Implementační dokumentace k 2. úloze do IPP 2019/2020

Jméno a příjmení: **Tomáš Dvořáček**

Login: **xdvora3d**

1. Interpret.py

Úvod:

Při implementaci skriptu interpret.py jsem použil v podstatě jen 3 knihovny, a sice getopt pro zpracování argumentů, XML ElementTree pro zpracováni vstupního XML a knihovnu sys pro použití funkce sys.exit().

Použité datové struktury a syntaktická analýza:

Samotná funkcionalita spočívá v tom, že se nejprve celý XML program řádek po řádku uloží do slovníku, kde klíč odpovídá atributu **order** a hodnota je reprezentována **polem elementů** jednoho řádku instrukce. Bohužel se ukázalo, že **order** nejde příliš dobře použít pro postupné zpracování instrukce, protože pořadí na sebe nemusí navazovat. Tento problém jsem vyřešil vytvořením seřazeného pole (podle **order**), kde na každém indexu je výše popsaný slovník, a funkcí, která přepočte **order** na index v poli.

Poté je nad tímto polem slovníků spuštěna syntaktická analýza, která kontroluje počty operandů u instrukcí a ukládá návěští instrukce **LABEL.** V případě konstant se u některých instrukcí kontrolují i datové typy.

Interpretace:

Nakonec je spuštěna samotná interpretace, která je tvořena nekonečným cyklem s postupnou inkrementací výše zmiňovaného pole slovníků. Skokové instrukce jsou opět prováděny pomocí přepočtu **order** na index. V téměř každé instrukci je použita funkce ***work_with_var(),*** která vrací dvojici (hodnota proměnné : typ proměné). Při použití parametru "get" se vrací tato dvojice, při parametru "write" do této dvojice zapisuje.

Jednotlivé datové rámce jsou reprezentovány polem, takže lze s výhodou použít fukce jako append(), pop() pro simulaci zásobníku.

Pro převody datových typů jsem vytvořil funkce *isbool(), isstring() a isint()*, které převádějí řetězce na příslušné datové typy (hodnoty proměnných jsou interně uloženy jako řetězec a převádí se na základě uložených typů).

2. Test.php

Zpracování parametrů příkazové řádky:

Skript test.php nejprve vytvoří instanci třídy **Parse** a následně se zavolá její metoda *arguments()*, která zpracuje parametry příkazové řádky a případné argumenty uloží do statických atributů této třídy. V případě chybných parametrů nebo jejich nepovolených kombinací se skript ukončí **návratovou hodnotou 10.**

Nastavení základního adresáře, cest ke skriptům a základu HTML:

Zavolá se funkce *open_directory()*, jenž v případě zadaného parametru –2irektory nastaví cestu ke složce s testy, v případě opačném se jako složky s testy berou složky z aktuálního adresáře. Nastaví se také cesty ke skriptům **parse.php** a **interpret.py.** Poté je vygenerována základní HTML kód složený z HTML hlavičky a základních informací včetně cesty ke složce ze které byly spuštěny testy.

Rekurzivní průchod adresáře a kontrola souborů:

Dále se spustí funkce *test()*, která při zadání parametru –recursive a v případě že mateřský adresář obsahuje podadresáře, rekurzivně volá samu sebe. Pokud se při prohledávání adresáře narazí na soubor s příponou .src, volá se funkce *file_checker(),* která zkontroluje přítomnost souborů s koncovkami .in, .out a .rc. Vygeneruje se část HTML s aktuálním adresářem ze kterého byl soubor .src.

Testování:

Jádrem celého skriptu je funkce *run_test()*, jenž při zadání parametrů typu –int-only, --parse-only spustí požadované testy. Zde používám funkce jako shell_exec, exec, diff a program jexamxml. V případě úspěšného testu se inkrementuje proměnná *passed*, v případě neúspěchu proměnná *failed* a zároveň se vypíše jméno souboru, který neprošel. Na závěr se vygeneruje v HTML malá statistika o celkovém počtu testů v adresáři a počtu úspěchů a neúspěchů.