Zadání projektu z předmětu IPP 2019/2020

Zbyněk Křivka
E-mail: krivka@fit.vutbr.cz

1 Základní charakteristika projektu

Navrhněte, implementujte, dokumentujte a testujte sadu skriptů pro interpretaci nestrukturovaného imperativního jazyka IPPcode20. K implementaci vytvořte odpovídající stručnou programovou dokumentaci. Projekt se skládá ze dvou úloh a je individuální.

První úloha se skládá ze skriptu `parse.php` v jazyce PHP 7.4 (viz sekce 3) a dokumentace k tomuto skriptu (viz sekce 2.1). Druhá úloha se skládá ze skriptu `interpret.py` v jazyce Python 3.8 (viz sekce 4), testovacího skriptu `test.php` v jazyce PHP 7.4 (viz sekce 5) a dokumentace těchto dvou skriptů (viz sekce 2.1).

2 Požadavky a organizační informace

Kromě implementace skriptů 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 *Souborech* předmětu IPP.

Termíny:

připomínky[1] k zadání projektu do 10. února 2020;

fixace zadání projektu od 11. února 2020;

odevzdání první úlohy ve středu **11. března** 2020 do 23:59:59;

odevzdání druhé úlohy v úterý **14. dubna** 2020 do 23:59:59.

Dodatečné informace a konzultace k projektu z IPP:

- *Wiki* předmětu IPP v IS FIT včetně Často kladených otázek (*FAQ*).
- *Fórum* předmětu IPP pro ak. rok 2019/2020, témata `IPP: Projekt.*`.
- U cvičících: Ivana Burgetová (burgetova@fit.vutbr.cz), Šárka Květoňová (kvetona@fit.vutbr.cz), Dominika Regéciová (iregeciova@fit.vutbr.cz), Dominika Klobučníková (iklobucnikova@fit.vutbr.cz), Martin Tomko (itomko@fit.vutbr.cz), Adam Kövari (ikovari@fit.vutbr.cz), Tomáš Dvořák (idvorakt@fit.vutbr.cz).
- 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`.
- Dušan Kolář (garant předmětu; jen v závažných případech): po dohodě e-mailem (uvádějte předmět začínající `"IPP:"`), viz `http://www.fit.vut.cz/person/kolar`.

[1] Naleznete-li v zadání nějakou chybu či nejasnost, dejte mi prosím vědět na *Fóru* předmětu nebo emailem na krivka@fit.vutbr.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* předmětu (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 IS FIT v předmětu IPP (odevzdání emailem není možné nebo je bodově postihováno) a odevzdáním stvrzujete výhradní autorství skriptů i dokumentace.

Do termínu „Projekt - 1. úloha v PHP 7.4“ odevzdáte archív pro první úlohu v jazyce PHP 7.4 (skript `parse.php`; včetně dokumentace tohoto skriptu). Po termínu odevzdání 1. úlohy bude otevřen termín „Projekt - 2. úloha v Pythonu 3.8 a testovací skript v PHP 7.4“ pro odevzdání archívu pro druhou úlohu (skripty `interpret.py` a `test.php`; včetně dokumentace pro obsažené skripty). Součástí archívu první úlohy mohou být i nedokončené skripty a dokumentace odevzdávané k hodnocení až v druhé úloze a naopak v archívu druhé úlohy se smí vyskytovat skript z první úlohy a adresář s vašimi testy. V případě odevzdání již plně funkčního skriptu `test.php` v rámci archívu 1. úlohy lze získat 1 bonusový bod.

