Obsah

1 Programovací jazyk PASCAL – Turbo **5**

1.1 Základní prostředky pro zápis programu . . . 5

1.1.1 Lexikální symboly jazyka . . . 6

1.2 Struktura programu . . . 7

1.3 Část definicí a deklarací . . . 8

1.3.1 Deklarace návěští . . . 8

1.3.2 Definice konstant . . . 8

1.3.3 Definice typů a deklarace proměnných . . . 9

1.3.4 Deklarace podprogramů . . . 11

1.4 Datové typy . . . 11

1.4.1 Jednoduché datové typy . . . 11

1.4.1.1 Celočíselné typy . . . 11

1.4.1.2 Typy reálných číselných hodnot . . . 12

1.4.1.3 Typ char (znaková hodnota) . . . 12

1.4.1.4 Typ Boolean (logická hodnota) . . . 13

1.4.1.5 Typ interval . . . 16

1.4.1.6 Typ definovaný výčtem . . . 16

1.4.2 Strukturované datové typy . . . 17

1.4.2.1 Typ řetězec . . . 17

1.4.2.2 Typ soubor . . . 18

1.4.2.3 Typ záznam . . . 18

1.4.2.4 Typ množina . . . 19

1.4.2.5 Typ pole . . . 20

1.4.2.6 Typ ukazatel . . . 21

1.4.2.7 Typ podprogram . . . 21

1.5 Část příkazová . . . 21

1.5.1 Jednoduché příkazy . . . 22

1.5.1.1 Příkaz přiřazovací . . . 22

1.5.1.2 Příkaz procedury . . . 24

1.5.1.3 Příkaz prázdný . . . 25

1.5.1.4 Příkaz skoku . . . 25

1.5.2 Strukturované příkazy . . . 26

1.5.2.1 Příkaz složený . . . 26

1.5.2.2 Příkaz podmíněný . . . 26

1.5.2.3 Příkaz selektivní . . . 26

1.5.2.4 Příkaz cyklu . . . 27

1.5.2.4.1 Příkaz cyklu repeat — until . . . 27

1.5.2.4.2 Příkaz cyklu while - do . . . 28

1.5.2.4.3 Příkaz cyklu for - to - do resp. for - down - to . . . 28

1.5.2.5 Příkaz with . . . 29

Kapitola 1

Programovací jazyk PASCAL – Turbo

Turbo Pascal je programový systém, který umožňuje uživateli na osobním počítači efektivně pracovat s Pascalem a v Pascalu. I když i na osobních počítačích (dále jen PC) se můžeme setkat s různými implementacemi Pascalu (např. MicroSoft Pascal), je dnes v jejich prostředí nejrozšířenější právě **Turbo Pascal**.

S Turbo Pascalem se střetáváme již od počátku éry PC začátkem 80. let, kdy jej americká firma *Borland International* distribuuje k 8 bitovým PC s operačním systémem CP/M i MS–DOS. Verze Turbo Pascal 3.0 byla první verzí implementovanou na 16 bitových PC pod operačním systémem MS DOSem, verze Turbo Pascal 4.0 již dává možnosti modulárního programování (zavádí programové jednotky), verze Turbo Pascal 5.0 poskytuje výkonný ladící program a stává se nejúspěšnějším programovým produktem roku 1988 na světě. Verze Turbo Pascal 5.5 přináší možnosti objektově orientovaného programování a v našich zemích je značně rozšířena. Verze Turbo Pascal 6.0 již obsahuje knihovny pro podporu objektového programování, umožňuje editaci více souborů najednou, přináší i efektivní prostředí pro práci s myší aj. Turbo Pascal verze 7.0 přináší uživateli další výhody.

Obecně se dá říci, že každá nová verze je oproti předchozí rychlejší, poskytuje komfortnější prostředí a zavádí nové standardní podprogramy. Programy sestavené v předchozí verzi jsou zpravidla použitelné i pod novou verzí. Jistou nevýhodou je, že každá nová verze Turbo Pascalu je vzdálenější **standardnímu Pascalu**[1] (je jeho stále širší nadstavbou), což může znepříjemnit jak přenositelnost (portabilitu) pascalovských programů z PC na jiné výpočetní systémy, tak i slučitelnost (kompatibilitu) programů či jejich částí.

1.1 Základní prostředky pro zápis programu

Při výuce živého jazyka jsme nuceni se seznámit nejprve s jeho abecedou, pak se zpravidla naučíme několik slovíček (lexikálních jednotek) a po zvládnutí základů gramatiky alespoň v oblasti syntaxe (způsobu zápisu) jsme schopni základní komunikace formou mluvenou či písemnou. Musíme si uvědomit, že určitá syntaktická konstrukce může mít více sémantických (obsahových) významů, přičemž syntaktických konstrukcí můžeme vytvořit prakticky neomezený počet. Z uvedeného vyplývá potřeba neustálého zdokonalování se v živém jazyce.

