Implementace překladače imperativního jazyka IFJ17

Tým 074, varianta II

Implementovaná rozšíření
UNARY, BASE, FUNEXP, IFTHEN, BOOLOP

Daniel Dolejška	xdolej08, vedoucí	40%
Petr Ullrich	xullri00	30%
Lukáš Kulda	xkulda01	30%

Contents

List of Figures

1 Podrobnosti spolupráce

1.1 Rozdělení práce

Daniel Dolejška	40%	syntaktický a sémantický analyzátor, testování, dokumentace
Petr Ullrich	30%	lexikální analyzátor, dokumentace
Lukáš Kulda	30%	generátor, dokumentace

K nerovnoměrnému rozdělení bodů došlo z důvodu rozdílného rozsahu a náročnosti jednotlivých zpracovávaných modulů.

1.2 Komunikace a splupráce

Komunikace probíhala osobně a pomocí GitLabu, kde měl každý člen týmu vlastní repozitář a mohl tak průběžně ukládat změny a také stahovat změny od ostatních. Každý tak měl možnost vidět kdo, kdy provedl jaké změny.

2 Lexikální analýza

Úlohou lexikálního analyzátoru je rozpoznat jednotlivé vstupní znaky a vytvořit z nich příslušné tokeny.

2.1 Řešení

Hlavní funkcí ovládající lexikální analyzátor je **Scanner_GetToken**. Ta cyklicky načítá jednotlivé znaky se standartního vstupu a na základě jejich obsahu mění svůj stav a postupně vytváří token a jeho obsah.

Cyklus načítání je ukončen v případě, kdy obsah načteného znaku nevyhovuje pravidlům aktuálního stavu automatu. Pokud se automat nachází v konečném stavu nově načtený znak již není součástí aktuálního tokenu a je vrácen tak, aby mohl být načten při dalším volání této funkce, vzniklý token je přes ukazatel v parametru vrácen volající funkci. V případě, kdy se automat v konečném stavu nenachází jedná se o lexikální chybu.

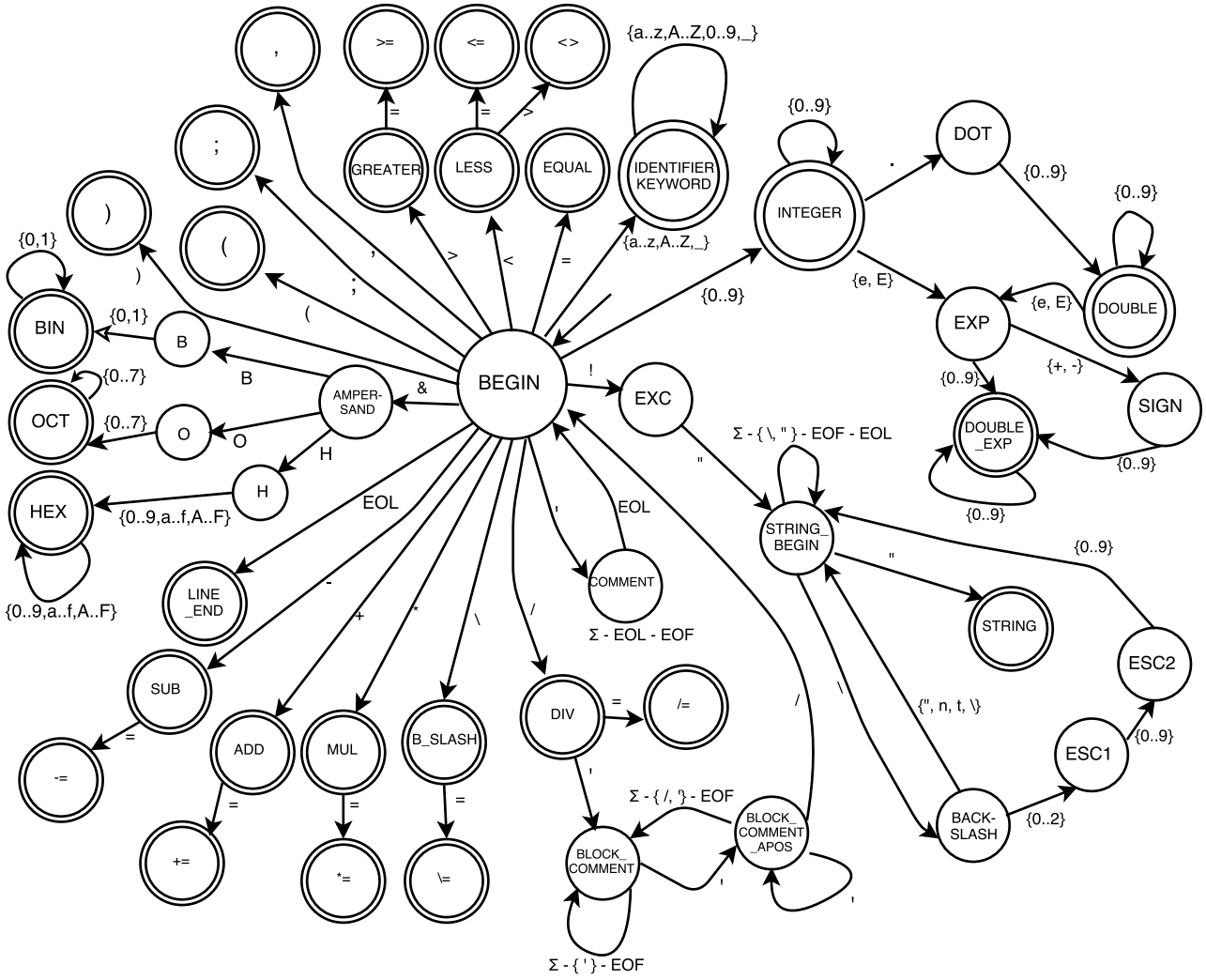
Po ukončení funkce se vrací číselná hodnota značící zda bylo zpracování úspěšně dokončeno či ne (zastoupena enumerátorem ErrorCodes). Token obsahuje informaci o svém typu a příslušnou hodnotu, která je reprezentována řetězcem znaků a je použita v dalších částech překladače.

