P. Fischer

VVS - Bratislava

# PROGRAMOVANIE V PASCALE

Programovací jazyk PASCAL charakterizujeme ako jazyk vyššej úrovne. Vyššie programovacie jazyky prakticky vo všetkých sférach aplikácií vytlačili assemblery a strojový kód. Tento proces sa rozšíril aj do systémového a aplikačného programovania. Základné prednosti takýchto jazykov sú:

- rýchlejšie a menej namáhavé programovanie;
- väčšia zrozumiteľnosť, prehľadnosť a čitateľnosť programu;
- menšia pravdepodobnosť výskytu chýb;
- silná podpora pri ladení programu.

Za tieto prednosti sa zvyčajne platí zníženou účinnosťou a rýchlosťou programu. Kvalitný vyšší programovací jazyk žiadame:

- aby programy zostavené v tomto jazyku boli kompatibilné pre všetky jeho realizácie;
- aby sa dala dokázať korektnosť programov v ňom napísaných, eventuálne niektoré iné vlastnosti;
- aby sa v prípade, že máme jeho implementáciu, druhá implementácia na inom počítači dá realizovať s minimálnou námahou.

V súčasnej dobe je známych pomerne veľa programovacích jazykov vyššej úrovne, ktoré vo väčšej alebo

menšej miere spĺňajú uvedené vlastnosti. Z nich PASCAL rozhodne figuruje na jednej z popredných pozícií. Pri uvažovaní jednotlivých jazykov sa brala do úvahy miera, akou príslušný jazyk spĺňa tú - ktorú vlastnosť.

Programovací jazyk je definovaný svojou syntaxou a sémantikou. Syntax je popísaná nejakou gramatikou, ktorá obvykle patrí k triede bezkontextových gramatík. Sémantika je to, čo vlastne program robí. Syntax a sémantika však ešte nepopisujú kompilátor jazyka úplne, ale nechávajú ešte niečo ľubovoli realizátorom. Tento zvyšok budeme nazývať "fyzickou reprezentáciou" a bude predstavovať tretí komponent popisu jazyka. Jazyk je teda definovaný: syntaxou, sémantikou a fyzickou reprezentáciou.

V ďalšom si z tohto definičného aspektu priblížime programovací jazyk PASCAL, konkrétne jeho dátové a riadiace štruktúry, ako aj niektoré špeciálne črty.

Programovací jazyk PASCAL navrhol profesor Niklaus Wirth v roku 1970 a implementovali ho na ETH Zürich. Hlavným cieľom jazyka bol základ pre kurzy vyučovania programovacích jazykov a programovania. Od tej doby existuje vo svete viacero implementácií jazyka PASCAL na rôznych počítačoch a v rôznych modifikáciách (medzi inými i verzia PASCAL-B realizovaná vo Výskumnom výpočtovom stredisku v Bratislave na CDC 3300(MASTER). Zmienená verzia PASCALu-B bola navrhnutá a implementovaná so zreteľom pre potreby programovania veľkého programového systému (ISIS verzia 1-2, resp. SOFIS).

Programovací jazyk PASCAL bol vyvinutý na báze jazyka ALGOL60. Vzhľadom k ALGOLu 60 PASCAL charakterizuje značná rozmanitosť dátových štruktúr, čím sa podstatne zvýšil rozsah aplikovateľnosti tohoto jazyka. V zmysle

zamýšľaného použitia jazyka (jednak ako nástroja na vyučovanie a potom ako prostriedku pre realizovanie veľkých programových systémov) bol hlavný dôraz kladený na primerane malý počet fundamentálnych konceptov - jednoduchá a systematická štruktúra jazyka a efektívna implementovateľnosť jazyka. Pevodný kompilátor jazyka PASCAL, ktorý bol vytvorený na spomínanej škole ETH v Zürichu, bol skonštruovaný pre počítač CDC 6000. Kompilátor bol jednoprechodový a na jeho implementáciu sa použil samotný jazyk PASCAL.