Každá úloha bude odevzdána ve zvláštním archívu, kde budou soubory k dané úloze zkomprimovány programem ZIP, TAR+GZIP či TAR+BZIP do jediného archívu pojmenovaného `xlogin99.zip`, `xlogin99.tgz`, nebo `xlogin99.tbz`, kde `xlogin99` je Váš login. Velikost každého archívu bude omezena informačním systémem (pravděpodobně na 1 MB). Archív nesmí obsahovat speciální či binární spustitelné soubory. Názvy všech souborů mohou obsahovat pouze písmena anglické abecedy, číslice, tečku, pomlčku a podtržítko. **Skripty budou umístěny v kořenovém adresáři odevzdaného archívu**. Po rozbalení archívu na serveru Merlin bude možné skript(y) spustit. Archív smí obsahovat v rozumné míře pomocné adresáře (typicky pro vaše vlastní testy, vaše knihovny a pomocné skripty nebo povolené knihovny, 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.php` až 6 bodů; `interpret.py` až 8 bodů; `test.php` až 3 body. Tj. v součtu až **17 bodů**. Dokumentace a úprava zdrojových textů skriptů bude dohromady hodnocena až **3 body** (až 1 bod za popis skriptu `parse.php` a až 2 body za popis skriptů `interpret.py` a `test.php`), avšak maximálně 30 % ze sumy hodnocení skriptů 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 IS FIT zaokrouhlují na desetiny bodu.

Skripty budou spouštěny na serveru Merlin příkazem: *interpret skript parametry*, kde *interpret* bude `php7.4` nebo `python3.8`, *skript* a *parametry* závisí na dané úloze a skriptu. Hodnocení většiny funkčnosti bude zajišťovat automatizovaný nástroj. Kvalitu dokumentace, komentářů a strukturu zdrojového kódu budou hodnotit cvičící.

Vaše skripty budou hodnoceny nezásvisle na sobě, takže je možné odevzdat například jen skript `interpret.py` nebo jen `test.php` do 2. úlohy bez odevzdání 1. úlohy.

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

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 rozšíření[2] (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* předmětu IPP). 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 termínů „Projekt - Pokusné odevzdání 1. úlohy“ do **4. března 2020** a „Projekt - Pokusné odevzdání 2. úlohy“ do **6. dubna 2020**. Není nutné zahrnout dokumentaci, která spolu s rozšířeními pokusně vyhodnocena nebude. U skriptu `test.php` bude pokusné hodnocení omezeno jen na velmi základní automatizovatelné testy (bez manuální kontroly skutečného obsahu).

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 Markdown[3] (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). Dokumentace může být doplněna např. o UML diagram tříd, navržený konečný automat, pravidla Vámi vytvořené gramatiky nebo popis jiných formalismů a algoritmů. Nicméně **nesmí obsahovat ani částečnou kopii zadání**.

Rozsah textu dokumentace pro každý skript bude přibližně 1 normostrana. Vysázená dokumentace 1. úlohy popisující skript `parse.php` by neměla přesáhnout 1 stranu A4 a u 2. úlohy (popisující skripty `interpret.py` a `test.php`) pak 2 strany A4. Doporučení pro sazbu: 10-bodové 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 ro-

[2] Identifikátory rozšíření jsou uvedeny u konkrétního rozšíření tučně.

[3] Bude-li přítomna dokumentace v Markdown i PDF, tak bude hodnocena verze v PDF, kde je jistější sazba.

zumné míře je vhodné používat nadpisy první a druhé úrovně (12-bodové a 11-bodové písmo Times New Roman či ## a ### v Markdown) pro vytvoření logické struktury dokumentace.

Nadpis a hlavička dokumentace[4] bude na prvních třech řádcích obsahovat:

```
Implementační dokumentace k %cislo%. úloze do IPP 2019/2020
Jméno a příjmení: %name_surname%
Login: %xlogin99%
```

kde `%name_surname%` 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 hodnoceno i komentování zdrojového kódů (minimálně každá funkce a modul (třída) by měly mít svůj komentář o jejich účelu a parametrech; u složitějších funkcí okomentujte i omezení na vstupy či výstupy).

2.2 Programová část

Zadání projektu vyžaduje implementaci tří skriptů[5], které mají parametry příkazové řádky a je definováno, jakým způsobem manipulují se vstupy a výstupy. Skript (vyjma `test.php`) 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 došlo 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);
- 20 – 69 - návratové kódy chyb specifických pro jednotlivé skripty;
- 99 - interní chyba (neovlivněná vstupními soubory či parametry příkazové řádky; např. chyba alokace paměti).

