Moodle* předmětu.

²Pro 2. úlohu budou povolené knihovny k dispozici ve složce `vendor`.

Registrovaná rozšíření: V případě implementace některých registrovaných rozšíření za bonusové body bude odevzdaný archív obsahovat soubor **rozsireni**, ve kterém uvedete na každém řádku identifikátor jednoho implementovaného rozšíření³ (řádky jsou ukončeny unixovým koncem řádku, tj. znak s dekadickou ASCII hodnotou 10). V průběhu řešení mohou být zaregistrována nová rozšíření úlohy za bonusové body (viz *Fórum*). Nejpozději do termínu pokusného odevzdání dané úlohy můžete na *Fórum* zasílat návrhy na nová netriviální rozšíření, která byste chtěli navíc implementovat. Cvičící rozhodne o přijetí/nepřijetí rozšíření a hodnocení rozšíření dle jeho náročnosti včetně přiřazení unikátního identifikátoru. Implementovaná rozšíření neidentifikovaná v souboru **rozsireni** nebudou hodnocena.

Pokusné odevzdání: Pro zvýšení motivace studentů pro včasné vypracování úloh nabízíme koncept nepovinného pokusného odevzdání. Výměnou za pokusné odevzdání do uvedeného termínu (cca týden před finálním termínem) dostanete zpětnou vazbu v podobě zařazení do některého z pěti rozmezí hodnocení (0–10 %, 11–30 %, 31–50 %, 51–80 %, 81–100 %). Bude-li vaše pokusné odevzdání v prvním rozmezí hodnocení, máte možnost osobně konzultovat důvod, pokud jej neodhalíte sami. U ostatních rozmezí nebudou detailnější informace poskytovány.

Pokusné odevzdání bude relativně rychle vyhodnoceno automatickými testy a studentům zaslána orientační informace o správnosti pokusně odevzdané úlohy z hlediska části automatických testů (tj. nebude se jednat o finální hodnocení; proto nebudou sdělovány ani body či přesnější procentuální hodnocení). Využití pokusného termínu není povinné, ale jeho nevyužití může být negativně vzato v úvahu v případě reklamace hodnocení projektu.

Formální požadavky na pokusné odevzdání jsou totožné s požadavky na finální termín a odevzdání se bude provádět do speciálních aktivit „Projekt - Pokusné odevzdání 1. úlohy“ do **5. března 2024** a „Projekt - Pokusné odevzdání 2. úlohy“ do **9. dubna 2024**. Není nutné zahrnout dokumentaci, která spolu s rozšířeními pokusně vyhodnocena nebude.

2.1 Dokumentace

Implementační dokumentace (dále jen dokumentace) musí být stručným a uceleným průvodcem **vašeho způsobu řešení** skriptů 1., resp. 2. úlohy. Bude vytvořena ve formátu **PDF** nebo **Markdown** (viz [4]). Jakékoliv jiné formáty dokumentace než PDF či Markdown⁴ (přípona **md**) budou ignorovány, což povede ke ztrátě bodů za dokumentaci. Dokumentaci je možné psát buď česky, slovensky (s diakritikou, formálně čistě), nebo anglicky (formálně čistě).

Dokumentace bude popisovat celkovou filozofii návrhu, interní reprezentaci, způsob a váš specifický postup řešení (např. řešení sporných případů nedostatečně upřesněných zadáním, konkrétní řešení rozšíření, případné využití návrhových vzorů, implementované/nedokončené vlastnosti). Dokumentaci je vhodné doplnit např. o UML diagram tříd (povinný u 2. úlohy), navržený konečný automat, pravidla vámi vytvořené gramatiky nebo popis jiných formalismů a algoritmů. Nicméně **nesmí obsahovat ani částečnou kopii zadání**.

Vysázená dokumentace 1. úlohy popisující skript **parse.py** by neměla přesáhnout 1 stranu A4 (tj. rozsah přibližně 1 až 2 normostrany). U 2. úlohy (popisující skript **interpret.php**) pak nepřesáhněte vysázené 2 strany A4 (nepočítaje vhodně velký UML diagram). Doporučení pro sazbu: 10bodové písmo Times New Roman pro text a Courier pro identifikátory a skutečně krátké úryvky zajímavého kódu (či zpětné apostrofy v Markdown); nevkládejte žádnou zvláštní úvodní stranu, obsah ani závěr. V rozumné míře je vhodné používat nadpisy první a případně druhé úrovně (12bodové a 11bodové písmo Times New Roman či **##** a **###** v Markdown) pro vytvoření logické struktury dokumentace.

³Identifikátory rozšíření jsou uvedeny u konkrétního rozšíření tučně.

⁴Bude-li přítomna dokumentace v Markdown i PDF, bude hodnocena verze v PDF, kde je jistější sazba.

Nadpis a hlavička dokumentace⁵ bude na prvních třech řádcích obsahovat:

Implementační dokumentace k %cislo%. úloze do IPP 2023/2024

Jméno a příjmení: %jmeno_prijmeni%

Login: %xlogin99%

kde %jmeno_prijmeni% je vaše jméno a příjmení, %xlogin99% váš login a %cislo% je číslo dokumentované úlohy.

Dokumentace bude v kořenovém adresáři odevzdaného archívu a pojmenována `readme1.pdf` nebo `readme1.md` pro první úlohu a `readme2.pdf` nebo `readme2.md` pro druhou úlohu.

V rámci dokumentace bude hodnocena i funkční/objektová dekompozice a komentování zdrojových kódů. Není-li to krystalicky jasné ze samotného identifikátoru funkce/modulu/třídy/parametru a případné typové anotace/nápovědy, doplňte vhodný komentář o účelu apod.