Ako už bolo spomenuté vyššie, PASCAL vychádzal z Algolu 60. Je zrejmé, že pri vzniku nového jazyka sa nesleduje iba púhy cieľ "vytvoriť nový jazyk", ale že je snahou, pokiaľ možno, čo najviac uplatniť poznatky a skúsenosti rezultujúce z existujúcich jazykov. Teda: existujúce jazyky predstavujú bázu pre návrh a realizovanie nového programovacieho jazyka. V tomto zmysle Algol 60 predstavoval základ pre vytvorenie jazyka PASCAL. Algol 60, na rozdiel od ostatných jazykov, najväčšmi spĺňal požiadavky a ciele kladené na moderný programovací jazyk, tj. systematická štruktúra, flexibilita programov a dátových štruktúr, efektívna a účinná implementovateľnosť. Z tohto aspektu - princípy štruktúrovania prevzal PASCAL z jazyka Algol 60, rovnako ako i formu výrazov (syntax a sémantiku). Hlavné rozšírenie, či zmena PASCAL vs. ALGOL 60 predstavujú dátové štruktúry. V nich tkvela príčina relatívne úzkeho poľa aplikovateľnosti Algolovských produktov. Zavedenie štruktúry záznam (RECORD), či štruktúry súbor (FILE) umožnilo adaptovať PASCAL na riešenia rozných komerčných problémov ako i výhodne ho používať ako vyučovací nástroj v rámci kurzov programovania. V roku 1965 vzniká jazyk ALGOL ako následovník Algolu 60 a ako priamy predchodca jazyka PASCAL. Tento jazyk obsahoval niektoré rozšírenia v oblasti definícií dát a možno ho pokladať

DYNAMICKÉ - až počas výpočtu sa určí, ktorá premenná
PREMENNÉ prislúcha danému menu. V priebehu výpočtu sa
môže tým istým menom odvolávať na viacej
premenných toho istého typu. Dynamické pre-
menné sa realizujú pomocou premenných typu
smerník. Premenná, ktorá v danom okamihu
prislúcha danému menu vznikne vykonaním štan-
dardnej procedúry ALLOC. Aj pri deštrukcii
rekurzívnych funkcií pomocou DYNAMIC
PREMENNÝCH sa premenné vyhradzujú dynamicky.

Základným typom je skalár - usporiadaná množina hod-
nôt. Definícia skaláru určuje identifikátor pre každú
hodnotu usporiadanej množiny. V jazyku existujú aj štan-
dardné predefinované skalárne typy:

BOOLEAN - má dve hodnoty: falošnú a pravdivú. Zapíšeme ich
FALSE a TRUE.

INTEGER - množinou hodnôt sú všetky celé čísla, dostupné
pre daný počítač.

Skalár môže byť definovaný aj ako interval - určitá pod-
množina už definovaného ľubovoľného skalára - urče-
ním najmenšej a najväčšej hodnoty.

Ostatnými základnými typmi sú REAL, CHAR a ALFA:

REAL - množinou hodnôt sú všetky reálne čísla dostupné
pre daný počítač.

CHAR - množinou hodnôt sú všetky znaky, dostupné pre da-
ný počítač. Používame ich uvedením daného znaku
v úvodzovkách.

ALFA - preddefinovaný rozsahový s dĺžkou 8 znakov.
Používame ho uvedením ľubovoľných znakov v úvodzovkách.

Ďalším typom je typ smerník. Množinou hodnôt sú im-
plicitné názvy premenných, ktoré sú rovnakého typu a tvoria
zložky danej premennej typu RECORD. Tieto implicitné názvy

nie sú bezprostredne dostupné užívateľovi. Implicitný názov vytvorí procedúra ALLOC a uloží ho do označenej premennej typu smerník. V priebehu výpočtu môžeme pristupovať pomocou hodnoty jednej premennej typu smerník k rôznym zložkám danej premennej typu TRIEDA, ale v každý časový okamih výpočtu iba k jednej.

Pascal umožňuje definovať nové typy nazývané štrukturované, pomocou už existujúcich (t.j. štandardných alebo už definovaných). Štruktúrovaný typ sa zavádza označením spôsobom štruktúrovania a typu zložiek, z ktorých sa skladá. Rôzne spôsoby štruktúrovania sa odlišujú prístupovým mechanizmom, ktorý umožňuje prístup k hodnotám zložiek premennej daného štruktúrovaného typu.

Máme nasledovné spôsoby štruktúrovania:

POLE - všetky zložky musia byť rovnakého typu. Zložky sú prístupné pomocou selektorov, ktorými sú vypočítateľné indexy. Tieto môžu byť typu skalár a udávajú sa v každej konkrétnej definícii typu POLE. Pre danú hodnotu indexu selektor poskytne hodnotu príslušnej zložky. Preto môžeme o každej premennej typu POLE uvažovať ako o zobrazení hodnôt indexov na hodnoty zložiek. Čas potrebný na získanie hodnoty zložky nezávisí od hodnoty indexu.