Pokud zadání nestanoví níže jinak, tak veškeré vstupy a výstupy jsou v kódování UTF-8. Pro účely projektu z IPP musí být na serveru Merlin ponecháno implicitní nastavení `locale`[6], 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 (`.php` pro PHP 7 a `.py` pro Python 3). Vyhodnocení skriptů bude prováděno na serveru `Merlin` s aktuálními verzemi interpretů (dne 1. 2. 2020 bylo na tomto serveru nainstalováno `php7.4`[7] verze 7.4.2 a `python3.8`[8] verze 3.8.1).

[4] Anglické znění nadpisu a hlavičky dokumentace najdete v *FAQ* na *Wiki* předmětu.

[5] Tyto skripty jsou aplikace příkazové řádky neboli konzolové aplikace.

[6] Správné 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`.

[7] Upozornění: Na serveru Merlin je třeba dodržet testování příkazem `php7.4`, protože pouhým `php` se spouští verze, která nemá přístup k souborovému systému!

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

K řešení lze využít standardně předinstalované knihovny obou jazykových prostředí na serveru `Merlin`. V případě využití jiné knihovny kromě knihovny podporující načítání/ukládání formátu XML, zpracování parametrů příkazové řádky a zpracování regulárních výrazů je třeba konzultovat s garantem projektu (především z důvodu, aby se řešení projektu použitím vhodné knihovny nestalo zcela triviálním). Seznam povolených a zakázaných knihoven bude udržován aktuální na *Wiki* předmětu. 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 *Wiki*).

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). 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 cvičícího).

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, tak tento soubor/cesta může být zadán/a relativní cestou[9] nebo absolutní cestou; výskyt znaku uvozovek a rovnítka ve *file* neuvažujte. Cesta/jméno souboru mohou obsahovat i Unicode znaky v UTF-8.

3 Analyzátor kódu v IPPcode20 (`parse.php`)

Skript typu filtr (`parse.php` v jazyce PHP 7.4) načte ze standardního vstupu zdrojový kód v IPPcode20 (viz sekce 6), 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 IPPcode20;
- 22 - neznámý nebo chybný operační kód ve zdrojovém kódu zapsaném v IPPcode20;
- 23 - jiná lexikální nebo syntaktická chyba zdrojového kódu zapsaného v IPPcode20.

3.1 Popis výstupního XML formátu

Za povinnou XML hlavičkou[10] následuje kořenový element `program` (s povinným textovým atributem `language` s hodnotou IPPcode20), který obsahuje pro instrukce elementy `instruction`. Každý element `instruction` obsahuje povinný atribut `order` s pořadím instrukce (počítáno od 1, souvislá

[9] Relativní cesta nebude obsahovat zástupný symbol ~ (vlnka).

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

posloupnost) a 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 @), nebo jméno návěští, nebo typ, nebo identifikátor proměnné (včetně určení rámce a @). U proměnných ponechávejte označení rámce vždy velkými písmeny (samotné jméno proměnné ponechejte beze změny). V případě číselných literálů je zápis ponechán ve formátu ze zdrojového kódu (např. zůstanou kladná znaménka čísel nebo počáteční přebytečné nuly) a není třeba kontrolovat jejich lexikální správnost (na rozdíl od řetězcových literálů). 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 IPPcode20 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[11], viz [2].

3.2 Bonusová rozšíření

STATP Sbírání statistik zpracovaného zdrojového kódu v IPPcode20. Skript bude podporovat parametr `--stats=`*file* pro zadání souboru *file*, kam se agregované statistiky budou vypisovat (po řádcích dle pořadí v dalších parametrech s možností jejich opakování; na každý řádek nevypisujte nic kromě požadovaného číselného výstupu). 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 instrukcí pro podmíněné/nepodmíněné skoky, volání a návraty z volání dohromady. Je-li uveden pouze parametr `--stats` bez upřesnění statistik k výpisu, bude výstupem prázdný soubor. Chybí-li při zadání `--loc`, `--comments`, `--labels` nebo `--jumps`, parametr `--stats`, jedná se o chybu 10 [1 b].

