VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

IFJ - Projektová dokumentace

Implementace překladače jazyka IFJ22

Tým "Tým xkalis03", varianta BVS

Autoři:

(v)	Vojtěch Kališ (xkalis03)	25%
	Jan Lutonský (xluton02)	25%
	Jan Salaš (xsalas02)	25%
	Lucie Hlaváčová (xhlava60)	25%

Implementovaná rozšíření: FUNEXP

Obsah

1 Úvod

Cílem tohoto projektu bylo vytvořit překladač implementovaný v jazyce C, který ze standartního vstupu načte vstupní kód napsaný v jazyce IFJ22, přeloží jej do cílového jazyka IFJcode22 a výskedek pak vypíše na standartní výstup. Jazyk IFJ22 vznikl jako obdoba jazyka PHP.

2 Implementace

Celý překladač jsme si rozdělili na více dílčích problémů, jejichž funkčnost byla individuálně testována. Tyhle dílčí problémy byly nadále vzájemně propojovány a opět testována jejich funkčnost.

2.1 Lexikální analyzátor

2.2 Syntaktický analyzátor

2.2.1 Precedenční syntaktický analyzátor

2.3 Sémantický analyzátor

Sémantický analyzátor provádí sémantickou analýzu nad vstupním programem, a je obsažen v souboru *semantic.c*; jeho hlavičkový soubor pak analogicky nese název *semantic.h*. Sémantický analyzátor pracuje převážně s Globální tabulkou symbolů (využívající implementace Tabulky symbolů a Abstraktním syntaktickým stromem (dále jen ASS). Očekává se korektní naplnění ASS v rámci syntaktické analýzy. Na začátku své funkce sémantický analyzátor inicializuje Globální tabulku symbolů, projde ASS a vyhledá v něm všechny definice funkcí, jež vloží jakožto nody s typem *function* do Globální tabulky symbolů; možné parametry definované při deklaraci funkce zase vloží do Lokální tabulky symbolů dané funkce jakožto nody s typem *variable*. Do Globální tabulky symbolů jsou také vloženy deklarace vestavěných funkcí (zavoláním funkce), společně s funkcí nazvanou ”:b”sloužící jako hlavní tělo programu (*body*).

Jakmile je vše připraveno, Sémantický analyzátor vstoupí do funkce *AST_DF_traversal*, plnící funkci hlavní smyčky, která prochází již zmíněný AST do hloubky a v rámci switch case-u pak hledá AST nody, jejichž sémantickou korektnost je třeba prověřit; jakmile nějakou takovou nodu najde, spustí nad ní speciální funkci zabývající se prověřením sémantické korektnosti toho konkrétního typu AST nody. V případě, že je nalezena sémantická chyba, program ukončí svou činnost, vypropaguje kód odpovídající nalezené chybě, a zaručí, že dojde ke kompletnímu uvolnění veškeré alokované paměti.

2.4 Generátor cílového kódu

3 Speciální datové struktury

3.1 Tabulka symbolů

Tabulka symbolů byla implementována jako binární vyhledávací strom, což bylo i nárokem naší varianty zadání. Téměř celá tabulka symbolů je napsána nerekurzivním (tedy iterativním) postupem, a to především z důvodu snížení časové komplexity na úkor složitější implementace. Tabulka symbolů je využívána Sémantickým analyzátorem pro vytvoření Globální tabulky symbolů a její využití je již popsáno v rámci jeho popisu.

3.2 Obousměrně vázaný seznam

Implementaci obousměrně vázaného seznamu lze najít v souboru *dll.c*, a odpovídající hlavičkový soubor pak pod názvem *dll.h*. Obousměrně vázaný seznam je v projektu využíván ve struktuře nody Tabulky symbolů, a to pro účely snadného uchování názvů argumentů vkládaných funkcí.

3.3 ADT #3

3.4 ...

4 Práce v týmu

Na projektu jsme začali pracovat ihned po zveřejnění zadání, a to jeho prostudováním a domluvením první schůzky, v rámci které byla vypracována prvotní verze pravidel LL-gramatiky, a LL tabulka. Obojí se ještě v čase dalších několika týdnů upravovalo v případě nalezení chyby, až se nakonec vše ustálilo do konečné podoby, prezentované v tomhle dokumentě a dohledatelné na jeho konci.