ZÁZNAM - tento typ umožňuje deklarovať premenné, ktorých štruktúra sa môže (ale nemusí) meniť počas výpočtu. To znamená, že sa môže meniť počet a typ zložiek. Premenná môže obsahovať konečný počet variantov. Informáciu, o ktorý variant štruktúry premennej ide v danej chvíli výpočtu, obsahuje jej zložka, nazývaná značková položka. Túto informáciu si však musí užívateľ nastavovať sám pomocou priraďovacie-

ho príkazu. Zložky jedného variantu nemusia byť rovnakého typu. Aby bol typ vybranej zložky jasný z programovaného textu (nie až počas vykonávania programu) nie je hodnota zložky prístupná pomocou vypočítateľných hodnôt, ale pomocou identifikátora, ktorý je uvedený priamo v definícii daného typu ZÁZNAM. Čas potrebný na získanie hodnoty zložky nezávisí od identifikátora.

TRIEDA - definuje triedu zložiek rovnakého typu. Počet zložiek sa môže meniť počas behu programu. Nová zložka sa pridáva pomocou štandardnej procedúry ALLOC, ktorej parametrom je identifikátor premennej typu smerník. Pomocou tejto premennej je prístupná zložka, ktorú sme pridali. Pre každú premennú typu TRIEDA je potrebné deklarovať premenné typu smerník, pomocou ktorých sú prístupné jedine jej zložky. Zložky inej premennej typu TRIEDA nie sú prístupné pomocou týchto premenných. Premenná typu smerník môže mať hodnotu NIL. Vtedy pomocou nej nie je prístupná žiadna zložka. Hodnotu NIL môže mať každá premenná typu smerník, bez ohľadu na to, ku ktorej premennej typu TRIEDA patrí.

MNOŽINA - Typ množina určuje rozsah hodnôt, ktorým je množina všetkých podmnožín nejakého skaláru alebo intervalu. Hodnotou premennej typu množina môže byť v danom čase výpočtu jedná, ale ktorákoľvek, z podmnožín zvoleného rozsahu. Tento typ umožňuje bezprostredne narábať s bitmi.

Operácie s údajmi sa vykonávajú podľa príkazov. Základným druhom príkazu je priraďovací príkaz. Tento urču-

je, ktorej premennej alebo zložka premennej sa priraďuje hodnota. Nová hodnota premennej alebo jej zložky sa získa hodnotením výrazu, ktorý je vpravo od znaku priradenie.

Výrazy vytvárame z premenných, konštánt, množín a funkcií pomocou nasledovných operátorov:

1. Aritmetické operátory: sčítanie, odčítanie, násobenie, delenie, zvyšok po delení, inverzia znamienka. Operandy i výsledok operácií sú typu INTEGER alebo REAL.
2. Logické operátory: logický súčet (alebo), súčin (a) a zápor. Operandy i výsledok operácií je typu BOOLEAN.
3. Množinové operátory: zjednotenie, prienik, rozdiel. Operandy i výsledok sú typu MNOŽINA.
4. Relačné operátory: rovnosť, nerovnosť, usporiadanie prvkov skaláru, príslušnosť k množine. Výsledok operácie s ľubovoľným z relačných operátorov je typu BOOLEAN. Ľubovoľné dva operandy sa možu porovnať len vtedy, ak sú rovnakého typu.

Priraďovací príkaz patrí do skupiny jednoduchých príkazov, pretože neobsahuje v sebe žiadny príkaz. Jednoduchým príkazom je aj vyvolánie procedúry a EXIT príkaz (ukončenie štruktúrovaného príkazu).

Podobne, ako možeme z jednoduchších typov vytvárať zložitejšie, možeme v Pascale B vytvárať zložitejšie príkazy z jednoduchších. Príkaz, ktorý obsahuje v sebe iné príkazy, sa nazýva štruktúrovaný.

Štruktúrovaný príkaz umožňuje určit postupne, výberové alebo opakujúce sa vykonávanie príkazov, ktoré sú jeho zložkami:

- postupné vykonávanie príkazov je určené zloženým príkazom,
- výberové vykonávanie umožňujú príkazy IF a SELECT podľa hodnoty logických výrazov alebo príkaz CASE podľa hodnoty výrazu typu skalár,
- opakujúce sa vykonávanie umožňuje príkaz FOR, ak je počet opakovaní dopredu známy, alebo REPEAT a WHILE, kde

sa opakovanie vykoná vo podľa hodnoty logického výrazu.

