Katedra informatiky a výpočetní techniky
Fakulta Aplikovaných věd
Západočeská univerzita v Plzni

Překladač jazyka PL0 v jazyce Java

Semestrální práce z předmětu Formální jazyky a překladače (KIV/FJP)

Michal Beran
berny@students.zcu.cz
5.1.2001

Zadání úlohy

Napište v programovacím jazyce Java překladač jazyka PL0. K dispozici jsou dány zdrojové texty překladačů jazyka PL0 v programovacím jazyce Pascal a C.

Překladače a jazyk PL0

Programovací jazyk PL0 byl navržen profesorem Niklause Wirthem z pedagogických důvodů, aby sloužil jako model pro výklad základních principů konstrukcí překladačů. PL0 je jakýmsi kompromisem mezi jednoduchým systémem a rozsáhlým programovacím jazykem (např. Pascal), neboť obsahuje takové konstrukce, které jsou pro výklad základních principů překladačů dostačující. Proto se s některými, dnes běžně užívanými konstrukcemi, v PL0 nesetkáme, neboť by ukázkový model činily těžkopádným a pro samotný výklad tak naprosto nevhodným. To je hlavní důvod, proč kupříkladu PL0 pracuje pouze s jedním datovým typem (celočíselný datový typ); tento je však implicitně dán, a proto se v deklaracích neuvádí). Kompletní syntax jazyka PL0 je ve formě syntaktických diagramů uvedena ve [WIRTH].

Překladač lze programově realizovat dvěma základními způsoby. První z nich spočívá v použití formalismu, který má podobu tabulek, obsahující základní lexikální a syntaktická (popř. sémantická) pravidla, pomocí nichž je příslušný jazyk definován; obecnosti lze o takovém programu hovořit jako o konečném automatu. Program vytvořený tímto způsobem má poměrně jednoduchou strukturu (vzhledem k druhému způsobu realizace, viz dále), avšak čitelnost konstrukcí překladače je v tomto případě velmi nízká, nehledě na to, že další rozšíření jazyka o nové struktury je velmi obtížná.

Druhý způsob upřednostňuje rozšiřitelnost před výhodami formalismu. Program je sice vytvářen více méně intuitivně (intuici chápejme v tomto případě jako opak formalismu), tj. "život" překladače je popisován programovými konstrukcemi hostitelského jazyka (Pascal, C…), avšak v případě potřeby je možné definovat takové programové konstukce, kterými je možno zapsat libovolný syntaktický diagram, viz [WIRTH].

Práci překladače lze do jisté míry rozdělit na *lexikální*, *syntaktickou* a *sémantickou analýzu*. Lexikální analýza resp. analyzátor má za úkol jednoduše řečeno vygenerovat ze vstupního textu posloupnost symbolů, jež bude použita v dalších fázích analýzy, přičemž odstraní takové prvky, jako jsou bílé znaky, komentáře, atd., a vytvoří současně tabulku symbolů. Syntaktická analýza, která pracuje již jen se symboly, kontroluje správnost programové struktury na základě tzv. *gramatiky*. Sématická analýza pak definuje význam jednotlivých symbolů a pravidel. V reálném případě je však velmi obtížné od sebe tyto tři složky analýzy oddělit tak, aby pracovaly v sérii, ve většině případů jsou tyto celky nějakým způsobem provázány. Obdobně tomu bude i v překladači PL0.

Základní vlastnosti jazyka PL0

- v jazyce je definován pouze celočíselný datový typ, který se však při deklaracích neuvádí,
- program sestává z bloků, které mohou být do sebe vnořené, tj. např. podprogramy lze do sebe vhnízdit – lehký náznak modularity (obdobná struktura je i v jazyce Pascal, jazyk C tuto možnost nemá),
- ze strukturovaných příkazů obsahuje PL0 základní podmiňovací příkaz (dílčí příkaz se vykonává pouze při splněné podmínce) a cyklus s testem na začátku.