Další implementovanou funkcí je **Scanner_UngetToken**, která slouží k vrácení již vytořeného tokenu. Tato funkce předaný token uloží na zásobník tokenů a při dalším volání funkce **Scaneer_GetToken** se vrátí již vytvořený token uložený na vrcholu zásobníku, namísto vytváření nového.

2.2 Konečný automat

Implementace lexikálního analyzátoru je založena na konečném automatu, který čte jednotlivé vstupní znaky a skládá je do příslušných tokenů. Diagram implementovaného KA je obrázek 1 na straně 4.

Dojde-li k lexikální chybě, typ tokenu je nastaven na INVALID a jeho hodnota upřesňuje danou chybu.



Obrázek 1: Diagram konečného automatu

3 Syntaktická a sémantická analýza

Soubor funkcí kontrolující syntaktickou a sémantickou správnost vstupního kódu IFJ17.

3.1 Princip

Syntaktický a sémantický analyzátor se skládá z několika funkcí, které odpovídají jednotlivým možným strukturám v jazyce IFJ17. Každá daná funkce načítá tokeny z lexikálního analyzátoru, které následně kontroluje, dále zpracovává a vytváří adekvátní kostru v mezikódu IFJcode17 voláním funkcí generátoru, popřípadě volá další funkce syntaktického a sémantického analyzátoru, které zpracovávají potřebné použité struktury (výrazy, vnořené příkazy apod.).

3.2 Řešení

Syntaktický a sémantický analyzátor se skládá z 19 hlavních funkcí, které slouží ke kontrole syntaktické a sémantické správnosti vstupního kódu a několika dalších pomocných funkcí, které jsou používány pro zjednodušení opakovaných akcí napříč všemi hlavními funkcemi.

3.2.1 Spuštění překladu

Zpracování začíná v hlavní funkci **Parser_ParseInitial**, která vytvoří výchozí úvodní instrukce výstupního programu a dále cyklicky zpracovává příchozí tokeny. Tato úvodní funkce pracuje na stejném principu jako funkce pro zpracování vnořeného kódu, přijímá pouze značně omezený seznam typů tokenů: *deklarace funkce*, *definice funkce*, *definice scope*.

Po přijetí očekávaných klíčových slov volá odpovídající funkce, které načítají další tokeny, kontrolují jejich správnost a dále je zpracovávají. Při přijetí tokenu neočekávaného typu analyzuje chybu a vrací odpovídající chybový kód hlavní funkci celého programu.

3.2.2 Vnořený kód

Funkce pro zpracování obecného vnořeného kódu (**Parser_ParseNestedCode**) slouží jako křižovatka, která na základě prvního příchozího tokenu volá odpovídající funkci pro zpracování očekávané struktury.

Dále také rozhoduje, zda je vůbec možné v dané situaci danou strukturu použít – tato rozhodnutí dělá především na základě seznamu zanoření, který určuje, v jaké struktuře se aktuálně vnořený kód zpracovává (zda se jedná o větev podmínky, cyklus či funkci).

V případě, že tato funkce obdrží token, který by mohl mít speciální význam – klíčové slovo, které nebylo očekáváno a pro vnořený kód nemá žádný smysl (ovšem může se jednat o ukončení nadřazené struktury – konec definice funkce, cyklu apod.), vrací se o úroveň výše, a funkci, která jej zavolala, tento token předává. Ta rozhodne o jeho správnosti.

3.2.3 Zpracování funkcí

O zpracování funkcí v programu se starají 3 funkce, kde každá z nich má konkrétní účel – jedna funkce pro deklaraci funkce (**Parser_ParseFunctionDeclaration**), jedna pro její definici (**Parser_ParseFunctionDefinition**) a jedna zpracovává volání těchto funkcí (**Parser_ParseFunctionCall**).

Funkce pro deklaraci a definici funkcí jsou volány z hlavní úvodní funkce syntaktického a sémantického analyzátoru. Funkce pro volání funkcí jsou volány z funkcí pro zpracování výrazů.