Príkaz môže byť označený menom (identifikátorom). Toto meno sa môže použiť jedine v príkaze ukončenia (príkaz EXIT). Príkaz ukončenia môže použiť meno jedine toho štruktúrovaného príkazu, ktorého je časťou.

V Pascale môžeme používať procedúry. Je to časť programu popísaná pomocou deklarácie procedúry, ktorá sa dá aktivovať z niektorého miesta programu. Na aktivácia slúži príkaz vyvolania procedúry obsahujúci meno žiadanej procedúry.

Procedúra je samostatnou jednotkou programu v tom zmysle, že môže obsahovať deklarácie premenných, ďalších procedúr a definície typov. Takto deklarované premenné, procedúry a typy môžeme použiť len v rámci procedúry samotnej. Preto ich nazývame lokálnymi vzhľadom k danej procedúre. Ich identifikátory majú lokálnu účinnosť jedine v programovom texte, ktorý tvorí deklarácia procedúry a ktorý sa nazýva rozsahom týchto identifikátorov. Preto že procedúry môžu byť deklarované ako lokálne pre iné procedúry, rozsahy môžu byť zaniezdené. Premenné, procedúry a typy, ktoré nie sú lokálne a sú deklarované v hlavnom programe, sa nazývajú globálne.

Procedúra má pevný, konečný počet parametrov, z ktorých každý je označený v deklarácii procedúry identifikátorom, nazývaným formálny parameter. Pri aktivovaní procedúry musíme určiť, akú hodnotu nadobúda každý z parametrov. Na túto hodnotu, ktorú volíme skutočný parameter, sa môžeme v procedúre odvolávať pomocou príslušného formálneho parametra.

Existujú tri druhy skutočných parametrov podľa použitia:

- Skutočný parameter je výraz, ktorý sa vyhodnotí pred započatím práce procedúry. Hodnota výrazu sa uloží v premennej, ktorej menom je príslušný formálny parameter.

- Skutočný parameter je premenná v procedúre sa použije na mieste výskytu príslušného formálneho parametra. Možné indexy sa vyhodnotia pred vykonaním procedúry.
- Skutočný parameter je identifikátor procedúry alebo funkcie. Všade tam, kde je v procedúre formálny parameter, vyvolá sa procedúra alebo funkcia, ktorej identifikátor slúži ako príslušný skutočný parameter.

Funkcie sú deklarované analogicky ako procedúry. Rozdiel je len v tom, že funkcia vytvára hodnotu typu skalár alebo smerník. Typ musí byť určený v deklarácii funkcie. K hodnote pristupujeme menom funkcie.

Ak funkcia vytvára skalár, môže byť častou výnimkou.

V Pascale je možné určiť, či je funkcia a procedúra rekurzívna (označenie DYNAMIC FUNCTION) alebo nerekurzívna (označenie FUNCTION). Nerekurzívne funkcie majú podstatne kratší čas vyvolania i vyhodnotenia než rekurzívne. Lokálne premenné nerekurzívnej funkcie a procedúry si zachovávajú hodnoty, ktoré nadobudli pri poslednom aktivovaní. Funkcia a procedúra môžu používať lokálne premenné lexikálne nadradenej rekurzívnej funkcie. Ak však lexikálne nadradená funkcia alebo procedúra je nerekurzívna, potom jej lokálne premenné sú prístupné už vo funkciách a procedúrach, ktoré sú do nej lexikálne vnorené.

Pascal je strojovo nezávislý jazyk. Vytvorenie programu vykonateľného pre daný počítač sa uskutočňuje pomocou kompilátora. Vykonateľný program sa môže vytvoriť jedným behom kompilátora, ale niekedy behmi, zo segmentov. Segment je nezávisle kompilovateľná jednotka. Odovzdávanie informácie medzi segmentami sa uskutočňuje pomocou parametrov, alebo pomocou premenných, ktoré sú na to špeciálne deklarované (pomocou [illegible]).

V následujúcom texte si detailnejšie priblížime jednotlivé elementy jazyka.

## IDENTIFIKÁTORY

Identifikátory slúžia ako mená na označenie konštánt, typov, premenných, procedúr a funkcií. Identifikátor môže označovať jedinú konštantu, premennú, typ, alebo funkciu v rámci rozsahu platnosti t.j. v rámci funkcie alebo procedúry, kde je deklarovaný. Napríklad nemôžeme tým istým identifikátorom označiť súčasne dve rozne konštanty, alebo funkciu a premennú. Syntakticky je identifikátor postupnosť písmen a číslic, začínajúca sa písmenom. Ako identifiktáry sa nesmú používať špeciálne symboly. V programu je možné napísať: identifikátor každej konečnej dĺžky, ale kompilátor berie do úvahy iba prvých osem znakov.