Dále jsme si jednotlivé části rozdělili mezi sebe a pracovali na nich jako jednotlivci popřípadě dvojice. Stále probíhaly schůzky, například pro řešení implementačních záležitostí a to především způsobu komunikace jednotlivých částí překladače, ovšem zvětšiny docházelo spíše k průběžnému testování překladače jako celku a kontrole pokroku ve vývoji.

4.1 Komunikace

Pro komunikaci byla využita aplikace *Discord*, kde byl vytvořen vlastní server na kterém pak probíhala veškerá vzájemná komunikace, ať už se jednalo o komunikaci textovou, hovorové schůzky celotýmové i třeba v menším počtu, nebo sdílení materiálů, diagramů apod.

4.2 Vzdálený repozitář

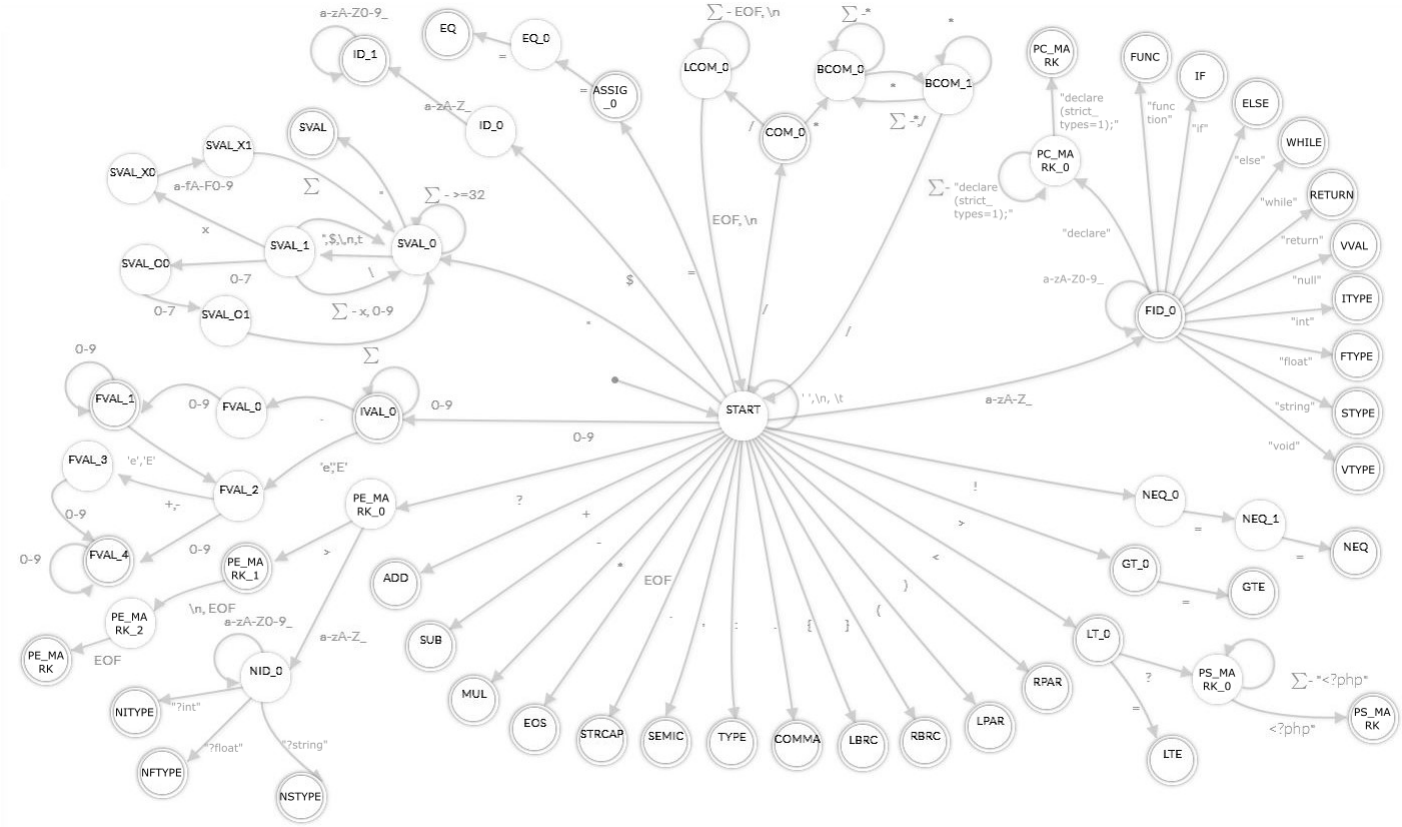
Jako vzdálený repozitář jsme zvolili Github, jenž nám umožnil sdílet mezi sebou zdrojové kódy dílčích úkolů a navzájem si testovat nejen funkčnost jednotlivých částí, ale i překladač jako celek.