2.2 Programová část

Zadání projektu vyžaduje implementaci dvou skriptů⁶, které mají parametry příkazové řádky a je definováno, jakým způsobem manipulují se vstupy a výstupy. Skript nesmí spouštět žádné další procesy či příkazy operačního systému. Veškerá chybová hlášení, varování a ladicí výpisy směřujte pouze na standardní chybový výstup, jinak pravděpodobně nedodržíte zadání kvůli modifikaci definovaných výstupů (ať již do externích souborů, nebo do standardního výstupu). Jestliže proběhne činnost skriptu bez chyb, vrací se návratová hodnota 0 (nula). Jestliže dojde k nějaké chybě, vrací se chybová návratová hodnota větší jak nula. Chyby mají závazné chybové návratové hodnoty:

- 10 - chybějící parametr skriptu (je-li třeba) nebo použití zakázané kombinace parametrů;
- 11 - chyba při otevírání vstupních souborů (např. neexistence, nedostatečné oprávnění);
- 12 - chyba při otevření výstupních souborů pro zápis (např. nedostatečné oprávnění, chyba při zápisu);
- 20 – 69 - návratové kódy chyb specifických pro jednotlivé skripty;
- 99 - interní chyba (neovlivněná integrací, vstupními soubory či parametry příkazové řádky).

Pokud zadání níže nestanoví jinak, veškeré vstupy a výstupy jsou v kódování UTF-8. Pro účely projektu z IPP musí být na serveru *Merlin* ponecháno implicitní nastavení `locale`⁷, tj. `LC_ALL=cs_CZ.UTF-8`.

Jména hlavních skriptů jsou dána zadáním. Pomocné skripty nebo knihovny budou mít příponu dle zvyklostí v daném programovacím jazyce (`.py` pro Python 3 a `.php` pro PHP 8). Vyhodnocení skriptů bude prováděno na serveru *Merlin* s aktuálními verzemi interpretů (dne 1. 2. 2024 bylo na tomto serveru nainstalováno `python3.10`⁸ verze 3.10.11 a `php8.3`⁹ verze 8.3.2).

⁵Anglické znění nadpisu a hlavičky dokumentace najdete v *FAQ* na *Moodle* předmětu

⁶Tyto skripty jsou aplikace příkazové řádky neboli konzolové aplikace.

⁷Správné nastavení prostředí je nezbytné, aby bylo možné používat a správně zpracovávat parametry příkazové řádky v UTF-8. Pro správnou funkčnost je třeba mít na UTF-8 nastaveno i kódování znakové sady konzole (např. u programu PuTTY v kategorii *Window Translation* nastavíte *Remote character set* na UTF-8). Pro změnu ovlivňující aktuální sezení lze využít unixový příkaz `export LC_ALL=cs_CZ.UTF-8`.

⁸Upozornění: Na serveru *Merlin* je třeba dodržet testování příkazem `python3.10`, protože pouhým `python` se spouští stará nekompatibilní verze! Python 3.x není zpětně kompatibilní s verzí 2.x!

⁹Upozornění: Na serveru *Merlin* je třeba dodržet testování příkazem `php8.3`, protože pouhým `php` se spouští verze, která má omezen přístup k souborovému systému!

K řešení lze využít standardně předinstalované knihovny obou jazykových prostředí na serveru *Merlin*. Případné využití jiné knihovny je třeba konzultovat s garantem projektu (především z důvodu, aby se řešení projektu použitím vhodné knihovny nestalo zcela triviálním a případně byla knihovna dostupná v rámci *ipp-core*). Seznam povolených a zakázaných knihoven je udržován aktuální na *Moodle*. Ve skriptech v jazyce PHP jsou některé funkce z bezpečnostních důvodů zakázány (např. `header`, `mail`, `popen`, `curl_exec`, `socket_*`; úplný seznam je u povolených/zakázaných knihoven na *Moodle*).

Každý skript bude pracovat s jedním společným parametrem:

- `--help` vypíše na standardní výstup nápovědu skriptu (nenačítá žádný vstup), kterou lze převzít ze zadání (lze odstranit diakritiku, případně přeložit do angličtiny dle zvoleného jazyka dokumentace), a vrací návratovou hodnotu 0. Tento parametr nelze kombinovat s žádným dalším parametrem, jinak skript ukončete s chybou 10.

Kombinovatelné parametry skriptů jsou odděleny alespoň jedním bílým znakem a mohou být uváděny v libovolném pořadí, pokud nebude řečeno jinak. U skriptů je možné implementovat i vaše vlastní nekolizní parametry (doporučujeme konzultaci na *Fóru* nebo u garanta projektu).

Není-li řečeno jinak, tak dle konvencí unixových systémů lze uvažovat zástupné zkrácené (s jednou pomlčkou) i dlouhé parametry (se dvěma pomlčkami), které lze se zachováním sémantiky zaměňovat (tzv. alias parametry), ale testovány budou vždy dlouhé verze.

Je-li součástí parametru i soubor (např. `--source=file` nebo `--source="file"`) či cesta, může být tento soubor/cesta zadán/a relativní cestou¹⁰ nebo absolutní cestou; výskyt znaku uvozovek a rovnítko ve *file* neuvažujte. Cesty/jména souborů mohou obsahovat i Unicode znaky v UTF-8.

Doporučení: Jelikož je velká část hodnocení odvozena automatickými testy, doporučujeme i pro vývoj skriptů využívat vlastní automatické testování. K tomu je potřeba sada testů, které si sestavíte na základě správného pochopení zadání. Lze se inspirovat několika zveřejněnými testy. Kromě návratového kódu je vhodné testovat i správnost výstupu v případě úspěšného dokončení skriptu. Mezi studenty je možné testy i sdílet.

3 Analyzátor kódu v IPPcode24 (`parse.py`)

Skript typu filtr (`parse.py` v jazyce Python 3.10) načte ze standardního vstupu zdrojový kód v IPPcode24 (viz sekce 5), zkontroluje lexikální a syntaktickou správnost kódu a vypíše na standardní výstup XML reprezentaci programu dle specifikace v sekci 3.1.

Tento skript bude pracovat s těmito parametry:

- `--help` viz společný parametr všech skriptů v sekci 2.2.

Chybové návratové kódy specifické pro analyzátor:

- 21 - chybná nebo chybějící hlavička ve zdrojovém kódu zapsaném v IPPcode24;
- 22 - neznámý nebo chybný operační kód ve zdrojovém kódu zapsaném v IPPcode24;
- 23 - jiná lexikální nebo syntaktická chyba zdrojového kódu zapsaného v IPPcode24.

¹⁰Relativní cesta nebude obsahovat zástupný symbol `~` (vlňka).