## ČÍSLA

Čísla sú konštanty. Na označenie celých čísel používame desiatkový a osmičkový zápis. Ich typ je INTEGER. Na označenie reálnych čísel používame len desiatkový zápis a ich typ je REAL.

## REŤAZEC

Reťazcom sa nazýva postupnosť znakov uzavretá v uvodzovkách. Hodnotami premenných môžu byť len reťazce dĺžky 1 alebo 8.

## KONŠTANTY

Konštantami sú čísla so znamienkom alebo bez, reťazce, identifikátory, NIL (môžu byť hodnotou premennej typu smerník) a konštantné výrazy.

DEFINÍCIA KONŠTANTY

Definícia konštanty zavádza identifikátor ako synonymum konštanty a zabezpečuje jej rovnoprávne uplatnenie miesto konštanty.

DEFINÍCIA TYPU

Typ určuje množinu hodnot, ktoré sú prípustné pre premenné toho typu a spôsob použitia týchto hodnot. Definícia priraďuje identifikátor ako meno typu. V prípade štruktúrovaného typu definícia určuje aj spôsob štruktúrovania t.j. druh, počet a poradie zložiek.

TYP SKALÁR

Typ skalár je základným typom; definuje usporiadanú konečnú množinu hodnot vymenovaním identifikátorov, ktoré označujú tieto hodnoty. Usporiadanie je určené poradím identifikátorov v definícii skaláru.

TYP INTERVAL

Tento typ môže byť definovaný ako interval iného skalárneho typu pomocou vyznačenia najnižšej a najvyššej hodnoty intervalu. Prvá konštanta určuje spodnú hranicu a nesmie byť väčšia než horná hranica.

TYP MNOŽINA

Typ množina určuje okruh hodnot ako množina všetkých podmnožín nejakého skaláru alebo intervalu, ktorý sa volá základ typu. Hodnotou premennej typu množina môže byť ktorákoľvek z podmnožín základu typu (t.j. zvoleného skaláru).

⟨typ množina⟩ ::=POWERSET ⟨identifikátor typu⟩|POWERSET ⟨typ interval⟩

Operátory, ktoré sa dajú použiť na všetky typy množín sú

AND prienik množín
OR zjednotenie množín
\- rozdiel množín
IN príslušnosť do množiny

Pretože premenná typu množina je realizovaná ako n-bitové slovo, nemôže základ typu obsahovať viacej ako n konštánt. Každá konštanta zodpovedá jednému bitu slova. Najvyšší bit zodpovedá najmenšej konštante (najľavejšej). Ak daná konštanta patrí do podmnožiny jej bit je zapínený; keď nepatrí, nie je zapínený. Takýmto spôsobom nám premenná typu množina umožňuje narábať s bitmi.

TYP POLE

Typ pole je štruktúra, ktorá pozostáva z pevného počtu zložiek rovnakého typu. Prvky poľa sú označené indexami. Definícia typu pole určuje typ prvkov poľa a rozsah hodnôt indexov.

⟨typ pole⟩::=ARRAY[⟨typ indexu⟩{ ,⟨typ indexu⟩}*] OF
⟨typ zložky⟩

Identifikátor typu môže označovať jedine typ interval alebo skalár. Pole sa volá n - rozmerným, ak má definovaných n indexov. Ku každému prvku poľa sa potom pristupuje pomocou n indexov.

TYP ZÁZNAM

Typ záznam je štruktúra, ktorá pozostáva z prvkov, nazývaných položky. Položky môžu byť rôzneho typu. Definícia typu záznam určuje pre každú položku jej typ a identifikátor. Identifikátor položky sa používa pri jej použití pomocou určovateľa položky.

Typ záznam obsahuje pevnú časť alebo premennú časť, prípadne obidve. Pevná časť pozostáva z položiek, ktoré obsahuje každá premenná daného typu záznam počas celého výpočtu. Premenná časť sa začína špeciálnym symbolom CASE a umožňuje zaviesť niekoľko variantov v rámci daného typu záznam. Varianty sa líšia rôznym počtom i typom svojich položiek. O ktorý variant záznamu v danom okamihu ide, určuje značková položka, ktorá je tiež súčasťou záznamu. Hodnotu značkovej položky si užívateľ nastavuje sám! Každý variant je označený pomocou návestia variantu, čo musí byť konštanta rovnakého typu ako značková položka. Značkovej položke sú ako hodnoty priraďované návestia variantov.

