Implementační dokumentace k 2. úloze do IPP 2021/2022

Jméno a příjmení: Jiří Mládek

Login: xmlade01

Struktura programu

Na začátku zdrojového kódu se nachází pomocné funkce, dále 2 třídy (`Instruction` sloužící pro ukládání a manipulaci s instrukcí, `Program` pro manipulaci s pamětí programu a volání hlavních funkcí). Po nich je zde několik funkcí přímo pro instrukční sadu, které jsou ve formátu „do_nazevInstrukce“. Na konci zdrojového souboru se nachází funkce main, která postupně provádí chod programu.

Průběh skriptu

Program začíná funkcí `main`, a to vytvořením instance třídy Program `prog`. Tato instance slouží pro volání hlavních metod a také pro správu paměti programu. Globální paměťový rámec a dočasný paměťový rámec jsem implementoval jako slovník, jehož klíčem je název proměnné, a hodnotou je list ve formátu „[datovy_typ, hodnota promenne]“ . Lokální paměťový rámec jsem poté implementoval jako list obsahující slovníky, které mají stejnou strukturu jako u GF a TF. Zásobník a zásobník volání jsem implementoval jako list.

Nejprve je nutné zkontrolovat a rozparsovat argumenty programu metodou `parse_args()`. Zde byla využita knihovna `argparse`, jelikož je vhodná pro práci s více argumenty a vypisování nápovědy. Dochází zde také ke kontrole existence souborů.

Dále metodou `parse_xml()` dochází k rozparsování source souboru s XML reprezentací. Pro tento účel jsem využil knihovnu `xml.etree.ElementTree`. Zde dochází ke kontrole správnosti XML zápisu a také k volání funkce `check_tag()`, která kontroluje, zda má určitý root tag pouze povolené atributy.

Dále je volána metoda load_instructions(), která iteruje přes XML, kontroluje správnost tagů a jejich atributů. Postupně pro všechny instrukce vytváří instanci třídy `Instruction`, do které se uloží operační kód, pořadí instrukce a její argumenty. Argumenty jsou ukládány instanční metodou `load_arguments()`, která s pomocí regulárních výrazů uloží potřebná data. Pokud je hodnota argumentu datového typu string, tak musí dojít k nahrazení escape sekvencí příslušným znakem. Implementace proběhla tímto způsobem: nejprve pomocí regulárního výrazu najdu všechny výskyty escape sekvence ve stringu a přes ně iteruji. V nalezené skupině odstraním zpětné lomítko a poté může být číslo pomocí funkce `chr()` převedeno na znak a být zapsáno do původního stringu.

Jelikož jsou instance ukládány do `listu` a ve třídě `instruction` existuje třídní metoda `get_instructions_list()`, tak k instrukcím mohu přistoupit i později. Dále dochází ke kontrole správnosti pořadí instrukcí a poté k seřazení instrukcí v listu (podle pořadí order) metodou `sort_instructions()`. Poté ještě dochází k projití listu a nalezení labelů, aby při samotném vykonávání skoků byly již známy umístění návěští.

Zavoláním metody `execute_instructions()` dochází k postupnému procházení již seřazených instrukcí. Zavolání konkrétní funkce pro instrukci podle operačního kódu jsem implementoval takto:

```
index, ignore_increment = map_opcode[op_code](index, arguments, ignore_increment, instruction)
```

Operační kód je klíčem do slovníku, ve kterém jsou hodnotou konkrétní názvy funkcí (např. „DEFVAR“ se mapuje na „do_defvar“). Některé z těchto funkcí mění přirozený chod cyklem (skokové instrukce), proto mohou měnit `index` a bool hodnotu `ignore_increment`, které se starají o skok na správnou pozici v listu instrukcí.

Většina funkcí pro instrukce má společný logický základ. Popíšu zde tedy průběh jedné funkce, a to konkrétně funkce na sčítání **ADD** (implementováno jako `do_add()`). Je známo, že funkce má 3 operandy, konkrétně jeden pro uložení výsledku a 2 pro výpočet. Nejprve jsou zpracovány operandy pro výpočet. Jsou to symboly, o kterých pomocí funkce `identify_symbol()` zjistím, zda se jedná o proměnnou nebo konstantu (pomocí regexů – knihovna `re`). Tuto informaci poté předám funkci `get_symbol_dtype_value()`, která mi zjistí datový typ a hodnotu symbolu. Nyní tedy může dojít k výpočtu výsledku. Posledním krokem je uložení hodnoty výsledku a

jeho datového typu do proměnné. O to se stará funkce store_to_var(), která po kontrole existence proměnné uloží potřebné do proměnné v určeném rámci.

Při projití celého listu instrukcí dojde k opuštění metody `execute_instructions()` a ukončení programu.

Rozšíření NVI

Při implementaci jsem se snažil využívat objektově orientovaného programování. Jak jsem již psal, tak v programu používám 2 třídy (třídu Program a třídu Instruction). U třídy Program jsem využil návrhového vzoru **Singleton**. Důvodem byla potřeba sdílet jedinou instanci mezi několika objekty a bloky a zajistit tak vlastně globální přístup k této třídě.