3.1 Popis výstupního XML formátu

Za povinnou XML hlavičkou¹¹ následuje kořenový element `program` (s povinným textovým atributem `language` s hodnotou `IPPCODE24`), který obsahuje pro instrukce elementy `instruction`. Každý element `instruction` obsahuje povinný atribut `order` s pořadím instrukce. Při generování elementů je pořadí číslováno od 1 v souvislé posloupnosti. Dále element obsahuje povinný atribut `opcode` (hodnota operačního kódu je ve výstupním XML vždy velkými písmeny) a elementy pro odpovídající počet operandů/argumentů: `arg1` pro případný první argument instrukce, `arg2` pro případný druhý argument a `arg3` pro případný třetí argument instrukce. Element pro argument má povinný atribut `type` s možnými hodnotami `int`, `bool`, `string`, `nil`, `label`, `type`, `var` podle toho, zda se jedná o literál, návěští, typ nebo proměnnou, a obsahuje textový element.

Tento textový element potom nese buď hodnotu literálu (již bez určení typu a bez znaku `@`), jméno návěští, typ, nebo identifikátor proměnné (včetně určení paměťového rámce a `@`). U proměnných vypisujte označení paměťového rámce vždy velkými písmeny, jak by mělo být již na vstupu. Velikosti písmen samotného jména proměnné ponechejte beze změny. Formát celých čísel je dekadický, oktalový nebo hexadecimální dle zvyklostí Python 3, nicméně na výstup tato čísla vypisujte přesně ve formátu, v jakém byla načtena ze zdrojového kódu (např. zůstanou kladná znaménka čísel nebo počáteční přebytné nuly). U literálů typu `string` při zápisu do XML nepřevádějte původní escape sekvence, ale pouze pro problematické znaky v XML (např. `<`, `>`, `&`) využijte odpovídající XML entity (např. `<`, `>`, `&`). Podobně převádějte problematické znaky vyskytující se v identifikátorech proměnných. Literály typu `bool` vždy zapisujte malými písmeny jako `false` nebo `true`.

Doporučení: Všimněte si, že analýza `IPPCODE24` je tzv. kontextově závislá (viz přednášky), kdy například můžete mít klíčové slovo použito jako návěští a z kontextu je třeba rozpoznat, zda jde o návěští, nebo ne. Při tvorbě analyzátoru doporučujeme kombinovat konečně-stavové řízení a regulární výrazy a pro generování výstupního XML využít vhodnou knihovnu.

Výstupní XML bude porovnáváno s referenčními výsledky pomocí nástroje `A7Soft JExamXML`¹², viz [2]. Pozor, v Python příkaz `return` neslouží pro návrat chybového kódu, použijte funkci `sys.exit`.

Pokud se teprve učíte psát čitelný kód v Python 3, tak doporučujeme nastudovat [6].

3.2 Bonusová rozšíření

STATP Sbírá statistiky zpracovaného zdrojového kódu v `IPPCODE24`. Skript bude podporovat parametr `--stats=file` pro zadání souboru `file`, kam se budou vypisovat statistiky v parametrech umístěných za tímto `--stats`. Statistiky se do souboru vypisují po řádcích dle pořadí v parametrech s možností jejich opakování; na každý řádek nevypisujte nic kromě požadovaného číselného výstupu a odřádkování; případně existující soubor je přepsán. Pro sběr skupin statistik do různých souborů se použije další výskyt parametru `--stats` s jiným jménem souboru a následovaný dalšími parametry nové skupiny statistik. Parametr `--loc` vypíše do statistik počet řádků s instrukcemi (nepočítají se prázdné řádky ani řádky obsahující pouze komentář ani úvodní řádek). Parametr `--comments` vypíše do statistik počet řádků, na kterých se vyskytoval komentář. Parametr `--labels` vypíše do statistik počet definovaných návěští (tj. unikátních možných cílů skoku). Parametr `--jumps` vypíše do statistik počet všech instrukcí návratů z volání a instrukcí pro *skoky* (souhrnně podmíněné/nepodmíněné skoky a volání), `--fwjumps` počet dopředných skoků, `--backjumps` zpětných skoků a `--badjumps` počet skoků

¹¹Tradiční XML hlavička včetně verze a kódování je `<?xml version="1.0" encoding="UTF-8"?>`

¹²Nastavení `A7Soft JExamXML` pro porovnávání XML (soubor `options`) je na Moodle.

na neexistující návěští. Je-li uveden pouze parametr `--stats` bez upřesnění statistik k výpisu, bude výstupem prázdný soubor. Parametr `--frequent` vypíše do statistik jména operačních kódů (velkými písmeny, čárkou oddělené, bez mezer, abecedně vzestupně), které jsou ve zdrojovém kódu nejčastější dle počtu statických výskytů. Parametr `--print=string` vypíše do statistik řetězec *string* a parametr `--eol` vytiskne do statistik odřádkování. Chybí-li při zadání parametrů statistik `--loc`, `--comments` a dalších před nimi parametr `--stats`, jedná se o chybu 10. Pokus o zápis více skupin statistik do stejného souboru během jednoho spuštění skriptu vede na chybu 12. [1,5 b]

NVP Při návrhu a implementaci skriptu `parse.py` a pomocných skriptů bude aplikováno objektově orientované programování a využit alespoň jeden vhodný standardní návrhový vzor výjma Jedináčka (viz [5] a tipy *na Moodle*). Podmínkou hodnocení tohoto rozšíření je řádná dokumentace (proč, kde, jak, popis omezení). V případě uvedení rozšíření v dokumentaci a bez vážné snahy o implementaci může obdržet jednobodový malus (tj. -1 b). [1 b]

4 Interpret XML reprezentace kódu (`interpret.php`)

Seznamte se s dodaným rámcem *ipp-core* v PHP 8.3 a s jeho **povinným** využitím navrhnete a implementujete interpret, který načte XML reprezentaci programu a tento program s využitím vstupu dle parametrů příkazové řádky interpretuje a generuje výstup. S některými základními úkony pomůže dodaný rámec *ipp-core* (např. načtení standardního vstupu, výpisy, základní zpracování parametrů). Vstupní XML reprezentace je definována trochu volněji než XML generované skriptem `parse.py`, ale lze předpokládat, že vstupní XML reprezentace již nebude obsahovat chyby¹³, jež měl ze zadání za úkol detekovat `parse.py`. Jelikož je rámec *ipp-core* navržen objektově orientovaně, tak požadujeme, aby i jeho využívání a rozšiřování bylo provedeno objektově orientovaně se zvažováním využití vhodných návrhových vzorů (doporučených na Moodle). K objektovému návrhu a implementaci je samozřejmě povinné sepsat také stručnou a terminologicky správnou dokumentaci, jak jste rámec *ipp-core* pro splnění zadání použili/rozšířili.