Ten istý identifikátor sa nesmie vyskytovať viackrát v jednej definícii typu záznam!

⟨typ záznam⟩ ::= RECORD ⟨zoznam položiek⟩ END

TYP TRIEDA

Typ trieda je štruktúra, ktorá pozostáva z prvkov rovnakého typu. Počet prvkov je počas behu programu premenný. Prvky sa môžu pridávať v čase behu programu pomocou štandardnej procedúry ALLOC a uvoľňovať v čase behu programu pomocou štandardnej procedúry FREE! Trieda sa vyprázdni pri každom vstupe do procedúry, v ktorej vnútri sme ju deklarovali.

Maximálny počet prvkov,ktoré môžeme pridať, určíme hodnotou nasledujúcou za špeciálnym symbolom CLASS. Typ uvedený za OF je typom pridávaného prvku.

⟨typ trieda⟩ ::=CLASS⟨maximum⟩ OF ⟨typ⟩

## TYP SMERNÍK

Množinu hodnot tvoria implicitné mená prvkov konkrétnej premennej typu trieda. Implicitné meno ale je prístupné užívateľovi. Užívateľ má prístup iba k prvku, ktorý dané impl. meno označuje.

Každý typ smerník je definovaný pre konkrétnu premennú typu trieda. Smerníkom nazývame premennú typu smerník. O smerníku z danej premennej typu trieda hovoríme, že sú viazané. Pomocou smerníku máme prístup jedine k prvkom tej premennej typu trieda, ku ktorej je viazaný. Každá premenná typu trieda má svoj typ smerník a svoje smerníky.

Hodnota smerníka určuje, ktorý prvok je pomocou neho prístupný. Hodnotu smerníku môžeme zmeniť pomocou priraďovacieho príkazu, alebo aktivovaním procedúry ALLOC, ktorá má ako parameter uvedený identifikátor daného smerníku. Procedúra ALLOC pridá nový prvok premennej typu trieda a smerníku priradí hodnotu, pomocou ktorej je prístupný novevytvorený prvok. Ak meníme hodnotu smerníka pomocou priraďovacieho príkazu, potom smerník referencuje prvok príslúchajúci priradenej hodnote. Stará hodnota smerníku stráca sa.

## DEKLARÁCIA A OZNAČENIE PREMENNÝCH

Pomocou deklarácie sa premennej priradí identifikátor, ktorý slúži ako jej meno, a typ, ktorý vymedzuje jej možné hodnoty a spôsob ich použitia.

Deklarácie premenných pozostávajú zo zoznamu identifikátorov, ktoré označujú premenné, a typu, ktorý sa daným premenným priraďuje.

## VÝRAZY

Výrazy sú konštrukcie vytvorené z operátorov a operandov (premenné; konštanty, množiny a funkcie). Výrazy

popisujú pravidlá výpočtu nových hodnôt premenných. Typom výrazu nazývame typ výsledku posledného aplikovaného operátora.

Operátory delíme do štyroch precedenčných tried. V najvyššej triede je operátor NOT. Potom nasleduje trieda násobiac ich operátorov, ďalej trieda sčítacích operátorov. Najnižšiu precedenčnú triedu tvoria relačné operátory.

Ak poradie aplikovania operátorov nie je vyznačené zátvorkami, potom sa pridržiavame nasledujúcich pravidiel:

- v časti výrazu, ktorá neobsahuje zátvorky, postupujeme zľava doprava od prvého operátora,
- operátor sa aplikuje, ak za ním nasleduje operátor s nižšou alebo rovnakou precedenciou a začneme opäť od prvého operátora,
- operátor sa neaplikuje, ak za ním nasleduje operátor s vyššou precedenciou.

Skúmame aplikovateľnosť tohoto ďalšieho operátora. Všetky výrazy, ktoré sú prvkami množiny, musia byť rovnakého typu skalár, ktorý je základom typu danej množiny. [] označuje prázdnu množinu.

OPERÁTORY

1. Operátor NOT sa aplikuje na booleovskú premennú a označuje negáciu. Ak má booleovská premenná B hodnotu TRUE (FALSE), potom výraz NOT B má hodnotu FALSE (TRUE).