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

Program načte XML reprezentaci programu a tento program s využitím vstupu dle parametrů příkazové řádky interpretuje a generuje výstup. Vstupní XML reprezentace je např. generována skriptem `parse.php` (ale ne nutně) ze zdrojového kódu v IPPcode20. Interpret navíc oproti sekci 3.1 podporuje existenci volitelných dokumentačních textových atributů `name` a `description` v kořenovém elementu `program`. Sémantika jednotlivých instrukcí IPPcode20 je popsána v sekci 6. Interpretace instrukcí probíhá dle atributu `order` vzestupně (sekvence nemusí být souvislá na rozdíl od sekce 3.1).

[11] Nastavení A7Soft JExamXML pro porovnávání XML (soubor `options`) je v *Souborech* k projektu v IS FIT.

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;
- `--input=`*`file`* soubor se vstupy pro samotnou interpretaci zadaného zdrojového kódu.

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

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) či lexikální nebo syntaktická chyba textových elementů a atributů ve vstupním XML souboru (např. chybný lexém pro řetězcový literál, neznámý operační kód apod.).

Chybové návratové kódy interpretu v případě chyby během interpretace jsou uvedeny v popisu jazyka IPPcode20 (viz sekce 6.1).

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

4.1 Bonusová rozšíření

FLOAT Podpora typu `float` v IPPcode20 (**hexadecimální zápis** v analyzátoru i v interpretu včetně načítání ze standardního vstupu; např. `float@0x1.2000000000000p+0` representuje 1,125; viz funkce `float.fromhex()` a `float.hex()` v Python 3). Podporujte instrukce pro práci s tímto typem: INT2FLOAT, FLOAT2INT, aritmetické instrukce (včetně DIV), atd. (viz [3]). Podpora v `parse.php` 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/STRI2INTS 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.php` 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 výstupu). Podpora parametru `--insts` pro výpis počtu vykonaných instrukcí během interpretace do statistik. Podpora parametru `--vars` pro výpis maximálního počtu inicializovaných proměnných přítomných ve všech platných rámcích během interpretace zadaného programu do statistik. Chybí-li při zadání `--insts` či `--vars` parametr `--stats`, jedná se o chybu 10. [1 b]

5 Testovací rámec (`test.php`)

Skript (`test.php` v jazyce PHP 7.4) bude sloužit pro automatické testování postupné aplikace `parse.php` a `interpret.py`[12]. Skript projde zadaný adresář s testy a využije je pro automatické otestování správné funkčnosti obou předchozích programů včetně vygenerování přehledného souhrnu v HTML 5 do standardního výstupu. Pro hodnocení `test.php` budou dodány referenční implementace `parse.php` i `interpret.py`. Testovací skript nemusí u předchozích dvou skriptů testovat jejich dodatečnou funkčnost aktivovanou parametry příkazové řádky (s výjimkou potřeby parametru `--source` a/nebo `--input` u `interpret.py`).

Tento skript bude pracovat s těmito parametry:

- `--help` viz společný parametr všech skriptů v sekci 2.2;
- `--directory=`*`path`* testy bude hledat v zadaném adresáři (chybí-li tento parametr, tak skript prochází aktuální adresář; v případě zadání neexistujícího adresáře dojde k chybě 11);
- `--recursive` testy bude hledat nejen v zadaném adresáři, ale i rekurzivně ve všech jeho podadresářích;
- `--parse-script=`*`file`* soubor se skriptem v PHP 7.4 pro analýzu zdrojového kódu v IPPcode20 (chybí-li tento parametr, tak implicitní hodnotou je `parse.php` uložený v aktuálním adresáři);
- `--int-script=`*`file`* soubor se skriptem v Python 3.8 pro interpret XML reprezentace kódu v IPPcode20 (chybí-li tento parametr, tak implicitní hodnotou je `interpret.py` uložený v aktuálním adresáři);
- `--parse-only` bude testován pouze skript pro analýzu zdrojového kódu v IPPcode20 (tento parametr se nesmí kombinovat s parametry `--int-only` a `--int-script`), výstup s referenčním výstupem (soubor s příponou `out`) porovnávejte nástrojem A7Soft JExamXML (viz [2]);
- `--int-only` bude testován pouze skript pro interpret XML reprezentace kódu v IPPcode20 (tento parametr se nesmí kombinovat s parametry `--parse-only` a `--parse-script`). Vstupní program reprezentován pomocí XML bude v souboru s příponou `src`.
- `--jexamxml=`*`file`* soubor s JAR balíčkem s nástrojem A7Soft JExamXML. Je-li parametr vynechán uvažuje se implicitní umístění `/pub/courses/ipp/jexamxml/jexamxml.jar` na serveru Merlin, kde bude `test.php` hodnocen.