Interpret navíc oproti sekci 3.1 podporuje existenci volitelných dokumentačních textových atributů `name` a `description` v kořenovém elementu `program` a různě naformátované značky (např. volitelné využití zkráceného zápisu značek, pokud neobsahuje značka podelement). Sémantika jednotlivých instrukcí IPPcode24 je popsána v sekci 5. Interpretace instrukcí probíhá dle atributu `order` vzestupně (nicméně, sekvence nemusí být souvislá/seřazená na rozdíl od sekce 3.1).

Tento skript bude pracovat s těmito parametry:

- `--help` viz společný parametr všech skriptů v sekci 2.2;
- `--source=file` vstupní soubor s XML reprezentací zdrojového kódu dle definice ze sekce 3.1 a doplnění v úvodu sekce 4;
- `--input=file` soubor se vstupy¹⁴ pro samotnou interpretaci zadaného zdrojového kódu.

Alespoň jeden z parametrů (`--source` nebo `--input`) musí být vždy zadán. Pokud jeden z nich chybí, jsou chybějící data načítána ze standardního vstupu.

¹³Nicméně generované výstupní XML není v 1. úloze jinak validováno (viz chybové návratové kódy).

¹⁴Vstup/vstupní soubor může být prázdný; např. neinterpretuje-li se žádná instrukce READ.

Chybové návratové kódy specifické pro interpret:

- 31 - chybný XML formát ve vstupním souboru (soubor není tzv. dobře formátovaný, angl. *well-formed*, viz [1]);
- 32 - neočekávaná struktura XML (např. element pro argument mimo element pro instrukci, instrukce s duplicitním pořadím nebo záporným pořadím);
- 88 - integrační chyba (nevalidní integrace s rámcem *ipp-core*).

Chybové návratové kódy interpretu v případě chyby během interpretace jsou uvedeny v popisu jazyka IPPcode24 (viz sekce 5.1).

Vaše zdrojové kódy druhé úlohy budeme kontrolovat i statickou analýzou pomocí nástroje PHPStan, kde pro získání nějakých bodů z celé úlohy požadujeme splnění úrovně 0 a pro získání celého 1 bodu (součást hodnocení dokumentace a kvality kódu úlohy) bez ztrát požadujeme splnění úrovně 6¹⁵, ale všeobecně doporučujeme úroveň 9, která je v nástroji implicitní.

Rámec *ipp-core*: Pro podpoření a ilustraci vhodnosti objektového návrhu je připraven jednoduchý rámec jménem *ipp-core*, který povinně využijete při řešení této úlohy. Rámec je napsán v PHP 8.3 a jeho aktuální verze je k dispozici na fakultním git serveru¹⁶ nebo v adresáři `/pub/courses/ipp/ipp-core` na serveru Merlin. V zadání zmíníme jen základní vlastnosti rámce, více informací najdete v README a studiem komentovaného zdrojového kódu. Nejasnosti ohledně rámce je vhodné diskutovat na Fóru. Rámec *ipp-core* dodržuje standard zápisu tříd PSR-4¹⁷ (automatické načítání tříd, tzv. *autoloader*). Instalace rámce na serveru *Merlin* do vlastního aktuálního adresáře může vypadat následovně:

```
git clone https://git.fit.vutbr.cz/IPP/ipp-core.git
cd ipp-core
php8.3 composer.phar install
```

čímž se do složky `vendor` nainstalují skripty a konfigurace pro *autoloader* dle PSR-4 a nástroj *PHPStan* sloužící pro statickou analýzu kódu a základní ohodnocení kvality vašeho kódu. Analýzu vašeho kódu na serveru *Merlin* podle úrovně 6 potom spustíte příkazem:

```
php8.3 vendor/bin/phpstan analyze --level=6
```

Veškeré úpravy provádějte a nové soubory vytvářejte pouze ve složce `student`, jejíž obsah budete jako jediný odevzdávat. Tj. i dokumentace a případný soubor `rozsireni` musí být v této složce! Složka `core` uchovává třídy samotného rámce ze jmenného prostoru `IPP\Core` a ve složce `vendor` budou případně další knihovny třetích stran, budou-li povolené pro použití v této úloze.

Celý program využívající rámec se spouští předchystaným skriptem `interpret.php`. Vaše implementace začíná v souboru `student\Interpreter.php` a odpovídající třídě `Interpreter` ve jmenném prostoru `IPP\Student`.

Rámec *ipp-core* využijte pro načtení vstupního kódu ve formátu XML (vrací se `DOMDocument`), základní zpracování parametrů (třída `Settings`) a pro interpretaci vstupně-výstupních instrukcí využijte rozhraní `InputReader` a `OutputWriter`. K dispozici bude i skript `is_it_ok.sh` pro kontrolu základních formálních požadavků a `Makefile` s několika základními cíli. Pro výpis varování na standardní chybový výstup obsahuje `interpret.php` na začátku příkaz `ini_set('display_errors', 'stderr');`;

¹⁵<https://phpstan.org/user-guide/rule-levels>

¹⁶První přihlášení doporučujeme provést přes webový prohlížeč na <https://git.fit.vutbr.cz/IPP>, aby došlo k automatické aktivaci loginu jako uživatelského jména místo e-mailové adresy pro SSH přístup.