Při výuce programovacího (umělého, neživého) jazyka máme usnadněnou práci v tom, že

- veškerá syntaxe i sémantika možných konstrukcí, jichž je omezený počet, je normalizována,
- vše, co není výslovně povoleno, je zakázáno.

V dalším textu budeme pro popis syntaktických pravidel zápisů programů používat dvou druhů notací především grafů syntaxe, ale také Backus–Naurův formalismus.

[1] Standardním Pascalem nazýváme základní jádro jazyka, jehož syntaktické i sémantické definice tvoří mezinárodně schválenou normu jazyka Pascal — standardní Pascal.

1.1.1 Lexikální symboly jazyka

Program tvoříme z lexikálních symbolů jazyka. Lexikálními symboly jsou základní symboly, speciální symboly, identifikátory, čísla, návěští a řetězce znaků. Mezi dvěma lexikálními symboly může být libovolný počet mezer (někdy musí být alespoň jedna, tzv. oddělovač), konců řádků nebo komentář, což je část programu uzavřená mezi levou { a pravou }, resp. mezi (∗ a ∗), která slouží k orientaci člověka ve zdrojové verzi programu.

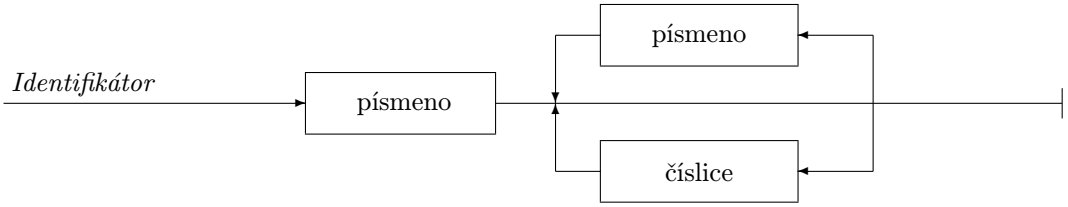
Základní symboly tvoří množina písmen a množina číslic:

písmeno	::=	$A \mid B \mid C \mid D \mid E \mid F \mid G \mid H \mid I \mid J \mid K \mid L \mid M \mid N \mid O \mid P \mid$
		$Q \mid R \mid S \mid T \mid U \mid V \mid W \mid X \mid Y \mid Z \mid a \mid b \mid c \mid d \mid e \mid f \mid g \mid$
		$h \mid i \mid j \mid k \mid l \mid m \mid n \mid o \mid p \mid q \mid r \mid s \mid t \mid u \mid v \mid w \mid x \mid y \mid z$
číslice	::=	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

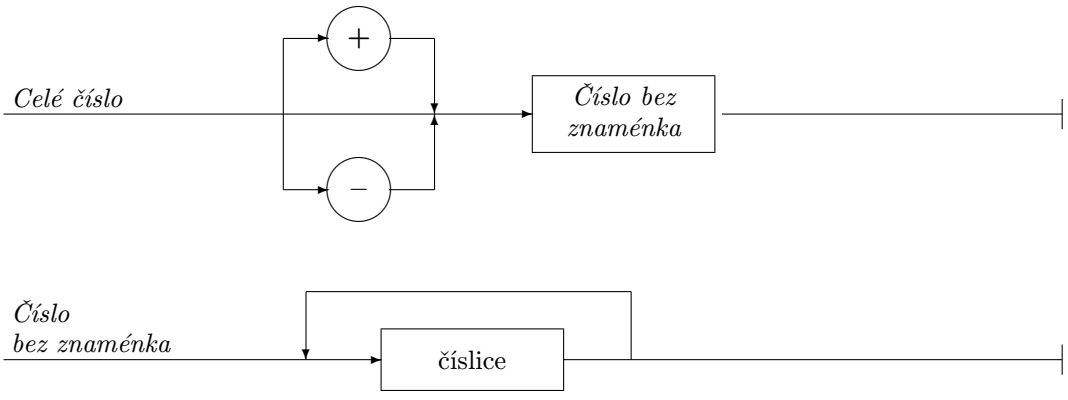
Speciální symboly jsou ty lexikální symboly, které mají pevně definovaný význam:

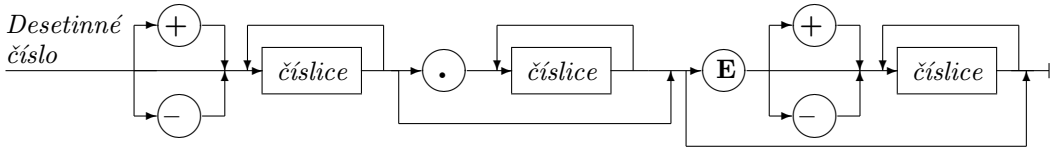
speciální symbol	::=	+ | − | ∗ | / |=|<|>| [|] | . |, |:|; | (|) | {|} | @ | .. |
		_ |:=|<=|>=|<>|′| klíčové slovo
klíčové slovo	::=	**absolute** | **and** | **array** | **begin** | **case** | **const** | **div** | **do** | **downto** | **else** | **end** | **external** | **file** | **for** | **forward** | **function** | **goto** | **if** | **implementation** | **in** | **inline** | **interface** | **interrupt** | **label** | **mod** | **nil** | **not** | **of** | **or** | **packed** | **procedure** | **program** | **record** | **repeat** | **set** | **shl** | **shr** | **string** | **then** | **to** | **type** | **unit** | **until** | **uses** | **var** | **while** | **with** | **xor**

Identifikátor (jméno) je posloupnost písmen a číslic začínající písmenem. Součástí identifikátoru může být též znak „_“ (podtržítko). Identifikátorem nemůže být klíčové slovo. Je irelevantní (nepodstatné), zda v identifikátoru je použito malé či velké písmeno. Identifikátor ahoj je identický s identifikátorem Ahoj resp. AHOJ a pod.



Čísla jsou symboly představující celá nebo desetinná čísla.





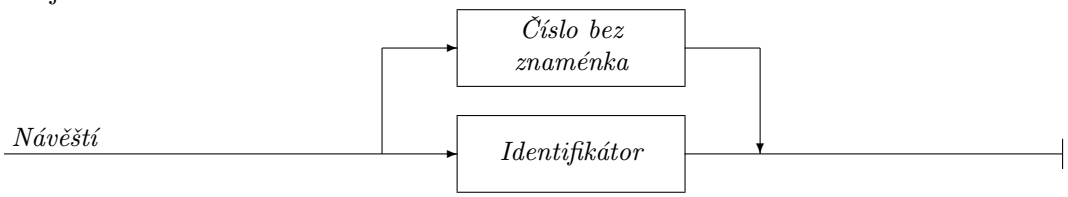
Podle uvedeného grafu synataxe pak desetinným číslem v Pascalu nazýváme zápisy např.

103.14 -0.25 +1.602E-19 6.022E23

nikoliv však zápisy např.

312. .25 9.80665E

Návěští je buď identifikátor nebo celé číslo bez znaménka.

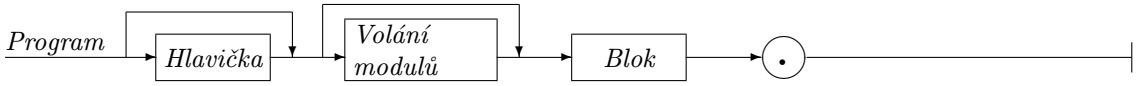


Řetězec znaků tvoříme z písmen, číslic a dalších symbolů, jež jsou např. na klávesách klávesnice. Řetězec znaků může být prázdný, může obsahovat jeden znak — hodnota standardního typu char (odstavec 1.4.1.3) — nebo může obsahovat více znaků. Řetězec je ohraničen tzv. řetězcovou závorkou zleva i zprava. Jedná se o speciální symbol ', který nazýváme apostrof.

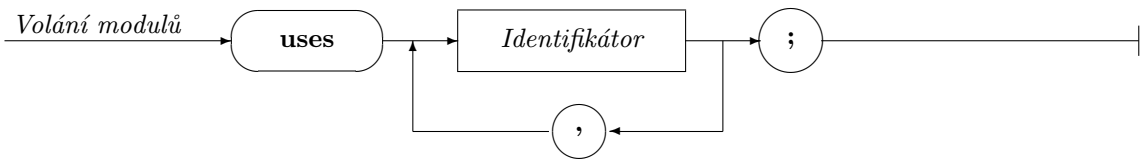
Již na tomto místě je třeba, aby si čtenář uvědomil, že řetězec znaků 'kino' je jiný než řetězec znaků 'Kino' a ten je jiný než řetězec 'KINO' a pod.
Znak 'A' je opravdu znak, avšak A je identifikátor. 'A' je něco jiného než 'a', avšak A i a je jeden a týž identifikátor, je to jméno, které identifikuje tentýž objekt.