4.3 Rozdělení práce v týmu

Rozdělení:
- Jan Lutonský (xluton02): Syntaktická analýza, Precedenční syntaktická analýza, Generátor cílového kódu, Speciální datové struktury, Testování
- Vojtěch Kališ (xkalis03): Sémantická analýza, Tabulka symbolů, Speciální datové struktury, Testování
- Jan Salaš (xsalas02): Lexikální analýza, Testování
- Lucie Hlaváčová (xhlava60): Lexikální analýza, Testování

5 Závěr

DKA pro konečný automat



Precedenční tabulka

```
FID  :  function = {FID}
ID   :  ID
TERM : term = {IVAL|FVAL|SVAL|NUL}
CL1  : class1 = {*| \ }
CL2  : class2 = {+| -|  .}
CL3  : class3 = {<| >| <=| >=}
CL4  : class4 = {===| !==}
```

	FID	ID	TERM	CL1	CL2	CL3	CL4	(	)	,	=	$
F	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	<	ERROR	ERROR	ERROR	ERROR
I	ERROR	ERROR	ERROR	>	>	>	>	ERROR	>	>	<	>
T	ERROR	ERROR	ERROR	>	>	>	>	ERROR	>	>	ERROR	>
CL1	<	<	<	>	>	>	>	<	>	>	ERROR	>
CL2	<	<	<	<	>	>	>	<	>	>	ERROR	>
CL3	<	<	<	<	<	>	>	<	>	>	ERROR	>
CL4	<	<	<	<	<	<	>	<	>	>	ERROR	>
(	<	<	<	<	<	<	<	<	=	<	ERROR	ERROR
)	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	>	ERROR	ERROR
,	<	<	<	<	<	<	<	<	>	>	ERROR	ERROR
=	<	<	<	<	<	<	<	<	>	>	ERROR	>
$	<	<	<	<	<	<	<	<	ERROR	<	<	ACCEPT

Pravidla LL gramatiky

```
prog -> PS_MARK PC_MARK prog_body

prog_body -> body_part prog_body
           | fun_def prog_body
           | EPS

prog_end -> PE_MARK EOS
          | EOS

body -> body_part body
      | EPS

body_part -> if_n
           | while_n
           | extended_expr
           | ret

extended_expr -> EXPR SEMIC
               | EXPR_FCALL SEMIC
               | EXPR_PAR SEMIC
               | EXPR_ASSIGN SEMIC

ret -> RETURN ret_cont
ret_cont -> EXPR SEMIC
          | EXPR_PAR SEMIC
          | EXPR_FCALL SEMIC
          | SEMIC

while_n -> WHILE EXPR_PAR LBRC body RBRC

if_n -> IF EXPR_PAR LBRC body RBRC else_n
else_n -> ELSE LBRC body RBRC
        | EPS

fun_def -> FUNC F_ID LPAR par_list RPAR TYPE ret_type LBRC fun_body RBRC

par_list -> type_n ID par_list_cont
          | EPS

par_list_cont -> COMMA par_list
               | EPS

ret_type -> type_n
          | VTYPE

type_n -> STYPE
        | ITYPE
        | FTYPE
        | NSTYPE
        | NITYPE
        | NFTYPE
```