Implementační dokumentace k 2. úloze do IPP 2020/2021
Jméno a příjmení: Ondřej Sloup
Login: xsloup02

Skript test.php

Velmi jednoduchý PHP script, který spouští jednotlivé testy na základě daných parametrů. Script prohledá uživatelem zvolenou složku (ať už rekurzivně či nikoli) a hledá všechny soubory s koncovkou ".src". Poté zjišťuje existenci ostatních souborů a dovytváří je, pokud jsou potřeba. Při kontrole volá jednotlivé scripty `parse.php` či `interpret.py` přes PHP funkci `exec` a ověřuje vrácenou hodnotu a jejich výstup na stdout, který je porovnán přes „`diff`" či „`jexamxml`". Script generuje dočasné soubory, které poté nahrává do dalších porovnávacích programů. Jednotlivý výstup poté ukládá do pole se všemi potřebnými hodnotami. Tyto hodnoty jsou použity pro generování jednoduchého a přehledného HTML na stdout (Je zde zakomentovaná funkce pro výstup HTML do souboru).

Funkce

- `argumentsValidation($argv)` – Jednoduchá funkce pro validaci uživatelských argumentů. Uloží všechny argumenty do dictionary, ze kterého poté kód vychází. Rozhodl jsem se napsat vlastní funkci, protože getOpt() nebylo možné použít tak, aby bylo validní se zadáním (kontrola chybného parametru)
- `createFile($name, $txt)` – Krátká funkce pro vytvoření soboru a vložení textu do něj
- `readTestFile($path)` – Čte soubor na zadané cestě
- `generateXMLStringExplain($value)` – Výsledný kód z porovnávání přeloží na uživatelsky přívětivé ohodnocení výsledku z „`jexamxml`"
- `generateDiffStringExplain($value)` – Podobné funkci `generateXMLStringExplain`, ale pro program `diff`
- `generateTable($array, $name, $mode)` – Generuje HTML tabulku za použití výše zmíněných funkcí
- `generateWeb($tests, $mode)` – Funkce pro generování kompletního HTML kódu
- main($argv) – Hlavní funkce souboru. Obsahuje hlavní cyklus a funkce pro procházení složek a spouští jednotlivé scripty. Rozhoduje se podle uživatelských parametrů, které funkce je nutné provést.

Použité knihovny a třídy

Při psaní scriptu jsem použil dvě knihovny, a to `RecursiveIteratorIterator` a `RecursiveDirectoryIterator` pro procházení složek.

Implementační dokumentace k 2. úloze do IPP 2020/2021
Jméno a příjmení: Ondřej Sloup
Login: xsloup02

Skript interpret.py

Interpret je strukturován jako standartní python package s několika „init.py“ soubory. Moje původní implementace byla vytvořit plně funkční transpilátor – přeložit XML do pythonu a tento kód potom spustit pomocí python funkce exec (v případě tohoto řešení je zde možnost přeložený python script uložit a spouštět s vytvořeným package). Bohužel v pokusném odevzdání neprošel všemi testy kvůli podmíněným skokům, kde bylo nutné dynamicky rozhodnout, zda se jedná o „`while`”, „`if`“ či „`while True with break`“ a proto jsem použil přístup asembleru – postupné spouštění kódu po řádkách.[1]

Kód je strukturován do několika tříd, kde „hlavní“ třída IPPCode21 spouští jednotlivé kontroly, kontroluje jeho tok a samotné spouštění kódu.

Třída Parser má na starosti zpracování a kontrolu XML vstupu. Pomocí funkce: `get_class_by_opcode(opcode: str) -> Union[Instruction, None]` rozpozná kterou instanci třídy má vytvořit pro specifickou instrukci a nahraje do ní zpracované argumenty. Tyto instance jsou uložené v poli, které je iterované a jednotlivé instrukce jsou postupně spouštěny. V případě nalezení instrukce jump, call či podobné, není spuštěná samotná funkce, ale je pozměněn index v tomto poli. Parser také obstarává instrukci label, protože je nutné načíst všechny labely dopředu, aby bylo možné provést skoky vpřed v kódu.

Tyto instance jsou poté spuštěny v třídě Memory pomocí přiřazených funkcí v proměnné `handler_function` a pomocí děděné funkce run instance vrátí přiřazenému handleru veškeré informace zpracované tak, aby s ním mohl pracovat.

Třída Memory dědí z python dictionary a obstarává veškerou zprávu stacku a i stdin, který je do ní načten. Veškerá operace s proměnnými je tímto uzavřena v rámci jedné třídy.

V celém kódu jsem vytvořil exception systém, který volá jednotlivé výjimky a připojuje chybu o zprávě která aktuálně nastala. Při mém minulém řešení jsem vytvořil i logger, který uměl vytáhnout jednotlivé chybové hlášky z funkce exec a propagovat je včetně stacktrace.

[1] Zachoval jsem původní kód a rád bych ho kompletně dokončil či požádal o konzultaci na toto téma, protože toto řešení mi přišlo velmi zajímavé. Bohužel jsem nedokázal přijít na optimální řešení od pokusného odevzdání.