1.2 Struktura programu

Program v Pascalu tvoří hlavička, blok a tečka. V Turbo – Pascalu je od hlavičky oddělena část specifikující použité externí moduly, jež nazýváme odstavec připojení jednotek nebo také volání modulů.

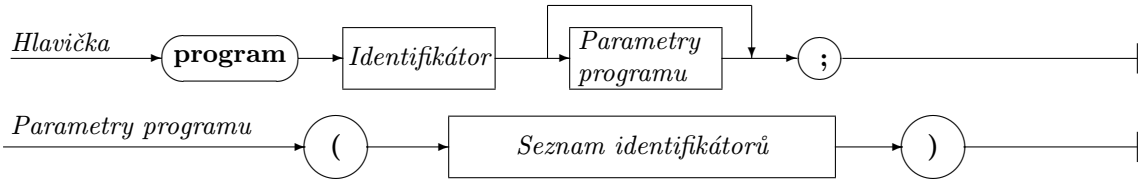


Odstavec připojení jednotek – část volání modulů – (knihoven podprogramů) píšeme v případě, že k našemu programu potřebujeme připojit některou ze standardních či vlastních jednotek (viz kap. 2.).

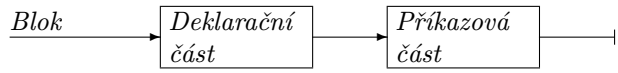


Hlavička programu je v Pascalu Turbo nepovinná; bývá zvykem ji psát nejen vzhledem k zajiš-

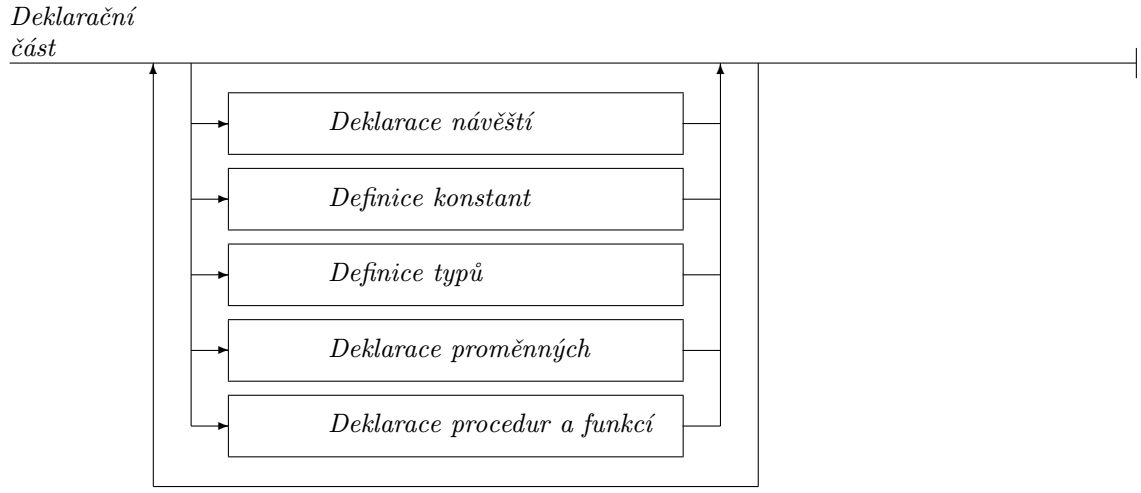
tění přenositelnosti programu pod jiné implementace Pascalů, v nichž bývá povinná.



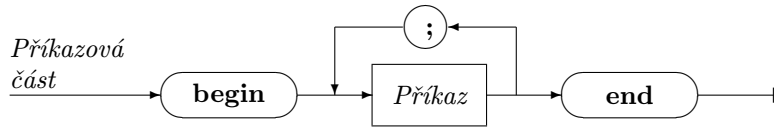
Blok je základní syntaktickou kategorií v Pascalu. Blok s hlavičkou programu tvoří program, blok s hlavičkou podprogramu vytváří podprogram a blok má zásadní význam i v modulových jednotkách.



Část definicí a deklarací může obsahovat pět odstavců, z nichž všechny jsou nepovinné (teoreticky nemusí být tedy uveden žádný – např. u programu, který pouze tiskne konstanty).



Část příkazová reprezentuje popis postupu řešení daného problému (algoritmus).