2. Násobiace operátory

⟨násobiaci operátor⟩ :: = * | / | DIV | MOD | AND

3. Sčítacie operátory

⟨sčítací operátor⟩ :: + | - | OR

4. Relačné operátory

⟨relačný operátor⟩ ::= LT | LE | GE | GT | EQ | NE | IN

PRÍKAZY

Príkazy popisujú algoritmickú činnosť, ktorá sa vykonáva počas behu programu (na rozdiel od definícií a deklarácií).

⟨príkaz⟩::= ⟨jednoduchý príkaz⟩|⟨štrukturovaný príkaz⟩|
|(⟨návestie⟩)⟨štrukturovaný príkaz⟩

JEDNODUCHÝ PRÍKAZ

Jednoduchý príkaz je taký, ktorý neobsahuje iný príkaz ako svoju časť (na rozdiel od štruktúrovaného).

⟨jednoduchý príkaz⟩ ::= ⟨priraďovací príkaz⟩|
⟨príkaz procedúry⟩|⟨príkaz ukončenia⟩

PRIRAĎOVACÍ PRÍKAZ

Pomocou priraďovacieho príkazu premenná na ľavej strane operátora priradenia nadobudne hodnotu, ktorá vznikne vyhodnotením výrazu na pravej strane od operátora priradenia.

PRÍKAZ VYVOLANIA PROCEDÚRY

Príkaz vyvolania procedúry slúži na aktivovanie procedúry. Príkaz vyvolania procedúry môže - okrem identifikátora procedúry - obsahovať zoznam skutočných parametrov, ktoré sú dosadené na zodpovedajúce miesta formálnych parametrov, definovaných v deklarácii procedúry. Rozoznávame štyri druhy parametrov: premenné, konštanty, procedúry a funkcie. Po ukončení procedúry sa vykoná príkaz nasledujúci za príkazom vyvolania procedúry.

PRÍKAZ UKONČENIA

⟨príkaz ukončenia⟩::= EXIT⟨návestie⟩

Príkaz ukončenia s daným návestím možno použiť jedine vnútri štruktúrovaného príkazu, ktorý je označený týmto návestím. Beh programu pokračuje vykonaním príkazu, ktorý nasleduje za príkazom označeným daným návestím.

## ŠTRUKTÚROVANÉ PRÍKAZY

Štruktúrované príkazy sú konštrukcie zostavené z iných príkazov, ktoré môžu byť vykonávané postupne (zložený príkaz), podľa rozhodovacej podmienky (podmienený príkaz) alebo niekoľkokrát v cykle (príkaz cyklu).

## ZLOŽENÝ PRÍKAZ

Zložený príkaz určuje, že jeho zložkové príkazy sa majú vykonávať v poradí, v akom sú napísané (prvý za BEGIN, posledný pred END).

## PODMIENENÝ PRÍKAZ

Podmienené príkazy sa používajú na výberové vykonávanie ich zložkových príkazov. Výber príkazu, ktorý sa má vykonať, sa uskutočňuje podľa hodnoty logických výrazov (príkaz IF a SELECT) alebo podľa hodnoty výrazu typu skalár (príkaz CASE).

## PRÍKAZ IF

⟨IF príkaz⟩ ::= IF ⟨výraz⟩ THEN ⟨príkaz⟩
IF ⟨výraz⟩ THEN ⟨príkaz⟩ ELSE ⟨príkaz⟩

Výraz medzi IF a THEN musí byť typu BOOLEAN.

Uskutočnenie príkazu IF prebieha podľa nasledujúceho pravidla:

- ak výraz za IF má hodnotu TRUE, potom sa vykoná príkaz, ktorý nasleduje za THEN. Tým sa vykonanie daného príkazu IF končí.

- ak výraz za IF má hodnotu FALSE, potom pre prvý variant príkazu IF sa jeho vykonanie končí, pre druhý variant príkazu IF sa vykoná príkaz uvedený za ELSE. Tým sa vykonanie daného príkazu IF končí.

PRÍKAZ SELECT

⟨príkaz SELECT⟩ ::= SELECT⟨výberový element⟩{;⟨výberový element⟩}*END ⟨výberový element⟩ ::= ⟨výraz⟩: ⟨príkaz⟩

Všetky výrazy v príkaze SELECT musia byť typu BOOLEAN. Vykonanie SELECT príkazu prebieha podľa nasledujúceho pravidla:

uskutoční sa ten príkaz z postupnosti výberových prvkov, ktorého výraz nadobudne ako prvý z postupnosti hodnotu TRUE. Ak všetky výrazy majú hodnotu FALSE, nevykoná sa príkaz zo žiadneho výberového elementu. (V tomto prípade je príkaz SELECT ekvivalentný prázdnemu príkazu). Prvý výberový element je za SELECT a posledný pred END.