Každý test je tvořen až 4 soubory stejného jména s příponami `src`, `in`, `out` a `rc` (ve stejném adresáři). Soubor s příponou `src` obsahuje zdrojový kód v jazyce IPPcode20 (příp. jeho XML reprezentaci). Soubory s příponami `in`, `out` a `rc` obsahují vstup a očekávaný/referenční výstup a očekávaný první chybový návratový kód analýzy resp. interpretace nebo bezchybový návratový kód 0. Pokud soubor s příponou `in` nebo `out` chybí, tak se automaticky dogeneruje prázdný soubor. V případě chybějícího souboru s příponou `rc` se vygeneruje soubor obsahující návratovou hodnotu 0. Při nenulovém návratovém kódu je test považován za úspěšný, pokud je získán správný návratový kód v odpovídajícím skriptu (výstupy se neporovnávají). V případě nulového kódu je třeba provést

[12] Za tímto účelem lze vytvářet dočasné soubory, které však nesmí přepsat žádný jiný existující soubor a potom musí být uklizeny.

i porovnání výstupu skriptu s referenčním výstupem. Při parametru `--parse-only` jsou výstupy porovnávány pomocí nástroje A7Soft JExamXML, jinak je použit unixový nástroj `diff`.

Testy budou umístěny v adresáři včetně případných podadresářů pro lepší kategorizaci testů. Adresářová struktura může mít libovolné zanoření. Není třeba uvažovat symbolické odkazy apod.

Požadavky na výstupní HTML verze 5: Přehledová stránka o úspěšnosti/neúspěšnosti jednotlivých testů a celých adresářů bude prohlédnuta ručně opravujícím, takže bude hodnocena její přehlednost a intuitivnost. Mělo by být na první pohled zřejmé, které testy uspěly a které nikoli (a kolik), a zda případně uspěly všechny testy (případně i po jednotlivých adresářích). Výsledná stránka nesmí načítat externí zdroje[13] a musí být možné ji zobrazit v běžném prohlížeči.

5.1 Bonusová rozšíření

FILES Podporujte parametr `--testlist=`*`file`* slouží pro explicitní zadání seznamu adresářů (zadaných relativními či absolutními cestami) a případně i souborů s testy (zadává se soubor s příponou `.src`) formou externího souboru *file* místo načtení testů z aktuálního adresáře (nelze kombinovat s parametrem `--directory`). Dále podporujte parametr `--match=`*`regexp`* pro výběr testů, jejichž jméno bez přípony (ne cesta) odpovídá zadanému regulárnímu výrazu *regexp* dle PCRE syntaxe (tj. včetně oddělovačů; syntaktická chyba výrazu vede na chybu 11). [1 b]

V případě odevzdání již plně funkčního skriptu `test.php` v rámci archívu 1. úlohy lze získat 1 bonusový bod.

6 Popis jazyka IPPcode20

Nestrukturovaný imperativní jazyk IPPcode20 vznikl úpravou jazyka IFJcode19 (jazyk pro mezikód překladače jazyka IFJ19, 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). Operandy 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 IPPcode20 jsou podporovány jednořádkové komentáře začínající mřížkou (#). Kód v jazyce IPPcode20 začíná úvodním řádkem[14] s tečkou následovanou jménem jazyka (nezáleží na velikosti písmen):

```
.IPPcode20
```

6.1 Návratové hodnoty interpretu

Proběhne-li interpretace bez chyb, vrací se návratová hodnota `0` (nula). Chybovým případům odpovídají následující návratové hodnoty:

[13] Přímo do generované HTML stránky je možné vložit vlastní JavaScript nebo CSS kód.