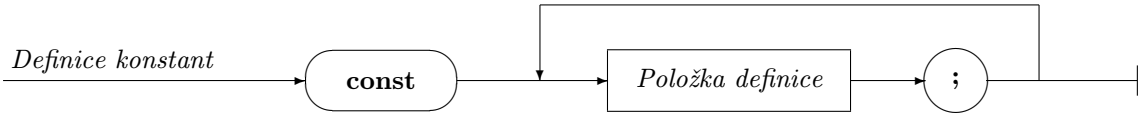
1.3 Část definicí a deklarací

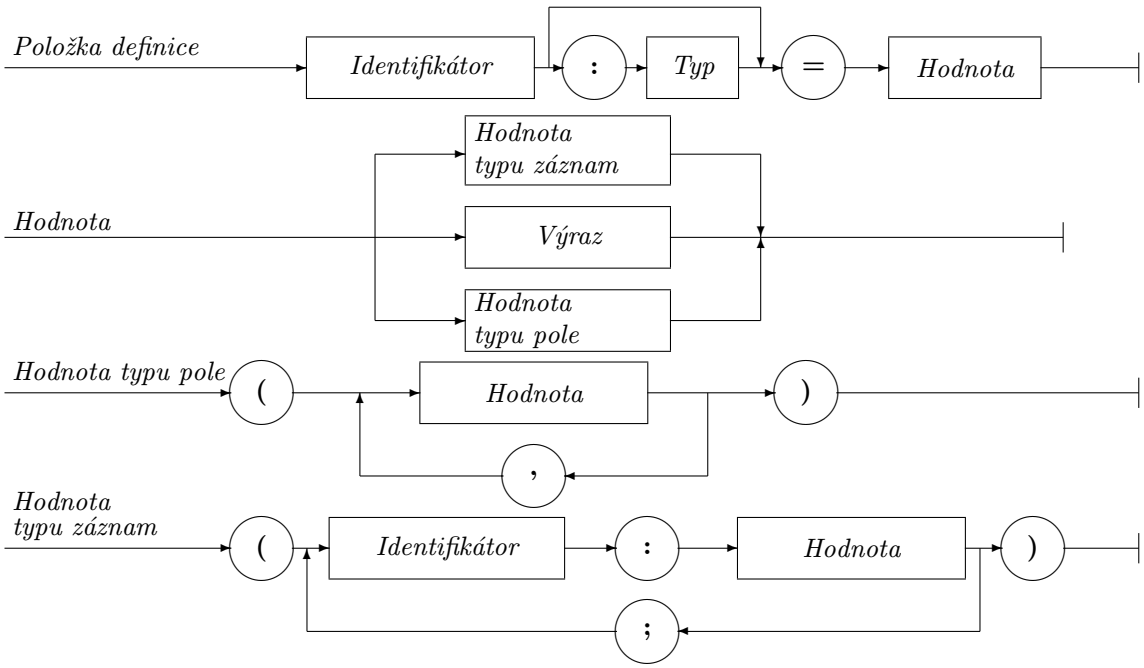
1.3.1 Deklarace návěští

Odstavec deklarace návěští je uvozen klíčovým slovem **label**, za nímž následuje seznam definovaných návěští. Poněvadž návěští potřebuje v Pascalu toliko příkaz **goto**, jenž není programátory v Pascalu oblíben (tudíž nepoužíván), považujeme další výklad o tomto odstavci za bezpředmětný.

1.3.2 Definice konstant

Odstavec definice konstant je uvozen klíčovým slovem **const** tak, jak ukazuje graf syntaxe:





Konstanta definovaná v odstavci konstant je datový objekt. Konstantu bez udaného typu nelze v příkazové části měnit (na rozdíl od proměnné), konstantu s udaným typem pak chápeme jako proměnnou s definovanou počáteční hodnotou.

♡ **Příklad:**

Definovat konstanty bez udaného typu lze zápisem např.

```
const  LC      = 3.14159;
       EL      = 1.602E–19;
       OKO     = 21;
       OZNAM   = 'chyba';
```

Definovat konstanty s udaným typem lze např. ve tvaru

```
const  MAT     : array[1..2,1..3] of byte = ((4,1,0),(5,11,6));
       SDELENI : string[5] = 'chyba';
```

Tento zápis je ekvivalentní zápisu ve tvaru

```
type   TMAT    = array[1..2,1..3] of byte;
       TSDEL   = string[5];
const  MAT     : TMAT = ((4,1,0),(5,11,6));
       SDELENI : TSDEL = 'chyba';
```

Otázka: Jaký je rozdíl mezi konstantami SDELENI a OZNAM?

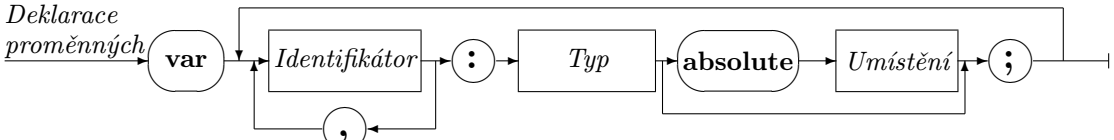
Pozn.: Nestandardnost *definic konstant s udaným typem* nutí v uvedeném zápisu porušit zásadu o struktuře části definicí a deklarací, která ve standardu přikazuje uvádět odstavec definicí konstant před odstavcem definicí typů.