PRÍKAZ CASE

⟨príkaz CASE⟩ ::= CASE⟨výraz⟩OF⟨element prípadu⟩{;⟨element prípadu⟩}*END | CASE ⟨výraz⟩ OF

⟨element prípadu⟩{;⟨element prípadu⟩}*;
OTHERWISE: ⟨príkaz⟩ END

⟨element prípadu⟩ ::={⟨návestie prípadu⟩ :}⊕⟨príkaz⟩
{⟨návestie prípadu⟩ :}⊕

⟨návestie prípadu⟩::= ⟨identifikátor⟩|⟨celé číslo⟩

Výraz uvedený medzi CASE a OF a všetky návestia prípadu, uvedené v danom príkaze CASE, musia byť rovnakého

typu skalár. Za štandardných skalárnych typov je dovolený len INTEGER a BOOLEAN. To isté návestie prípadu nesmie byť uvedené v dvoch rozných elementoch prípadu. Vykonanie príkazu CASE prebieha podľa následovného pravidla:

vyhodnotí sa výraz uvedený medzi CASE a OF. Ak jeho hodnota nie je totožná so žiadnym návestím prípadu, uvedeným v niektorom elemente prípadu, potom

- pre prvý variant príkazu je daný príkaz CASE ekvivalentný prázdnemu príkazu
- pre druhý variant príkazu sa vykoná príkaz uvedený v danom príkaze CASE za OTHERWISE:

Ináč sa vykoná príkaz toho elementu príkazu, ktorého návestím je vypočítaná hodnota.

PRÍKAZY CYKLU

Príkaz cyklu umožňuje niekoľkonásobné vykonanie určitej postupnosti príkazov. Ak je možné počet opakovaní nastaviť pred počatím prevádzania postupnosti príkazov (tj. vykonanie postupnosti príkazov nemože zmeniť počet opakovaní v cykle), použijeme príkaz FOR. Ak sa o ďalšom opakovaní postupnosti príkazov rozhoduje na základe predchádzajúceho behu danej postupnosti príkazov, potom použijeme príkaz WHILE alebo REPEAT. V tomto prípade ukončenie opakovania určuje hodnota logického výrazu.

PRÍKAZ WHILE

⟨príkaz WHILE⟩::= WHILE ⟨výraz⟩ DO ⟨príkaz⟩

Výraz v príkaze WHILE musí byť typu BOOLEAN. Vykonanie príkazu WHILE prebieha podľa nasledujúceho pravidla:

vyhodnotí sa logický výraz medzi WHILE a DO. Ak má hodnotu TRUE, vykoná sa príkaz uvedený za DO a opäť sa vykoná samotný príkaz WHILE. Ak má hodnotu FALSE, príkaz uvedený za DO sa nevykoná a vykonanie celého

príkazu WHILE sa končí.

PRÍKAZ REPEAT

⟨príkaz REPEAT⟩ ::= REPEAT ⟨príkaz⟩{; ⟨príkaz⟩}* UNTIL ⟨výraz⟩

Výraz v príkaze RPEAT musí byť typu BOOLEAN. Vykonanie príkazu REPEAT prebieha podľa nasledujúceho pravidla:

vykonajú sa príkazy uvedené medzi REPEAT a UNTIL. Potom sa vyhodnotí výraz za UNTIL. Ak má hodnotu FALSE, opäť sa vykoná zamotný príkaz REPEAT. Ak má hodnotu TRUE, vykonanie zamotného príkazu REPEAT sa končí.

PRÍKAZ FOR

⟨FOR príkaz⟩ ::= FOR ⟨riadiaca premanná⟩ =⟨zoznam ohraničení⟩
DO ⟨príkaz⟩

⟨zoznam ohraničení⟩ ::=⟨počiatočná hodnota⟩ TO ⟨konečná⟩ hodnota⟩|⟨počiatočná hodnota⟩DOWNTO ⟨konečná hodnota⟩

⟨riadiaca premenná⟩ ::=⟨identifikátor⟩

⟨počiatočná hodnota⟩ ::=⟨výraz⟩

⟨konečná hodnota⟩ ::= ⟨výraz⟩