[14] Před úvodním řádkem se mohou nacházet komentáře.

- 52 - chyba při sémantických kontrolách vstupního kódu v IPPcode20 (např. použití nedefinovaného návěští, redefinice proměnné);
- 53 - běhová chyba interpretace – špatné typy operandů;
- 54 - běhová chyba interpretace – přístup k neexistující proměnné (rámec existuje);
- 55 - běhová chyba interpretace – 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.

6.2 Paměťový model

Hodnoty během interpretace nejčastěji ukládáme do pojmenovaných proměnných, které jsou sdružovány do tzv. rámců, což jsou v podstatě slovníky proměnných s jejich hodnotami. IPPcode20 nabízí tři druhy 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 rámec na zásobníku rámců; slouží pro ukládání lokálních proměnných funkcí (zásobník rámců lze s výhodou využít při zanořeném či rekurzivním volání funkcí);
- dočasný, značíme `TF` (Temporary Frame), který slouží pro chystání nového nebo úklid starého rámce (např. při volání nebo dokončování funkce), jenž může být přesunut na zásobník rámců a stát se aktuálním lokálním rámcem. Na začátku interpretace je dočasný rámec nedefinovaný.

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

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

6.3 Datové typy

IPPcode20 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 Python 3.

Zápis každé konstanty v IPPcode20 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 $\backslash xyz$, kde xyz je dekadické číslo v rozmezí 000-999 složené právě ze tří číslic[15]; např. konstanta

[15]Zápis znaků s kódem Unicode větším jak 126 pomocí těchto escape sekvencí nebudeme testovat.

`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.

6.4 Instrukční sada

U popisu instrukcí sázíme operační kód tučně a operandy zapisujeme pomocí neterminálních symbolů (případně číslovaných) v úhlových závorkách. Neterminál ⟨*var*⟩ značí proměnnou, ⟨*symb*⟩ konstantu nebo proměnnou, ⟨*label*⟩ značí návěští. Identifikátor proměnné se skládá ze dvou částí oddělených zavináčem (znak `@`; bez bílých znaků), označení 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 rámci.