¹⁷<https://www.php-fig.org/psr/psr-4/>

Dokumentace a objektově-orientovaný (OO) návrh: Kód bude povinně využívat rámec *ipp-core* a bude navržen objektově, což znamená, že je třeba se seznámit s terminologií OOP (min. v rozsahu přednášek), zamyslet se nad možnostmi využití OO a návrhových vzorů pro tuto úlohu, návrh stručně a jasně (správnou terminologií) popsat v dokumentaci v souladu s Vaší implementací (navrhované, ale neimplementované části je třeba označit). Vhodným způsobem promítněte OO návrh do kódu (např. volitelné otypování signatur metod a funkcí) včetně vhodného volení identifikátorů tříd, atributů a metod s ohledem na sebedokumentaci. Nestačí-li jednoslovné pojmenování pro jasné zachycení účelu, doplňte komentář, u metody popište i význam/účel parametrů, pokud jméno a typová anotace nestačí. Povinnou součástí dokumentace je UML diagram tříd s tím, že třídy rámce lze uvádět ve zkrácené formě (tj. bez atributů a metod), ale pro přehlednost důležité vztahy i mezi třídami a rozhraními z rámce nevynechávejte. Nicméně Vámi neimplementované části UML diagramu vhodně odlište (např. šedou barvou). Případné využití návrhových vzorů je třeba náležitě zdůvodnit a jasně zdokumentovat, jak a kterými třídami je implementován. Především návrhový vzor Jedináček je téměř vždy využit nevhodně a navíc špatně implementovaný.

Doporučení: V případě nekompletní implementace se zaměřte na funkčnost globálního paměťového rámce, práce s proměnnými typu `int`, instrukce `WRITE` a instrukce pro řízení toku programu.

Pro vývoj na lokálním stroji je možné využít, že rámec *ipp-core* je k dispozici jako vývojářský kontejner, který lze pohodlně používat například ve Visual Studio Code (více viz README v rámci *ipp-core*).

4.1 Bonusová rozšíření

FLOAT Podpora typu `float` v IPPcode24 (dekadický i vědecký zápis v analyzátoru i v interpretu včetně načítání ze standardního vstupu dle pravidel pro desetinná čísla¹⁸ v PHP 8). Podporujte instrukce pro práci s tímto typem: `INT2FLOAT`, `FLOAT2INT`, aritmetické instrukce (včetně `DIV`), atd. (viz [3]). Podpora v `parse.py` je možná, ale nebude testována. [1 b]

STACK Podpora zásobníkových variant instrukcí (přípona `S`; viz [3]): `CLEARs`, `ADDs/SUBs`, `MULs/IDIVs`, `LTS/GTS/EQS`, `ANDs/ORS/NOTs`, `INT2CHARs/STR2INTs` a `JUMPIFEQs/JUMPIFNEQs`. Zásobníkové verze instrukcí z datového zásobníku vybírají operandy se vstupními hodnotami dle popisu tříadresné instrukce od konce (tj. typicky nejprve $\langle symb_2 \rangle$ a poté $\langle symb_1 \rangle$). Podpora v `parse.py` je možná, ale nebude testována. [1 b]

STATI Sbírání statistik interpretace kódu. Skript bude podporovat parametr `--stats=file` pro zadání souboru *file*, kam se agregované statistiky budou vypisovat (po řádcích dle pořadí v dalších (případně opakovaných) parametrech; na každý řádek nevypisujte nic kromě požadovaného číselného či řetězcového výstupu). Podpora parametru `--insts` pro výpis počtu tzv. vykonaných instrukcí (tj. kromě ladicích instrukcí a speciální instrukce `LABEL`) během interpretace do statistik. Parametr `--hot` vypíše do statistik hodnotu atributu `order` u vykonané instrukce, která byla provedena nejvícekrát a má hodnotu atributu `order` nejmenší. Podpora parametru `--vars` pro výpis maximálního počtu inicializovaných proměnných přítomných v jeden okamžik ve všech platných paměťových rámcích během interpretace zadaného programu do statistik. Parametr `--stack` vypíše do statistik max. obsazenost (počet uložených hodnot) datového zásobníku během celé interpretace programu. Parametr `--print=string` vypíše do statistik řetězec *string* a parametr `--eol` vytiskne do statistik odřádkování. Chybí-li při zadání některého z akčních parametrů úvodní parametr `--stats`, jedná se o chybu 10. [1 b]

¹⁸<https://www.php.net/manual/en/language.types.float.php>

5 Popis jazyka IPPcode24

Nestrukturovaný imperativní jazyk IPPcode24 vznikl úpravou jazyka IFJcode23 (jazyk pro mezikód překladače jazyka IFJ23, viz [3]), který zahrnuje instrukce tříadresné (typicky se třemi argumenty) a případně zásobníkové (typicky méně parametrů a pracující s hodnotami na datovém zásobníku). Každá instrukce se skládá z operačního kódu (klíčové slovo s názvem instrukce), u kterého nezáleží na velikosti písmen (tj. case insensitive). Zbytek instrukcí tvoří operandy, u kterých na velikosti písmen záleží (tzv. case sensitive). Operandů oddělujeme libovolným nenulovým počtem mezer či tabulátorů. Také před operačním kódem a za posledním operandem se může vyskytnout libovolný počet mezer či tabulátorů. Odřádkování slouží pro oddělení jednotlivých instrukcí, takže na každém řádku je maximálně jedna instrukce a není povoleno jednu instrukci zapisovat na více řádků. Každý operand je tvořen proměnnou, konstantou, typem nebo návěštím. V IPPcode24 jsou podporovány jednořádkové komentáře začínající mřížkou (#). Vyjma prázdných¹⁹ či zakomentovaných řádků obsahuje kód v jazyce IPPcode24 na začátku místo instrukce jen identifikátor jazyka (tečka následovaná jménem jazyka, kdy nezáleží na velikosti písmen):

.IPPcode24

5.1 Návrátové hodnoty interpretu

Proběhne-li interpretace bez chyb, vrací se návratová hodnota daná instrukcí EXIT nebo implicitně 0 (nula). Chybovým případům odpovídají následující návratové hodnoty:

- 52 - chyba při sémantických kontrolách vstupního kódu v IPPcode24 (např. použití nedefinovaného návěští, redefinice proměnné);
- 53 - běhová chyba interpretace – špatné typy operandů;
- 54 - běhová chyba interpretace – přístup k neexistující proměnné (paměťový rámec existuje);
- 55 - běhová chyba interpretace – paměťový rámec neexistuje (např. čtení z prázdného zásobníku rámců);
- 56 - běhová chyba interpretace – chybějící hodnota (v proměnné, na datovém zásobníku nebo v zásobníku volání);
- 57 - běhová chyba interpretace – špatná hodnota operandu (např. dělení nulou, špatná návratová hodnota instrukce EXIT);
- 58 - běhová chyba interpretace – chybná práce s řetězcem.

