Dokumentace projektu IFJ

Tým 20, varianta a/2/II

(vedoucí *)Jan Žárský – xzarsky03 1/5

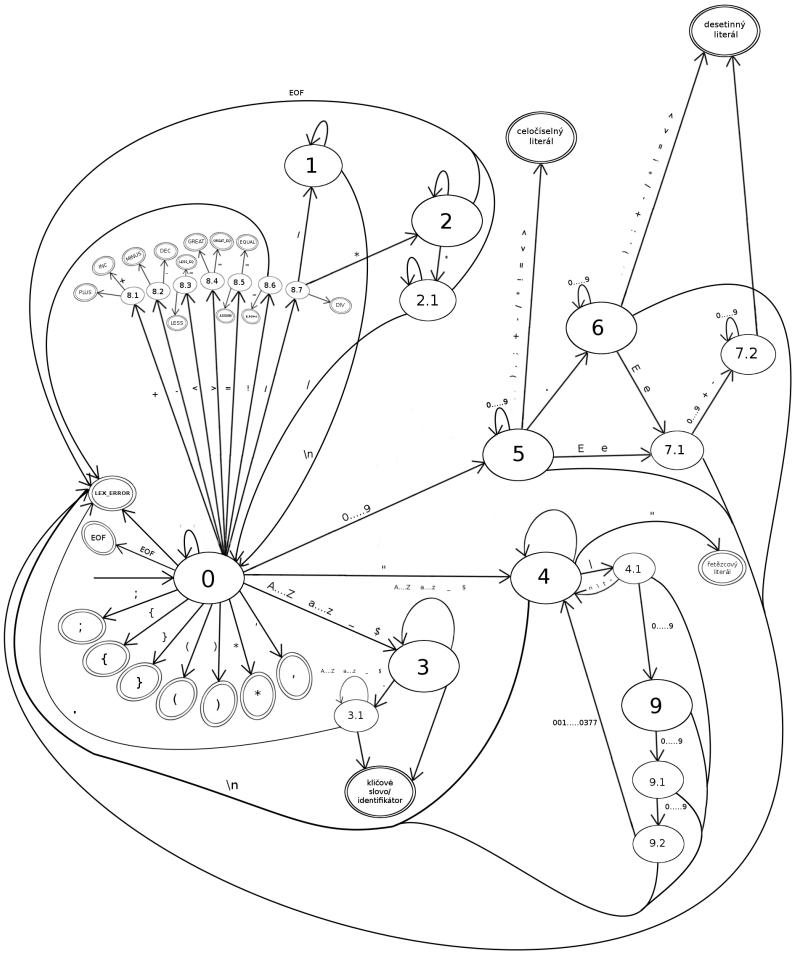
David Vlček – xvlcek23 1/5

Lucie Pelantová – xpelan04 1/5

Andrei Paplauski – xpapla00 1/5

Vít Mrlík – xmrlik00 1/5

Diagram konečného automatu



Legenda

- počáteční stav automatu
- stav automatu
- koncový stav automatu
- neprázdný přechod, možné vstupy jsou odděleny mezerou, např. *0....9 abc 5478*
- prázdný přechod vyjadřující "else", tzn. vstupem je jakýkoliv jiný znak, který již není obsažený v některém z neprázdných přechodů stavu automatu
- každý stav má max. 1 prázdný přechod

LL – gramatika

```
E -> E + E

E -> E – E

E -> E / E

E -> (E)

E -> ID

E -> INT

E -> DOUBLE

E -> STRING

E -> E < E

E -> E > E

E -> E <= E

E -> E >= E

E -> E == E

E -> E != E
```

Precedenční tabulka

	+ -	/ *	Rel	(	)	ID	$
+ -	>	<	>	<	>	<	>
/ *	>	>	>	<	>	<	>
Rel	<	<		<	>	<	>
(	<	<	<	<	=	<	
)	>	>	>		>		>
ID	>	>	>		>		>
$	<	<	<	<		<	

Popis způsobu řešení projektu

Lexikální analýza (scanner): Scanner používaný pro lexikální analýzu je realizován jako deterministický konečný automat pomocí konstrukce „switch“ v jazyce C.
K rozpoznání tokenu na vstupu jsme se rozhodli využít metodu načítání po jednom znaku ze vstupního souboru a jeho následné uložení do předem definované struktury jak bylo ilustrováno v ukázkovém projektu „Zjednodušená implementace interpretu jednoduchého jazyka“.
Největší překážkou při řešení scanneru bylo nejspíš odstranění nedeterminismu ve výsledném automatu.