♠ **Konec příkladu.**

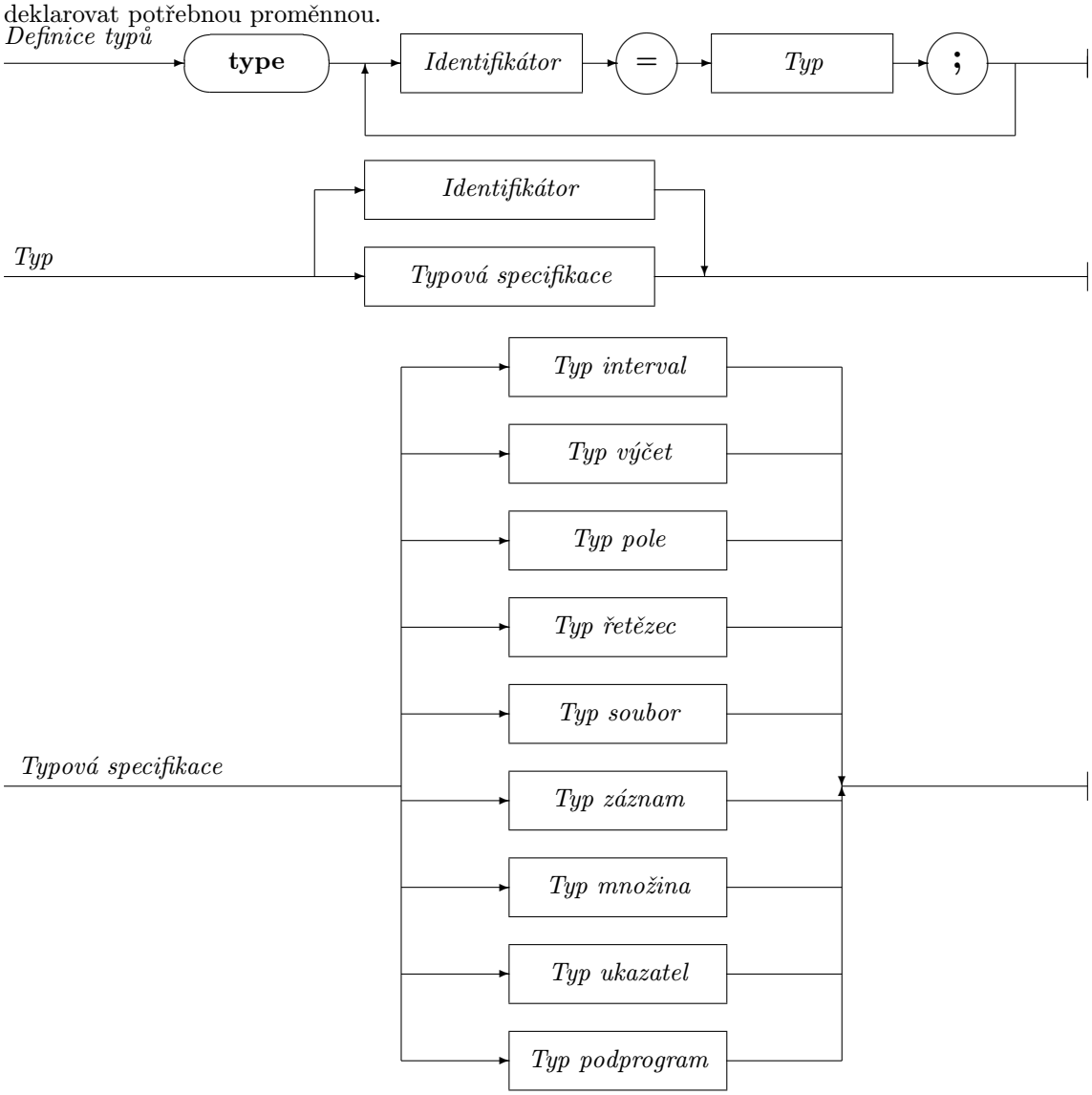
1.3.3 Definice typů a deklarace proměnných

Odstavec definic typů a odstavec deklarací proměnných mají v programu zásadní význam.

Proměnná je datový objekt, jehož hodnota se může v průběhu výpočtu měnit. Každá proměnná zabírá v paměti počítače určité místo, do kterého můžeme dát hodnotu. Velikost zabírané paměti i množina přípustných hodnot proměnné je určena typem proměnné. Datový typ proměnné (dále jen typ) a identifikátor (název) proměnné se zavádí v okamžiku deklarace proměnné v odstavci deklarací proměnných.