5.2 Paměťový model

Hodnoty během interpretace nejčastěji ukládáme do pojmenovaných proměnných, které jsou sdružovány do tzv. *paměťových rámců*, což jsou v podstatě slovníky proměnných s jejich hodnotami. IPPcode24 nabízí tři druhy paměťových rámců:

- globální, značíme GF (Global Frame), který je na začátku interpretace automaticky inicializován jako prázdný; slouží pro ukládání globálních proměnných;
- lokální, značíme LF (Local Frame), který je na začátku nedefinován a odkazuje na vrcholový/aktuální paměťový rámec na zásobníku rámců; slouží pro ukládání lokálních proměnných funkcí (zásobník rámců lze s výhodou využít při zanoření či rekurzivním volání funkcí);

¹⁹Řádek obsahující pouze bílé znaky je považován za prázdný.

- dočasný, značíme TF (Temporary Frame), který slouží pro chystání nového nebo úklid starého paměťového rámce (např. při volání nebo dokončování funkce), jenž může být přesunut na zásobník rámců a stát se aktuálním lokálním paměťovým rámcem. Na začátku interpretace je dočasný paměťový rámec nedefinovaný.

K překrytým (dříve vloženým) lokálním paměťovým rámcům v zásobníku rámců nelze přistoupit dříve, než vyjmeme později přidané rámce.

Další možností pro ukládání nepojmenovaných hodnot je datový zásobník využívaný zásobníkovými instrukcemi.

5.3 Datové typy

IPPCODE24 pracuje s typy operandů dynamicky, takže je typ proměnné (resp. paměťového místa) dán obsaženou hodnotou. Není-li řečeno jinak, jsou implicitní konverze zakázány. Interpret podporuje speciální hodnotu/typ `nil` a tři základní datové typy (`int`, `bool` a `string`), jejichž rozsahy i přesnosti jsou kompatibilní s jazykem PHP 8.

Zápis každé konstanty v IPPCODE24 se skládá ze dvou částí oddělených zavináčem (znak `@`; bez bílých znaků), označení typu konstanty (`int`, `bool`, `string`, `nil`) a samotné konstanty (číslo, literál, `nil`). Např. `bool@true`, `nil@nil` nebo `int@-5`.

Typ `int` reprezentuje celé číslo (přetečení/podtečení neřešte). Typ `bool` reprezentuje pravdivostní hodnotu (`false` nebo `true`). Literál pro typ `string` je v případě konstanty zapsán jako sekvence tisknutelných znaků v kódování UTF-8 (vyjma bílých znaků, mřížky (`#`) a zpětného lomítka (`\`)) a escape sekvencí, takže není ohraničen uvozovkami. Escape sekvence, která je nezbytná pro znaky s dekadickým kódem 000–032, 035 a 092, je tvaru `\xyz`, kde `xyz` je dekadické číslo v rozmezí 000–999 složené právě ze tří číslic²⁰; např. konstanta

```
string@řetězec\032s\032lomítkem\032\092\032a\010novým\035řádkem
```

reprezentuje řetězec

```
řetězec s lomítkem \ a
novým#řádkem
```

Pokus o práci s neexistující proměnnou (čtení nebo zápis) vede na chybu 54. Pokus o čtení hodnoty neinicializované proměnné vede na chybu 56. Pokus o interpretaci instrukce s operandy nevhodných typů dle popisu dané instrukce vede na chybu 53.

5.4 Instrukční sada

U popisu instrukcí sázíme operační kód tučně a operandy zapisujeme pomocí neterminálních symbolů (případně číslovaných) v úhlových závorkách. Neterminál $\langle var \rangle$ značí proměnnou, $\langle symb \rangle$ konstantu nebo proměnnou, $\langle label \rangle$ značí návěští. Identifikátor proměnné se skládá ze dvou částí oddělených zavináčem (znak `@`; bez bílých znaků), označení paměťového rámce `LF`, `TF` nebo `GF` a samotného jména proměnné (sekvence libovolných alfanumerických a speciálních znaků bez bílých znaků začínající písmenem nebo speciálním znakem, kde speciální znaky jsou: `_`, `-`, `$`, `&`, `%`, `*`, `!`, `?`). Např. `GF@_x` značí proměnnou `_x` uloženou v globálním paměťovém rámcu.

Na zápis návěští se vztahují stejná pravidla jako na jméno proměnné (tj. část identifikátoru za zavináčem).

Příklad jednoduchého programu v IPPCODE24:

²⁰Zápis znaků s kódem Unicode větším jak 126 pomocí těchto escape sekvencí nebudeme testovat.

```

.IPPcode24
DEFVAR GF@counter
MOVE GF@counter string@ #Inicializace proměnné na prázdný řetězec
#Jednoduchá iterace, dokud nebude splněna zadaná podmínka
LABEL while
JUMPIFEQ end GF@counter string@aaa
WRITE string@Proměnná\032GF@counter\032obsahuje\032
WRITE GF@counter
WRITE string@\010
CONCAT GF@counter GF@counter string@a
JUMP while
LABEL end

```

Instrukční sada nabízí instrukce pro práci s proměnnými v paměťových rámcích, různé skoky, operace s datovým zásobníkem, aritmetické, logické a relační operace, dále také konverzní, vstupně/-výstupní a ladicí instrukce.

5.4.1 Práce s paměťovými rámci, volání funkcí

MOVE $\langle var \rangle$ $\langle symb \rangle$ Přiřazení hodnoty do proměnné
Zkopíruje hodnotu $\langle symb \rangle$ do $\langle var \rangle$. Např. `MOVE LF@par GF@var` provede zkopírování hodnoty proměnné `var` v globálním paměťovém rámci do proměnné `par` v lokálním rámci.

CREATEFRAME Vytvoř nový dočasný paměťový rámec
Vytvoří nový dočasný paměťový rámec a zahodí případný obsah původního dočasného rámce.