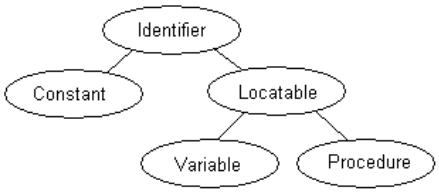
Objektový návrh překladače

Při použití objektově orientovaného jazyka, jakým bezesporu Java je, se tiše předpokládá, že způsob, jakým je daný model popisován, bude odpovídat základním myšlenkám objektově

orientovaného programování – data a algoritmy budou vhodným způsobem zapouzdřena do tříd, jež budou mezi sebou propojeny příslušnými vazbami. Tak je tomu i v našem případě.

Jako základní stavební kameny programu jsou využity následující třídy:

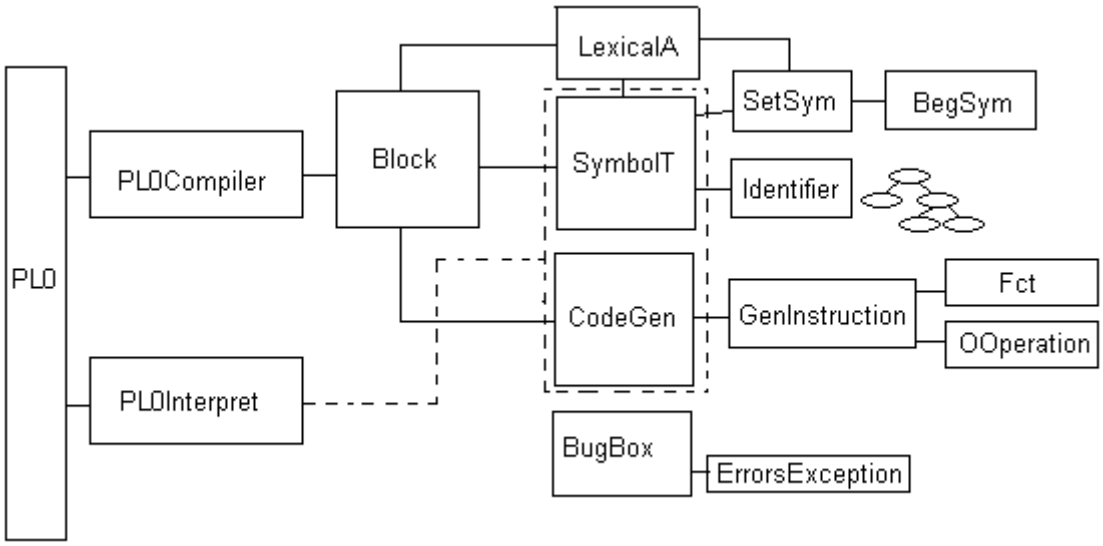
- `LexicalA` – lexikální analyzátor, zabezpečující čtení dat ze vstupního souboru pomocí vnitřní třídy `SourceReader`.
- `Block` – do jisté míry zapouzdřuje syntaktický analyzátor, vhnízdění bloků je realizováno vytvářením dalších instancí třídy Block uvnitř své třídy.
- `PL0Compiler` – je správcovskou třídou třídy Block – zavádí první blok a inicializuje lexikální analyzátor, tabulku symbolů a tabulku instrukcí.
- `CodeGen` – generovaný kód (tabulka instrukcí). Velikost generovaného kódu (resp velikost vstupního souboru) je staticky omezena (implementováno datovým typem pole). Elementární instrukci definuje třída `GenInstruction`, která využívá třídy `Fct` a `OOperation` pro určení kódu instrukce (`Fct`) a typ operace (`OOperation`). Obě dvě zmíněné třídy obsahují pouze veřejné konstanty, které pojmenovávají svým identifikátorem kód dané instrukce nebo typ operace (obdoba symbolických konstant známých z jazyka C).
- `SymbolT` – tabulka symbolů. Elementární symbol je zastoupen třídou `Identifier`, mezi jejíž potomky patří třídy `Constant`, `Locatable`, `Variable` a `Procedure`.