Řada typů (množin přípustných hodnot) je v Turbo Pascalu předdefinována — hovoříme o standardních datových typech, řadu typů si může uživatel definovat sám v odstavci definice typů. Zvykněme si v programu nejdříve definovat potřebný datový typ a teprve poté na jeho základě deklarovat potřebnou proměnnou.



Typ může být reprezentován svým názvem — identifikátorem typu. Tak např. identifikátor integer reprezentuje konečnou a souvislou množinu celých čísel počínaje číslem -32768 a konče číslem 32767.

Nadeklarujeme–li proměnnou A typu integer následujícím zápisem v části deklarací proměnných

var A : integer;

pak v paměti počítače vznikne prostor (buňka, datový objekt, proměnná), který se bude nazývat A a který bude v každém okamžiku moci obsahovat jednu hodnotu z výše uvedené množiny přípustných hodnot typu integer. Prostor bude zabírat 2 byte, (tj. 16 bitů).

V další části se naučíme definovat své typy, pomocí nichž a typů standardních budeme deklarovat proměnné.

1.3.4 Deklarace podprogramů

Problematice podprogramů v její kompletní šíři je věnována samostatná kapitola 2.

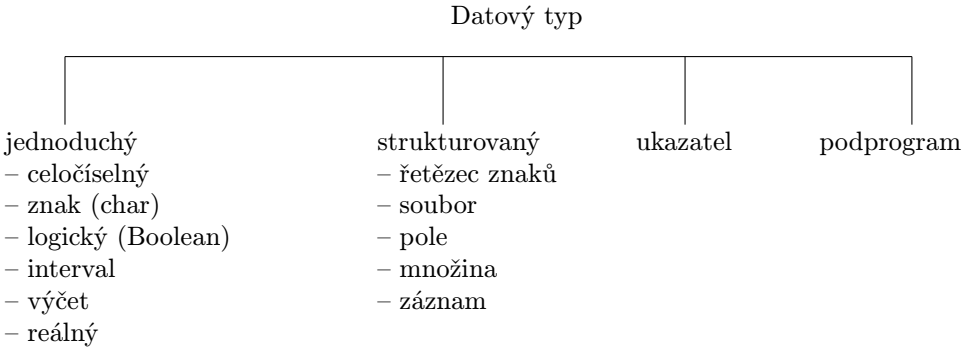
1.4 Datové typy

Datový typ určuje

a) obor přípustných hodnot proměnné, velikost potřebného místa pro proměnnou v paměti a způsob zobrazení přípustných hodnot v paměti počítače

b) přípustné operace nad těmito hodnotami

Datové typy si zpravidla definujeme v odstavci definic typů, hojně používané typy jsou již předdefinovány — hovoříme o standardních datových typech.



1.4.1 Jednoduché datové typy

Jednoduché typy dat charakterizují množiny hodnot, které se z hlediska operací použitelných v programu jeví jako elementární. Typickými jednoduchými typy jsou číselné typy. Jednoduchým typem rozumíme uspořádanou konečnou množinu hodnot, nad kterými jsou definovány relace „menší než“ a „větší než“. Jednoduché datové typy vyjma typu real tvoří důležitou skupinu tzv. ordinálních typů. Ordinální typ tvoří nejen uspořádaná, ale i souvislá konečná množina hodnot, tj. taková množina hodnot, pro jejíž každou hodnotu (vyjma prvé) je definován také její předchůdce dosažitelný např. funkcí *pred* a její následník (vyjma poslední) dosažitelný např. funkcí *succ* a jejíž každé hodnotě je přiřazeno ordinální (pořádkové, pořadové) číslo, dosažitelné funkcí *ord*. V každé implementaci Pascalu se setkáváme s předdefinovanými — tzv. standardními datovými typy

integer	–	konečná souvislá podmnožina celých čísel
real	–	konečná podmnožina reálných čísel
char	–	množina znaků (tabulka ASCII kódu)
Boolean	–	množina logických hodnot false a true

a s možností definovat si svůj

- typ interval a
- typ definovaný výčtem.

V Turbo Pascalu jsou číselné standardní datové typy bohatěji zastoupeny.

1.4.1.1 Celočíselné typy

Oborem základního celočíselného typu označeného rezervovaným identifikátorem integer je konečná souvislá podmnožina celých čísel (podmožina proto, že kapacita paměti počítače je konečná). Rozsah hodnot typu integer není definován jazykem, ale konkrétní implementací. Jazykem je definována standardní konstanta MAXINT, jejíž hodnota je implementačně závislá a je to největší číslo typu integer.
V Turbo–Pascalu je MAXINT=32767 a proměnná typu integer zabírá v paměti počítače prostor 2B.