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

Příklad jednoduchého programu v IPPcode20:

```
.IPPcode20
DEFVAR GF@counter
MOVE GF@counter string@  #Inicializace promenné na prázdný retezec
#Jednoduchá iterace, dokud nebude splnena zadaná podmínka
LABEL while
JUMPIFEQ end GF@counter string@aaa
WRITE string@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 rámcích, různé skoky, operace s datovým zásobníkem, aritmetické, logické a relační operace, dále také konverzní, vstupně/výstupní a ladicí instrukce.

6.4.1 Práce s rámci, volání funkcí

MOVE ⟨*var*⟩ ⟨*symb*⟩	Přiřazení hodnoty do proměnné
Zkopíruje hodnotu ⟨*symb*⟩ do ⟨*var*⟩. Např. `MOVE LF@par GF@var` provede zkopírování hodnoty proměnné `var` v globálním rámci do proměnné `par` v lokálním rámci.	
CREATEFRAME	Vytvoř nový dočasný rámec
Vytvoří nový dočasný rámec a zahodí případný obsah původního dočasného rámce.	

PUSHFRAME Přesun dočasného rámce na zásobník rámců

Přesuň `TF` na zásobník rámců. Rámec bude k dispozici přes `LF` a překryje původní rámce 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 rámce do dočasného

Přesuň vrcholový rámec `LF` ze zásobníku rámců do `TF`. Pokud žádný rámec v `LF` není k dispozici, dojde k chybě 55.

DEFVAR ⟨*var*⟩ Definuj novou proměnnou v rámci

Definuje proměnnou v určeném rámci dle ⟨*var*⟩. 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 ⟨*label*⟩ 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 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.

6.4.2 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 ⟨*symb*⟩ Vlož hodnotu na vrchol datového zásobníku

Uloží hodnotu ⟨*symb*⟩ na datový zásobník.

POPS ⟨*var*⟩ Vyjmi hodnotu z vrcholu datového zásobníku

Není-li zásobník prázdný, vyjme z něj hodnotu a uloží ji do proměnné ⟨*var*⟩, jinak dojde k chybě 56.

6.4.3 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 ⟨*var*⟩ ⟨$symb_1$⟩ ⟨$symb_2$⟩ Součet dvou číselných hodnot

Sečte ⟨$symb_1$⟩ a ⟨$symb_2$⟩ (musí být typu int) a výslednou hodnotu téhož typu uloží do proměnné ⟨*var*⟩.

SUB ⟨*var*⟩ ⟨$symb_1$⟩ ⟨$symb_2$⟩ Odečítání dvou číselných hodnot

Odečte ⟨$symb_2$⟩ od ⟨$symb_1$⟩ (musí být typu int) a výslednou hodnotu téhož typu uloží do proměnné ⟨*var*⟩.

MUL ⟨*var*⟩ ⟨$symb_1$⟩ ⟨$symb_2$⟩ Násobení dvou číselných hodnot

Vynásobí ⟨$symb_1$⟩ a ⟨$symb_2$⟩ (musí být typu int) a výslednou hodnotu téhož typu uloží do proměnné ⟨*var*⟩.

IDIV ⟨*var*⟩ ⟨$symb_1$⟩ ⟨$symb_2$⟩ Dělení dvou celočíselných hodnot

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

LT/GT/EQ ⟨*var*⟩ ⟨*symb*$_1$⟩ ⟨*symb*$_2$⟩ Relační operátory menší, větší, rovno

Instrukce vyhodnotí relační operátor mezi ⟨*symb*$_1$⟩ a ⟨*symb*$_2$⟩ (stejného typu; int, bool nebo string) a do ⟨*var*⟩ 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 ⟨*var*⟩ ⟨*symb*$_1$⟩ ⟨*symb*$_2$⟩ Základní booleovské operátory

Aplikuje konjunkci (logické A)/disjunkci (logické NEBO) na operandy typu bool ⟨*symb*$_1$⟩ a ⟨*symb*$_2$⟩ nebo negaci na ⟨*symb*$_1$⟩ (NOT má pouze 2 operandy) a výsledek typu bool zapíše do ⟨*var*⟩.

INT2CHAR ⟨*var*⟩ ⟨*symb*⟩ Převod celého čísla na znak

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

STRI2INT ⟨*var*⟩ ⟨*symb*$_1$⟩ ⟨*symb*$_2$⟩ Ordinální hodnota znaku

Do ⟨*var*⟩ uloží ordinální hodnotu znaku (dle Unicode) v řetězci ⟨*symb*$_1$⟩ na pozici ⟨*symb*$_2$⟩ (indexováno od nuly). Indexace mimo daný řetězec vede na chybu 58. Viz funkce `ord` v Python 3.

6.4.4 Vstupně-výstupní instrukce

READ ⟨*var*⟩ ⟨*type*⟩ Načtení hodnoty ze standardního vstupu