3.2.4 Zpracování podmínek

Podmínky zpracovávají dvě funkce – první (**Parser_ParseCondition**), která zpracovává prvotní podmínku a ELSE bez další následující podmínky, a druhá (**Parser_ParseSubCondition**), která zpracovává další libovolný počet navazujících ELSEIF podmínek.

Druhá funkce pro zpracování navazujících ELSEIF podmínek zpracovává pouze tyto typy – v případě, že obdrží token značící kompletní ukončení podmínky či začátek ELSE větve, vrací se postupně na nejvyšší úroveň, dokud se nevrátí do první funkce, která daný token zpracuje sama.

Tyto funkce dále využívají funkci pro zpracování výrazů (k vyhodnocení pravdivostní hodnoty v podmínce) a funkci pro zpracování vnořeného kódu.

3.2.5 Zpracování cyklů

Funkce pro zpracování cyklu DO WHILE (**Parser_ParseLoop_Do**) funguje principiálně stejně jako všechny ostatní hlavní funkce. Využívá funkcí pro zpracování pravdivostních výrazů a funkci pro zpracování vnořeného kódu pro zpracování instrukcí v cyklu.

Cyklus je složen z dvou návěstí, počátečního a konečného. Po počátečním návěstí následují instrukce pro vyhodnocení výrazu a podmíněný skok na konečné návěstí v případě nesplnění podmínky.

3.2.6 Zpracování výrazů

Pro zpracování funkcí slouží dvě funkce, první funkce (**Parser_ParseExpression**) ke zpracování hlavního výrazu a druhá funkce (**Parser_ParseSubExpression**) která zpracovává podvýrazy v závorkách.

OPERÁTOR	PRIORITA
*, /	1
\	2
+, -	3
<, <=, <>, >, >=, =	4
NOT	5
AND, OR	6

Obrázek 2: Precedenční tabulka operátorů

Tyto funkce využívají funkce pro převod příchozích tokenů na postfixový výraz. Po jednom dané tokeny funkcím odesílají a ty je zpracovávají a vytvářejí z nich postfixový výraz. Po přijetí tokenu neočekávaného typu (například neočekávané klíčové slovo nebo, EOL) je výraz ukončen a vyhodnocen.

Po ukončení postfixového výrazu jsou volány funkce na převod tohoto výrazu na instrukce, ty na základě informací v položkách postfixového seznamu vytvoří seznam instrukcí pro zpracování výrazu.

Tento převod probíhá ve dvou částech – první zpracování vzniklého postfixového řetězce vytvoří odpovídající seznam instrukcí na základě příchozích proměnných a operací. Druhé zpracování validuje správnost datových typů v operacích ve vzniklé posloupnosti instrukcí a v případě nutnosti provést implicitní převod datových typů, vloží mezi ostatní instrukce potřebnou instrukci pro požadovaný převod datových typů. Výsledné instrukce jsou pak ze speciálního seznamu překopírovány do hlavního seznamu instrukcí.

3.3 Pomocné datové struktury

Syntaktický a sémantický analyzátor využívá několik pomocných datových struktur pro jednodušší předávání informací mezi funkcemi. Mezi ně patří zásobník, seznam či tabulka s rozptýlenými položkami.

3.3.1 Zásobník

Zásobník je v hojném použití při převodu příchozích tokenů na postfixový výraz. Dále je také využíván při zpracování volání funkcí ve výrazech, k uložení tokenů popisujících parametry dané funkce či při „vracení“ tokenů lexikálnímu analyzátoru.

3.3.2 Seznam

V tomto projektu jsou využity jednosměrně i dvousměrně vázané seznamy. Mezi zástupce jednosměrně vázaného seznamu patří postfixový seznam, ve kterém je uložen zpracovaný postfixový výraz, seznam tokenů či seznam zanoření, který je používán především funkcí pro zpracování vnořeného kódu.

3.3.3 Tabulka s rozptýlenými položkami

Tato struktura byla použita při implementaci tabulky symbolů překladače na základě dané varianty zadání.

4 Generátor

Vytváří, ukládá a dále pracuje s instrukcemi mezikódu IFJcode17.

4.1 Řešení

Pomocí implementovaných funkcí pro skládání řetězců vytvoří instrukci jako jeden řetězec z názvu instrukce a případně jednoho či více symbolů obsahujících informace o jednotlivých argumentech, který následně uloží jako poslední prvek do dvousměrně vázaného seznamu. Tyto funkce sice umožňují snazší složení řetězců, ale vytváří tak alokované textové řetězce, které je potřeba následně uvolnit.

Každá instrukce mezikódu má svou vlastní funkci, která přijímá pouze argumenty, které daná instrukce vyžaduje. Při výpisu tiskne jednotlivé instrukce na standartní výstup, každou na nový řádek tak, jak jsou v seznamu uloženy.

4.2 Pomocné datové struktury

Tento modul využívá dvousměrně vázaný seznam pro ukládání vytvořených instrukcí a jejich následný výpis.