Obr.1 Hierarchie identifikátorů

- `SetSym` – s tabulkou symbolů je nepřímo spojena i třída `SetSym`, neboť umožňuje pracovat s množinami symbolů a kromě toho pojmenovává jednotlivé symboly svými třídními konstantami. Vedle toho existuje ještě třída `BegSys`, která nabízí některé předdefinované množiny, jež kompilátor používá.
- `BugBox` – správce chyb. Tato třída se stará o zpracování chyb kompilátoru; v případě, že počet chyb překročí hranici určenou jedním z atributů instance třídy `BugBox`, vygeneruje se výjimka `ErrorsException`, která se postará o ukončení překladu.
- `PL0Interpret`- interpret jazyka PL0; použije se tehdy, pokud překlad programu proběhne bez chyby. Interpret vytvoří v aktuálním adresáři soubory obsahující vygenerovaný kód (“`list.tab`”) a tabulku symbolů (“`symbol.tab`”).

Všechny uvedené třídy jsou obsaženy v balíku `pl0prog`. Vně tohoto balíku se nachází ještě třída `PL0`, která vytváří instanci třídy `PL0Compiler` a v případě bezchybného překladu předá instanci třídy `PL0Interpret` tabulku symbolů a vygenerovaný kód.



Obr 2. Zjednodušené schéma kompilátoru

Atributy a metody jednotlivých tříd

Generování kódu

Třída `Fct` – poskytuje jména instrukcí

Konstanty	
`LIT`	Uložení konstanty do zásobníku (`LIT 0,A`)
`OPR`	Provedení instrukce A (`OPR 0,A`)
`LOD`	Ulož proměnné L,A na vrchol zásobníku (`LOD L,A`)
`STO`	Zapiš proměnnou L z vrcholu zásobníku do paměti (`STO L,A`)
`CAL`	Volej proveduru A z úrovně L (`CAL L,A`)
`INT`	Zvyš obsah T-registru o hodnotu A (`INT 0,A`)
`JMP`	Proveď skok na adresu A (`JMP 0,A`)
`JPC`	Proveď podmíněný skok na adresu A (`JPC 0,A`)

Třída `OOperation` – poskytuje jména operací, uplatní se při `Fct.OPR`

Konstanty	
`NEG (0)`	Negace hodnoty na vrcholu zásobníku
`ADD (1)`	Součet dvou hodnot na vrcholu zásobníku
`SUB (2)`	Rozdíl dvou hodnot na vrcholu zásobníku
`MUL (3)`	Součin dvou hodnot na vrcholu zásobníku
`DI (4)`	Dělení dvou hodnot na vrcholu zásobníku
`MOD (5)`	Modulo dvou hodnot na vrcholu zásobníku
`ODD (6)`	Test lichosti hodnoty na vrcholu zásobníku
`EQ (7)`	Test rovnosti dvou hodnot na vrcholu zásobníku
`NE (8)`	Test nerovnosti dvou hodnot na vrcholu zásobníku
`LT (9)`	Menší
`GE (10)`	Větší nebo rovno
`GT (11)`	Větší
`LE (12)`	Menší nebo rovno

Třída `GenInstruction` – popisuje instrukci procesoru PL/0

`Proměnné`	
`FunctionCode`	Kód operace
`Level`	Úroveň vnoření

Operand	Kód operace/adresa
Metody	
GenInstruction	Vytvoření instrukce

Třída CodeGen – generátor kódu pro procesor PL/0

Metody	
getInstruction	Získání instrukce z tabulky
listcode	Vypis vybrane casti kodu na obrazovku
list	vypis generovaneho kodu do souboru
gen	Generovani instrukci do pole generovanych kodu.
getAddressCounter	Získání čítače adres.
getAdress	Získání hodnoty adresy od určité instrukce.
setAddressAt	Nastavení hodnoty adresy pro danou instrukci.