Načte jednu hodnotu dle zadaného typu ⟨*type*⟩ ∈ {int, string, bool} a uloží tuto hodnotu do proměnné ⟨*var*⟩. Načtení proveďte vestavěnou funkcí `input()` jazyka Python 3, pak proveďte konverzi na specifikovaný typ ⟨*type*⟩. Při převodu vstupu na typ bool nezáleží na velikosti písmen a řetězec „true“ se převádí na `bool@true`, vše ostatní na `bool@false`. V případě chybného nebo chybějícího vstupu bude do proměnné ⟨*var*⟩ uložena hodnota `nil@nil`.

WRITE ⟨*symb*⟩ Výpis hodnoty na standardní výstup

Vypíše hodnotu ⟨*symb*⟩ 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.

6.4.5 Práce s řetězci

CONCAT ⟨*var*⟩ ⟨*symb*$_1$⟩ ⟨*symb*$_2$⟩ Konkatenace dvou řetězců

Do proměnné ⟨*var*⟩ uloží řetězec vzniklý konkatenací dvou řetězcových operandů ⟨*symb*$_1$⟩ a ⟨*symb*$_2$⟩ (jiné typy nejsou povoleny).

STRLEN ⟨*var*⟩ ⟨*symb*⟩ Zjisti délku řetězce

Zjistí počet znaků (délku) řetězce v ⟨*symb*⟩ a tato délka je uložena jako celé číslo do ⟨*var*⟩.

GETCHAR ⟨*var*⟩ ⟨*symb*$_1$⟩ ⟨*symb*$_2$⟩ Vrať znak řetězce

Do ⟨*var*⟩ uloží řetězec z jednoho znaku v řetězci ⟨*symb*$_1$⟩ na pozici ⟨*symb*$_2$⟩ (indexováno celým číslem od nuly). Indexace mimo daný řetězec vede na chybu 58.

SETCHAR ⟨*var*⟩ ⟨*symb*$_1$⟩ ⟨*symb*$_2$⟩ Změň znak řetězce

Zmodifikuje znak řetězce uloženého v proměnné ⟨*var*⟩ na pozici ⟨*symb*$_1$⟩ (indexováno celočíselně od nuly) na znak v řetězci ⟨*symb*$_2$⟩ (první znak, pokud obsahuje ⟨*symb*$_2$⟩ více znaků). Výsledný řetězec je opět uložen do ⟨*var*⟩. Při indexaci mimo řetězec ⟨*var*⟩ nebo v případě prázdného řetězce v ⟨*symb*$_2$⟩ dojde k chybě 58.

6.4.6 Práce s typy

TYPE ⟨*var*⟩ ⟨*symb*⟩ — Zjisti typ daného symbolu

Dynamicky zjistí typ symbolu ⟨*symb*⟩ a do ⟨*var*⟩ zapíše řetězec značící tento typ (int, bool, string nebo nil). Je-li ⟨*symb*⟩ neinicializovaná proměnná, označí její typ prázdným řetězcem.

6.4.7 Instrukce pro řízení toku programu

Neterminál ⟨*label*⟩ označuje návěští, které slouží pro označení pozice v kódu IPPcode20. V případě skoku na neexistující návěští dojde k chybě 52.

LABEL ⟨*label*⟩ — Definice návěští

Speciální instrukce označující pomocí návěští ⟨*label*⟩ důležitou pozici v kódu jako potenciální cíl libovolné skokové instrukce. Pokus o redefinici existujícího návěští je chybou 52.

JUMP ⟨*label*⟩ — Nepodmíněný skok na návěští

Provede nepodmíněný skok na zadané návěští ⟨*label*⟩.

JUMPIFEQ ⟨*label*⟩ ⟨$symb_1$⟩ ⟨$symb_2$⟩ — Podmíněný skok na návěští při rovnosti

Pokud jsou ⟨$symb_1$⟩ a ⟨$symb_2$⟩ stejného typu nebo je některý operand nil (jinak chyba 53) a zároveň se jejich hodnoty rovnají, tak provede skok na návěští ⟨*label*⟩.

JUMPIFNEQ ⟨*label*⟩ ⟨$symb_1$⟩ ⟨$symb_2$⟩ — Podmíněný skok na návěští při nerovnosti

Jsou-li ⟨$symb_1$⟩ a ⟨$symb_2$⟩ stejného typu nebo je některý operand nil (jinak chyba 53), tak v případě různých hodnot provede skok na návěští ⟨*label*⟩.

EXIT ⟨*symb*⟩ — Ukončení interpretace s návratovým kódem

Ukončí vykonávání programu a ukončí interpret s návratovým kódem ⟨*symb*⟩, kde ⟨*symb*⟩ je celé číslo v intervalu 0 až 49 (včetně). Nevalidní celočíselná hodnota ⟨*symb*⟩ vede na chybu 57.

6.4.8 Ladicí instrukce

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

DPRINT ⟨*symb*⟩ — Výpis hodnoty na `stderr`

Předpokládá se, že vypíše zadanou hodnotu ⟨*symb*⟩ 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 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: `http://www.a7soft.com/jexamxml.html`

[3] Křivka, Z., a kol.: Zadání projektu z předmětu IFJ a IAL. c2019 [cit. 2020-01-30]. Dostupné z: `https://www.fit.vutbr.cz/study/courses/IFJ/private/projekt/ifj2019.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`