Dokumentace k implementaci projektu do IPP
2. Část

Jméno a příjmení: Tomáš Křenek

Login: xkrene15

Interpret.py

Zadání: Cílem projektu bylo implementovat interpret, který načte vygenerovanou xml reprezentaci kódu v jazyce `IPPcode20`, která je výstupem parse.php, a má za úkol ji interpretovat na standartní výstup.

Implementace: Interpret načítá xml ze souboru zadaného u parametru `–source`(argumenty jsou zpracovány pomocí funkce `getopt`), nebo ze standartního vstupu. Pro čtení je využita knihovna `xml.etree.ElementTree`.

Interpret používá 3 hlavní slovníky. Do jednoho jsou načteny instrukce, do druhého hodnoty argumentů a do třetího typy argumentů. Klíče jsou číslovány od 0 do n. Ještě před načtením do slovníků je ověřena lexikální a syntaktická správnost xml, zde jsou využity regulární výrazy z `parse.php`, které mi velice usnadnili práci. Pokud dojde k chybě, návratový kód je 32(při nalezení chyby je vždy na standartní chybový výstup zobrazena krátká informace o tom, k jaké chybě došlo). Pokud je vstupní xml zcela špatně formátováno, je použita výjimka, která vrací chybu 31.

Hlavní částí programu je procházení slovníku s instrukcemi. Podle názvu instrukce se vybere větev, ve které se bude pokračovat. Zde jsou ověřeny správné typy operandů. Poté se zavolá určitá funkce pro vykonání instrukce, jako parametry jsou předány operandy a jejich typy ze slovníků, které mají stejný klíč jako instrukce ve slovníku instrukcí a u některých instrukcí je také předán například parametr `counter,` který určuje aktuální pozici ve slovníku, například pro uložení při instrukci `CALL`, nebo při nesplnění podmínky podmíněného skoku. Některé kratší instrukce nemají implementovanou svoji funkci, ale jsou provedeny rovnou ve větvi programu.

Práce s proměnnými je řešena tak, že jsou implementovány tři slovníky `GF`, `LF` a `TF`, každý pro jeden rámec. Slovník, který obsahuje proměnné globálního rámce je od začátku prázdný slovník, ale lokální a dočasný rámec jsou na začátku `None`, a je třeba je nejdříve nějakou instrukcí inicializovat. V každém rámci je název proměnné uložen jako klíč ve slovníku a jako jeho hodnota je seznam, kde nultý index je typ proměnné a první index je hodnota proměnné. Dále je také implementovaný seznam `data_stack`, který funguje jako datový zásobník pro instrukce `PUSHS` a `POPS` a rozšíření `STACK`. Funkce instrukcí pracují také s některými důležitými funkcemi jako je například `getValFromVar`, která získá hodnotu a typ z proměnné, nebo funkce `putToVar`, která uloží hodnotu a typ do proměnné. Instrukce `READ` je implementována pomocí funkce `input()`, vstup je zadán jako soubor u parametru `–input`, pokud tento parametr není zadán, je soubor v interpretu uložen na standartní vstup a funkce `input()` z něj načítá. Interpret také umí zpracovávat zásobníkové instrukce, které jsou součástí rozšíření, ty vždy jako operandy vezmou prvky na vrcholu datového zásobníku a výsledek uloží opět na vrchol. Funkce WRITE je implementována funkcí `print()` s parametrem `end=''`, aby byl dodržen správný výstupní formát.

Závěr: Tato část projektu se mi zdála velice zajímavá a přínosná.