Projektová dokumentace

Implementace překladače imperativního jazyka IFJ20

Tým 55, varianta 1

9/12/2020

Stepaniuk Roman Bc.	**(xstepa64)**	%
Pastushenko Vladislav	(xpastu04)	%
Bahdanovich Viktoryia	(xbahda01)	%
Tomason Viktoryia	(xtomas34)	%

1 Úvod

Výpracovaný projekt načítá zdrojový kód zapsaný ve zdrojovém jazyce IFJ20 ze standardního vstupu a generuje výsledný mezikód v jazyce IFJcode20 na standardní výstup nebo vrací odpovídající chybový kód v případě chyby.

Tato dokumentace popisuje návrh, implementaci, způsob práce v týmu.

2 Návrh a implementace

2.1 Lexikální analýza

Lexikální analýza je naimplementovaná ve souboru `scanner.c` `get_token`,

2.2 Syntaktická analýza

`parser.c` `expression.c`

2.3 Zpracování výrazů

Když parser začíná zpracovávat výraz, on vytvoří zásobník pro tokeny, který bude ten výraz obsahovat. Kdy zpracování všech tokenů výrazu bude ukončeno, parser zavolá funkci `sort_to_postfix`, která přijímá zásobník tokenů s výrazem a vytváří nový zásobník tokenů s výrazem v postfixné formě.

Taky, `sort_to_postfix` zkontroluji pokud výraz má chyby a zjisti typ výrazu. `sort_to_postfix` vrací číslo typu výrazu nebo -1 pokud došlo k chybě.

Když výraz bude přeložen, `sort_to_postfix` uvolňuji paměť , a zavolá funkci `generateCode`, která přečte zásobník s výrazem v postfixné formě a generují odpovídající instrukcí Assembleru. Všichni operandy se zapisují do zásobníku assembleru, a operátory volá odpovídající matematické instrukce, ve souladem s typem výrazu. Když `generateCode` ukončí spravovat výraz, on uvolňuje paměť a ukončí práce.

Priorita	Operátor
1	* /
2	+ -
3	< > =< >= == !=

`Tabulka priority operátorů`

2.4 Sémantická analýza

`parser.c`

3 Datové struktury a speciální algoritmy

Vybrali jsme variantu projektu s abstraktnou dátovou strukturou binárný vyhledávácí strom.Tak byly naimplementovany tabulky symbolů. Operace nad binárním stromem jsme prováděli rekurzivně.Ve struktuře `SymTab` máme 3 struktury pro různé druhy tabulek: tabulka symbolů pro funkce `function`,tabulka symbolů pro proměnné `variable`, a tabulka symbolů pro praci s generátorem kódu `genVariable`.

Při vytváření tabulky symbolů pro proměnné jsme se setkáli s následujícím problémem. Proměnné mohou mít stejné názvy, ale se nacházet na různých úrovních. Abychom mohli to rozlišovat, vytvářili jsme nový binární strom na každé úrovni. Tím pádem jsme získali linearní seznam binárních stromů. Každý uzel stromu obsahuje identifikátor, ukazatele na jeho dva podstromy a data. V binárních stromech hledáme pomocí klíče, který je pro nás `token->data`. Implementovali jsme několik funkcí pro práci s tabulkou. To jsou následující funkce: inicializace, přidání nové položky, vložení typu(pro proměnne), přidání vstupních a výstupních argumentů (pro funkci), vyhledání položky, porovnání proměnných, odstranění položky, uvolnění tabulky z paměti.

Pro vestavěné funkce jsme udělali funkce `symTab_for_inbuilt_func` a ručně vytvořili tokeny tak, aby při volání tabulky byly tyto funkce vnořeny.

Funkce pro prací s tabulkou symbolů jsou implementovany ve souboru `symtable.c`. Hlavičkový soubor je `symtable.h`

4 Generování cílového kódu

Funkce pro generování kódu se nachází v souboru `parser.c`. Spuštění těchto funkcí nastává při druhém průchodu kódu. Pro praci s generátorem kódu používáme tabulku symbolů `genVariable`. Máme dva průchody těla funkcí. Při prvním průchodu všechny proměnné zapisujeme do tabulky symbolů. Pro vyhýbání opakované definici proměnné, vytýkáme každou novou definici přes DEFVAR před funkcí.Toto pravidlo se používáme pro cyklus "for"a v případě deklarace proměnných se stejnými názvy na stejné úrovni, ale v různých blocích. Při druhém průchodu dochází k zpracovávání funkcí s již deklarovanými proměnnými.

Při generování funkcí každá funkce má svůj lokální rámec a je tvořena návěštím podle názvu funkce. Před voláním funkce definujeme hodnoty parametrů v dočasném rámci a po vstupu do funkce přesouvá na zásobník rámců a se stává aktuálním lokálním rámcem. Po provedení funkce výsledek je uložen do proměnné s návratovou hodnotou funkce. Vestavěné funkce jsou předepsány přímo v jazyce IFJcode20.Jsou vyvolány na začátku programu.

Nepoužíváme globální rámec. Globálně vyhlašujeme jen podtržitko

5 Práce v týmu

5.1 Způsob práce v týmu a komunikace

Nejprve jsme vytvořili plan a nějakou strukturu projektu. Každý pátek jsme měli "schůzku kde jsme analyzovali, co bylo provedeno, co je třeba změnit a rozhodnout nad čím pracovat dál. Na jednotlivých ulohách jsme pracovali jednotlivě nebo dvojice členů týmu. A pořad jsme konzultovali mezi sebou.

Komunikace mezi členy týmů probíhala na začátku osobně, pak prostřednictvím aplikace Discord, kde jsme měli skupinové konverzace nebo psali přímo mezi sebou. Tam jsme probírali a řešili problémy týkající se různých částí projektu.

5.1.1 Verzovací systém

Pro správu souborů projektu jako verzovací systém jsme zvolili Git. Jako vzdálený repositář jsme používali GitHub.

Každý z nás měl svou vlastní větev. Díky GitHubu jsme mohli pracovat na více úkolech současně. Při změnách jsme si nechali komentáře, co bylo změněno. Po týmovém schválení a otestování jsme tyto úpravy spojovali do hlavní větve.

5.2 Rozdělení práce mezi členy týmu

Práci jsme rozdělili následovně:

6 Závěr

Použitá literatura

[1] Slajdy z přednášek předmětu Formální jazyky a překladače

[2] Slajdy z přednášek předmětu Algoritmy

[3] Wiki stránka o binárním stromu
https://cs.wikipedia.org/wiki/Binární_vyhledávací_strom