PUSHFRAME Přesun dočasného rámce na zásobník rámců
Přesun TF na zásobník rámců. Paměťový rámec bude k dispozici přes LF a překryje původní rámec na zásobníku rámců. TF bude po provedení instrukce nedefinován a je třeba jej před dalším použitím vytvořit pomocí `CREATEFRAME`. Pokus o přístup k nedefinovanému rámci vede na chybu 55.

POPFRAME Přesun aktuálního paměťového rámce do dočasného
Přesun vrcholový rámec LF ze zásobníku rámců do TF. Pokud žádný rámec v LF není k dispozici, dojde k chybě 55.

DEFVAR $\langle var \rangle$ Definuj novou proměnnou v paměťovém rámci
Definuje proměnnou v určeném rámci dle $\langle var \rangle$. Tato proměnná je zatím neinicializovaná a bez určení typu, který bude určen až přiřazením nějaké hodnoty. Opakovaná definice proměnné již existující v daném rámci vede na chybu 52.

CALL $\langle label \rangle$ Skok na návěští s podporou návratu
Uloží inkrementovanou aktuální pozici z interního čítače instrukcí do zásobníku volání a provede skok na zadané návěští (případnou přípravu paměťového rámce musí zajistit jiné instrukce).

RETURN Návrat na pozici uloženou instrukcí CALL
Vyjme pozici ze zásobníku volání a skočí na tuto pozici nastavením interního čítače instrukcí (úklid lokálních rámců musí zajistit jiné instrukce). Provedení instrukce při prázdném zásobníku volání vede na chybu 56.

5.4.2 Práce s datovým zásobníkem

Operační kód zásobníkových instrukcí je zakončen písmenem „S“. Zásobníkové instrukce případně načítají chybějící operandy z datového zásobníku a výslednou hodnotu operace případně ukládají zpět na datový zásobník.

PUSHS $\langle symb \rangle$ Vlož hodnotu na vrchol datového zásobníku
Uloží hodnotu $\langle symb \rangle$ na datový zásobník.

POPS $\langle var \rangle$

Vyjmi hodnotu z vrcholu datového zásobníku
Není-li zásobník prázdný, vyjme z něj hodnotu a uloží ji do proměnné $\langle var \rangle$, jinak dojde k chybě 56.

5.4.3 Aritmetické, relační, booleovské a konverzní instrukce

V této sekci jsou popsány tříadresné instrukce pro klasické operace pro výpočet výrazu. Přetečení nebo podtečení číselného výsledku neřešte.

ADD $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$

Součet dvou číselných hodnot

Sečte $\langle symb_1 \rangle$ a $\langle symb_2 \rangle$ (musí být typu int) a výslednou hodnotu téhož typu uloží do proměnné $\langle var \rangle$.

SUB $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$

Odečítání dvou číselných hodnot

Odečte $\langle symb_2 \rangle$ od $\langle symb_1 \rangle$ (musí být typu int) a výslednou hodnotu téhož typu uloží do proměnné $\langle var \rangle$.

MUL $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$

Násobení dvou číselných hodnot

Vynásobí $\langle symb_1 \rangle$ a $\langle symb_2 \rangle$ (musí být typu int) a výslednou hodnotu téhož typu uloží do proměnné $\langle var \rangle$.

IDIV $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$

Dělení dvou celočíselných hodnot

Celočíselně podělí celočíselnou hodnotu ze $\langle symb_1 \rangle$ druhou celočíselnou hodnotou ze $\langle symb_2 \rangle$ (musí být oba typu int) a výsledek typu int přiřadí do proměnné $\langle var \rangle$. Dělení nulou způsobí chybu 57.

LT/GT/EQ $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$

Relační operátory menší, větší, rovno

Instrukce vyhodnotí relační operátor mezi $\langle symb_1 \rangle$ a $\langle symb_2 \rangle$ (stejného typu; int, bool nebo string) a do $\langle var \rangle$ zapíše výsledek typu bool (**false** při neplatnosti nebo **true** v případě platnosti odpovídající relace). Řetězce jsou porovnávány lexikograficky a **false** je menší než **true**. Pro výpočet neostrých nerovností lze použít AND/OR/NOT. S operandem typu nil (další zdrojový operand je libovolného typu) lze porovnávat pouze instrukcí EQ, jinak chyba 53.

AND/OR/NOT $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$

Základní booleovské operátory

Aplikuje konjunkci (logické A)/disjunkci (logické NEBO) na operandy typu bool $\langle symb_1 \rangle$ a $\langle symb_2 \rangle$ nebo negaci na $\langle symb_1 \rangle$ (NOT má pouze 2 operandy) a výsledek typu bool zapíše do $\langle var \rangle$.

INT2CHAR $\langle var \rangle \langle symb \rangle$

Převod celého čísla na znak

Číselná hodnota $\langle symb \rangle$ je dle Unicode převedena na znak, který tvoří jednoznakový řetězec přiřazený do $\langle var \rangle$. Není-li $\langle symb \rangle$ validní ordinální hodnota znaku v Unicode (viz funkce **chr** v Python 3), dojde k chybě 58.

STR2INT $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$

Ordinální hodnota znaku

Do $\langle var \rangle$ uloží ordinální hodnotu znaku (dle Unicode) v řetězci $\langle symb_1 \rangle$ na pozici $\langle symb_2 \rangle$ (indexováno od nuly). Indexace mimo daný řetězec vede na chybu 58. Viz funkce **ord** v Python 3.

5.4.4 Vstupně-výstupní instrukce

READ $\langle var \rangle \langle type \rangle$

Načtení hodnoty ze standardního vstupu

Načte jednu hodnotu dle zadaného typu $\langle type \rangle \in \{\text{int, string, bool}\}$ a uloží tuto hodnotu do proměnné $\langle var \rangle$. Načtení proveďte pomocí předchystané třídy **Reader** rámce *ipp-core*. V případě chybného nebo chybějícího vstupu bude do proměnné $\langle var \rangle$ uložena hodnota **nil@nil**.