Lexikální analyzátor

Třída LexicalA – lexikální analýza

Metody	
GetCurrentSymbol	Vrátí naposledy načtený symbol
GetCurrentNumber	Vrátí naposledy načtené číslo
GetCurrentChar	Vrátí naposledy načtený znak
GetCurrentIdentifier	Vrátí naposledy načtený identifikátor
WriteError	Vypíše číslo chyby na odpovídající místo v řádku

Třída LexicalA.SourceReader – třída zabezpečující čtení ze souboru

Metody	
GetChar	Vrací další znak ze souboru

Tabulka symbolů

Třída SymbolT – tabulka symbolů

Metody	
Position	Vyhledání indexu do tabulky daného symbolu.
GetIdentifier	Vrací identifikátor z tabulky na základě indexu
GetTableIndex	Vrátí místo posledního uloženého identifikátoru
List	Výpis tabulky symbolů do souboru
EnterConstant	Vložení konstanty do tabulky symbolů
EnterVariable	Vložení proměnné do tabulky symbolů
EnterProcedure	Vložení procedury do tabulky symbolů
SetAddr	Nastavení adresy pro příslušný prvek v tabulce

Třída SetSym – množina symbolů

Konstanty					
NULL	Bez významu	LEQ	<=	THENSYM	then
IDENT	Identifikátor	GTR	>	WHILESYM	while
NUMBER	Číslo	GEQ	>=	DOSYM	do
PLUS	+	LPAREN	(	CALLSYM	call
MINUS	-	RPAREN	)	CONSTSYM	const
TIMES	*	COMMA	,	VARSYM	var
SLASH	/	SEMICOLON	;	PROCSYM	procedure
MODULO	%	PERIOD	.	SYMBOLS	
ODDSYM	Odd	BECOMES	:=		
EQL	==	BEGINSYM	begin		
NEQ	<>	ENDSYM	end		
LSS	<	IFSYM	if		

Metody	
`zero`	Odstraň všechny prvky z množiny
`get`	Zjisti, zda je příslušný prvek v množině
`set`	Vložení symbolu do množiny
`clear`	Vyjmutí symbolu z množiny
`or`	Sjednocení dvou množin
`and`	Průnik dvou množin

Třída `BegSym` – předdefinované množiny symbolů

Konstanty	
`StatBegSys`	Množina symbolů příkazů
`DeclBegSys`	Množina symbolů deklarační části
`FacBegSys`	Množina symbolů faktoru

Syntaktická analýza

Třída `Block` – syntaktický analyzátor

Metody	
`SetBugBox`	Nastavení správce chyb,
`GetBugBox`	Poskytnutí správce chyb.
`Run`	Spuštění překladu hlavního bloku.
`Test`	Testuje, zda je zadaný symbol v dané množině.
`Statement`	Příkaz
`Condition`	Podmínka
`Factor`	Faktor
`Expression`	Výraz
`Term`	Term
`Block`	Blok
`ConstDeclaration`	Deklarace konstanty
`VarDeclaration`	Deklarace proměnné

Třída `PL0Compiler` – syntaktický analyzátor

Metody	
`GetSymbolTable`	Poskytnutí tabulky symbolů
`GetCode`	Poskytnutí vygenerovaný kód
`GetBugBox`	Poskytnutí správce chyb
`Compile`	Inicializuj první blok

Interpret

Třída `PL0Interpret` – interpret jazyka PL0

Metody	
`run`	Spuštění interpretu

Spuštění programu

Překladač jazka PL0 se spouští dvojím způsobem:

1. `java PL0` – po spuštění se program dotáže na jméno souboru, který má být zkompilován,
2. `java PL0` *jméno_souboru* – program zahájí překlad souboru *jméno_souboru.*

Reference

[WIRTH] N.Wirth – Algorítmy a štruktury údajov – Bratislava 1990