Implementační dokumentace k projektu do IPP 2017/2018
Jméno a příjmení: Tomáš Kukaň
Login: xkukan00

parser.php

Úvodní skript v jazyce php5.6, který analyzuje kód v jazyce IPPcode18, lexikálně, syntakticky ho kontroluje a generuje z něho XML soubor, jsem se rozhodl dělat bez použití objektově orientovaného návrhu a nedefinoval jsem jedinou třídu. Definoval jsem si tedy různé funkce které postupně volám.
Program po spuštění zpracuje argumenty a pokud jsou validní tak začne zpracovávat jednotlivé řádky ze standartního vstupu. Nejdříve se načte samotná instrukce a poté se ve switchi rozhodne, s jakými argumenty se zavolá funkce `XmlAddArgument` která přidá argument ve formátu xml na standartní výstup.
Na samotné generování XML dokumentu jsem použil knihovnu XMLWriter, která usnadňuje zejména přepis speciálních znaků na znaky které jsou validní v XML, samotné vypisování tagů a atributů a tisk odsazení.
Žádní rozšíření jsem neimplementoval.

interpret.py

Tento skript je hlavní část projektu, který interpretuje kód IPPcode18 uložený v XML ve formátu který generuje parser.php. Interpret nejen kód vykonává ale také během vykonávání kontroluje syntaktickou, lexikální a sémantickou správnost zdrojového kódu.
Vytvořil jsem si několik tříd, které mě zjednodušují ukládání dat (např. Argument, Instruction...), ale skript není čistě objektově orientovaný.
Program po spuštění stejně jako parser nejdříve zpracuje argumenty, ve kterých mimo jiné nalezne cestu ke vstupnímu souboru. Tento soubor se poté načte a pomocí `xml` modulu se zpracuje. Ten reverzně k XMLWriteru z prvního skriptu sám převede speciální znaky a ulehčí mi práci. Mimo to z celého souboru vytvoří strom, skrz který se poté dá jednoduše procházet. Takto zpracovaný soubor zkontroluji a instrukce si načtu do globální proměnné.
Poté program započne zpracovávat instrukce v hlavním cyklu programu. Na počátku se dle instrukce vybere vhodná funkce, je zavolána a vykoná instrukci.
Jelikož téměř pro každou instrukci mám funkci která ji obsluhuje, tak abych zamezil duplikaci kódu při syntaktické kontrole, vytvořil jsem si dekorátor s argumenty `args_check`, který zkontroluje počet argumentů a jejich typ.
Pro implementaci rámců jsem se rozhodl použít Pythonovský slovník, kde jako klíče používám název proměnné. Pro každý rámec mám tedy jeden slovník a při instrukcích PUSHFRAME a POPFRAME si slovník pro dočasné proměnné uložím na konec listu těchto slovníků, zároveň také přidám referenci do proměnné nesoucí referenci na lokální rámec.
Obecný zásobník na hodnoty jsem implementoval stejně jako zásobník rámců, tedy list do kterého přidávám na konec a z konce odebírám.
Stejně jako u první části jsem žádné rozšíření neimplementoval.

test.php

Stejně jako u parse.php jsem se rozhodl projekt nedělat objektově, ale držet se jednoduchých funkcí, protože se tu žádné objekty přímo nenabízí.
Nejdříve zpracuji argumenty a poté si získám cesty ke všem zdrojovým souborům (končící příponou .src). V jedné složce jde lehce využít funkce `glob`, ale pro zanořené získávání cest jsem tuto funkci musel použít tuto funkci rekurzivně. Pro všechny takto získané zdrojové soubory poté zkontroluji jestli existují i ostatní (.in, .rc) a v negativní případě je vytvořím a jejich obsah nastavím na výchozí hodnotu, tedy nulu nebo prázdný soubor.

Pro každý zdrojový soubor potom spouštím parse.php a pokud neskončí s chybou, tak si jeho výstup ukládám do globální proměnné.
Pro všechny testy které parse.php úspěšně zpracoval následně volám interpret.py, jeho výstup si ukládám do dočasného souboru a nakonec tento soubor pomocí unixové utility `diff` porovnám. U všeho kontroluji návratové hodnoty a testuji je.
Nakonec vypíšu všechny výsledky v přehledných tabulkách ve formátu HTML na standartní výstup.
Stejně jako u ostatních částí projektu, ani zde jsem nedělal žádné rozšíření.