WRITE $\langle symb \rangle$ Vypis hodnoty na standardní výstup
Vypíše hodnotu $\langle symb \rangle$ na standardní výstup. Až na typ `bool` a hodnotu `nil@nil` je formát výpisu kompatibilní s příkazem `print` jazyka Python 3 s doplňujícím parametrem `end=''` (zamezí dodatečnému odřádkování). Pravdivostní hodnota se vypíše jako `true` a nepravda jako `false`. Hodnota `nil@nil` se vypíše jako prázdný řetězec.

5.4.5 Práce s řetězci

CONCAT $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$ Konkatenace dvou řetězců
Do proměnné $\langle var \rangle$ uloží řetězec vzniklý konkatenací dvou řetězcových operandů $\langle symb_1 \rangle$ a $\langle symb_2 \rangle$ (jiné typy nejsou povoleny).

STRLEN $\langle var \rangle \langle symb \rangle$ Zjistí délku řetězce
Zjistí počet znaků (délku) řetězce v $\langle symb \rangle$ a tato délka je uložena jako celé číslo do $\langle var \rangle$.

GETCHAR $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$ Vrať znak řetězce
Do $\langle var \rangle$ uloží řetězec z jednoho znaku v řetězci $\langle symb_1 \rangle$ na pozici $\langle symb_2 \rangle$ (indexováno celým číslem od nuly). Indexace mimo daný řetězec vede na chybu 58.

SETCHAR $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$ Změň znak řetězce
Zmodifikuje znak řetězce uloženého v proměnné $\langle var \rangle$ na pozici $\langle symb_1 \rangle$ (indexováno celočíselně od nuly) na znak v řetězci $\langle symb_2 \rangle$ (první znak, pokud obsahuje $\langle symb_2 \rangle$ více znaků). Výsledný řetězec je opět uložen do $\langle var \rangle$. Při indexaci mimo řetězec $\langle var \rangle$ nebo v případě prázdného řetězce v $\langle symb_2 \rangle$ dojde k chybě 58.

5.4.6 Práce s typy

TYPE $\langle var \rangle \langle symb \rangle$ Zjistí typ daného symbolu
Dynamicky zjistí typ symbolu $\langle symb \rangle$ a do $\langle var \rangle$ zapíše řetězec značící tento typ (`int`, `bool`, `string` nebo `nil`). Je-li $\langle symb \rangle$ neinicilizovaná proměnná, označí její typ prázdným řetězcem.

5.4.7 Instrukce pro řízení toku programu

Neterminál $\langle label \rangle$ označuje návěští, které slouží pro označení pozice v kódu IPPcode24. V případě skoku na neexistující návěští dojde k chybě 52.

LABEL $\langle label \rangle$ Definice návěští
Speciální instrukce označující pomocí návěští $\langle label \rangle$ důležitou pozici v kódu jako potenciální cíl libovolné skokové instrukce. Pokus o vytvoření dvou stejně pojmenovaných návěští na různých místech programu je chybou 52.

JUMP $\langle label \rangle$ Nepodmíněný skok na návěští
Provede nepodmíněný skok na zadané návěští $\langle label \rangle$.

JUMPIFEQ $\langle label \rangle \langle symb_1 \rangle \langle symb_2 \rangle$ Podmíněný skok na návěští při rovnosti
Pokud jsou $\langle symb_1 \rangle$ a $\langle symb_2 \rangle$ stejného typu nebo je některý operand `nil` (jinak chyba 53) a zároveň se jejich hodnoty rovnají, tak provede skok na návěští $\langle label \rangle$.

JUMPIFNEQ $\langle label \rangle \langle symb_1 \rangle \langle symb_2 \rangle$ Podmíněný skok na návěští při nerovnosti
Jsou-li $\langle symb_1 \rangle$ a $\langle symb_2 \rangle$ stejného typu nebo je některý operand `nil` (jinak chyba 53), tak v případě různých hodnot provede skok na návěští $\langle label \rangle$.

EXIT $\langle symb \rangle$ Ukončení interpretace s návratovým kódem
Ukončí vykonávání programu, případně vypíše statistiky a ukončí interpret s návratovým kódem $\langle symb \rangle$, kde $\langle symb \rangle$ je celé číslo v intervalu 0 až 9 (včetně). Nevalidní celočíselná hodnota $\langle symb \rangle$ vede na chybu 57.

5.4.8 Ladicí instrukce

Následující ladicí instrukce (DPRINT a BREAK) nesmí ovlivňovat standardní výstup. Jejich skutečnou funkcionalitu nebudeme testovat, ale mohou se v testech objevit.

DPRINT $\langle symb \rangle$

Výpis hodnoty na `stderr`

Předpokládá se, že vypíše zadanou hodnotu $\langle symb \rangle$ na standardní chybový výstup (`stderr`).

BREAK

Výpis stavu interpretu na `stderr`

Předpokládá se, že na standardní chybový výstup (`stderr`) vypíše stav interpretu (např. pozice v kódu, obsah paměťových rámců, počet vykonaných instrukcí) v danou chvíli (tj. během vykonávání této instrukce).

Reference

- [1] Extensible Markup Language (XML) 1.0. W3C. World Wide Web Consortium [online]. 5. vydání. 26. 11. 2008 [cit. 2020-02-03]. Dostupné z: <https://www.w3.org/TR/xml/>
- [2] A7Soft JExamXML is a java based command line XML diff tool for comparing and merging XML documents. c2018 [cit. 2020-02-03]. Dostupné z: <https://www.a7soft.com/jexamxml.html>
- [3] Křivka, Z., a kol.: Zadání projektu z předmětu IFJ a IAL. c2023 [cit. 2024-01-22]. Dostupné z: <https://www.fit.vutbr.cz/study/courses/IFJ/private/projekt/ifj2023.pdf>
- [4] The text/markdown Media Type. Internet Engineering Task Force (IETF). 2016 [cit. 2020-02-03]. Dostupné z: <https://tools.ietf.org/html/rfc7763>
- [5] Gamma, E., a kol.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.
- [6] van Rossum, G., a kol.: PEP 8 – Style Guide for Python Code. c2023 [cit. 2024-01-30]. Dostupné z: <https://peps.python.org/pep-0008/>

Revize zadání:

2024-02-14: Opravy překlepů. Specifikace řazení operačních kódů pro statistiku `--frequent` v rozšíření STATP. Finalizace zadání.