Tabulka symbolů:

K sestavení tabulky symbolů jsme použili metody prezentované v rámci předmětu IAL.
Při testování správnosti tabulky jsme se rozhodli provést několik změn zejména v rámci optimalizace a snížení paměťových nároků např. Použití pole ukazatelů na položky namísto pole samotných položek.

Naše výsledná tabulka obsahuje následující funkce:

-Hashovací funkce

-Inicializace tabulky

-Přidávání položky

-Vyhledání položky

-Uvolnění tabulky

Syntaktická a sémantická analýza:

Vestavěné funkce:

Interpret:

Vstupem interpretu je seznam instrukcí, které vykonává v cyklu dokud nenarazí na konec seznamu.

Pro snadnější manipulaci s interpretem, jsme se rozhodli, že ne vždy budeme předávat adresy jako parametry instrukce, ale pro předávání hodnot využijeme zásobník.
Tento zásobník také využíváme pro realizaci aritmetických operací.
Při návrhu interpretu jsme narazili na problém, jak realizovat vnoření podmínek.
Pro tento účel používáme zásobník bool hodnot, pomocí kterého vyhodnocujeme, na jaké místo v listu instrukcí interpret skočí.

Popis využitých algoritmů

Vestavěné funkce:

Tabulka symbolů:

-hash_function

Při implementaci rozptylovací funkce, jsme použili algoritmus, který pro nalezení klíče využívá součet jednotlivých položek řetězce vždy násobených konstantou. V našem řešení projektu tento řetězec představuje identifikátor položky. Po krátkém hledání nejlepší konstanty, pro tento algoritmus, jsme nalezli prvočíslo 65599, které zajišťuje velice dobré rozptýlení položek v tabulce.

-st_init

Pro inicializaci pole tabulky symbolů, jsme zvolili co nejjednodušší postup, a to cyklus, který všechny položky pole nastaví na NULL.

-st_add

Tato funkce nejprve zavolá rozptylovací funkci a na výsledném klíči nově vzniklou položku zařadí na začátek seznamu synonym. Pro jednoduché další upravování položky tato funkce vrací ukazatel na nově vzniklý prvek.

-st_find

Tato funkce je implementována cyklem, který po vygenerování klíče prohledává zřetězený seznam položek, tak dlouho dokud nenarazí na hledaný prvek nebo konec seznamu. Při práci na této funkci nastala otázka, zda je možné mít jen jednu tabulku a poté v ní vyhledávat poslední výskyt položky. Ale po podrobnějším průzkumu, jsme zjistili, že by nebylo možné ošetřit opětovnou deklaraci proměnné, a proto jsme od této možnosti ustoupili.

-st_find_global

Výše popsaná vyhledávací funkce neumožňuje vyhledávání identifikátorů, které obsahují i název třídy, ve které se nacházejí. Proto tato funkce nejdříve před samotný identifikátor přidá název třídy a až tento řetězec použije jako parametr předchozí vyhledávací funkce.

-st_free

Pro zajištění uvolnění veškeré paměti alokované pro tabulku, jsme museli správně zacházet s ukazateli, tak abychom neztratili žádný odkaz na neuvolněnou položku. Z tohoto důvodu tato funkce nejprve postupně uvolňuje jednotlivé položky zřetězených seznamů synonym a nakonec celé pole ukazatelů na položky.

Práce v týmu a rozdělení řešení

Projekt jsme se snažili řešit během semestru průběžně a to jak pomocí online komunikace a využití depozitáře github tak formou pravidelných osobních schůzek ve volném čase, kde jsme na projektu pracovali společně.
Rozdělní jednotlivých částí mezi členy týmu bylo zhruba následující:

Lexikální analýza (xvlcek23), syntaktická a sémantická analýza (xzarsky03, xpapla00), vestavěné funkce (xmrlik00), tabulka symbolů (xpelan04, xvlcek23), interpret (xzarsky03, xpelan04, xvlcek23), celková koordinace a testování (xzarsky03).

Příčemž se dle potřeby členové týmu podíleli i na debugování / pomoci při řešení ostatních částí projektu.

Při dělení práce jsme se také snažili zohlednit pokročilost programování v jazyce C a zájmy členů týmu.

Referenční literatura

V rámci řešení projektu jsme se rozhodli využít následujících referenčních zdrojů, autorům děkujeme!

–Z ukázkového projektu "Zjednodušená implementace interpretu jednoduchého jazyka" jsme využili některé funkce z knihovny pro práci s řetězci (soubory string.c a string.h).

–Prvočíslo 65599, které zajišťuje velice dobré rozptýlení položek v tabulce jsme nalezli na http://www.cse.yorku.ca/%7Eoz/hash.html#sdbm kde je popsáno jako magická konstanta, která byla nalezena při experiementování s různými konstantami v rámci práce s hashovacími funkcemi.