Vedle základního celočíselného datového typu integer jsou v Turbo–Pascalu použitelné i další celočíselné datové typy:

identifikátor	množina přípustných hodnot	potřebný prostor
shortint	-128..127	1 B
integer	-32768..32767	2 B
longint	-2147483648..2147483647	4 B
byte	0..255	1 B
word	0..65535	2 B

Pro celočíselný typ argumentu je definována řada standardních funkcí, jako např. abs, sqr aj. (viz odstavec 7.6.1). Pro celočíselné operandy jsou definovány aritmetické operátory +, −,∗,/, **div** (celočíselné dělení) a **mod** (zbytek po celočíselném dělení) — viz odstavec 1.2.

Pozor

- Výsledkem dělení dvou celočíselných hodnot při použití operátoru / je hodnota typu real.
- Pracujeme-li s hodnotami a proměnnými např. typu byte (tj. v intervalu 0..255), pak zápis
  C:= 150;
  A:= 140 + C − 70 vede obecně k chybnému výsledku (přetečení), zapíšeme-li jej ve tvaru
  A:= 140 − 70 + C bude výsledek vždy správný.

1.4.1.2 Typy reálných číselných hodnot

Oborem základního reálného číselného typu označovaného rezervovaným identifikátorem real je konečná podmnožina množiny reálných hodnot. Tato podmnožina je opět stanovena implementací. V Turbo–Pascalu proměnná typu real zabírá v paměti prostor 6B.

Reálné (desetinné) číslo v Pascalu má svoji celou část a desetinnou část, mezi nimiž se píše desetinná tečka (ne tedy čárka, jak jsme zvyklí z běžného zápisu).

Př.: 14.58 nebo také 1.458E1 ale nikdy 14,58
0.00486 nebo také 4.86E-3 ale nikdy 0,00486

Vedle základního reálného číselného datového typu real jsou v Turbo–Pascalu použitelné i další reálné datové typy:

identifikátor	množina přípustných hodnot			platných číslic	prostor
real	$\pm 2.9 * 10^{-39}$	..	$\pm 1.7 * 10^{38}$	11–12	6 B
single [2]	$\pm 1.5 * 10^{-45}$	..	$\pm 3.4 * 10^{38}$	7– 8	4 B
double [2]	$\pm 5.0 * 10^{-324}$	..	$\pm 1.7 * 10^{308}$	15–16	8 B
extended[2]	$\pm 3.4 * 10^{-4932}$	..	$\pm 1.1 * 10^{4932}$	19–20	10 B
comp [2]	$-9.2 * 10^{18}$	..	$+9.2 * 10^{18}$	19–20	8 B

Pro reálný typ argumentu je definována řada standardních funkcí, jako např. sin, sqrt aj. (viz odstavec 7.6.1). Nelze použít standardní funkce pro argument ordinálního typu (např. pro definici následníka resp. předchůdce, neboť jak víme, každé reálné číslo má nekonečně mnoho předchůdců i následníků).

Pro reálné operandy jsou definovány aritmetické operátory +,−, ∗, / — viz odstavec 1.5.1.1.

Při práci především s reálnými hodnotami je třeba si uvědomit, že aritmetika počítače je konečná a že tedy některé matematické zákony (asociativní a distributivní) zde platí pouze v omezeném rozsahu. Můžeme se setkat nejen s problémem nežádoucího zoakrouhlení, ale i s problémem tzv. přetečení (přeplnění vyhrazeného paměťového místa) či podtečení.

1.4.1.3 Typ char (znaková hodnota)

Proměnná typu char zabírá v paměti počítače 1B. Je to prostor, na kterém můžeme rozlišit 256 různých stavů. Množinu přípustných hodnot tvoří 256 různých typografických znaků, které jsou

[2] Typy single, double, extended a comp je možno od verze 5.5 použít i tehdy, není-li náš počítač vybaven matematickým koprocesorem, neboť v tom případě dochází automaticky k jeho emulaci. Přechod na využívání služeb koprocesoru laděnému programu sdělíme buď zařazením před program řádky obsahující direktivu N pro překladač, tj. {$N+}, nebo přepnutím možnosti Numeric processing z polohy software do polohy 8087/80287 v položce Compiler nabídky Options. (Ve verzi 5.0 bylo třeba při nepřítomnosti koprocesoru jeho emulaci vyvolat direktivou překladače E, tj. řádek před programem měl tvar {$E+,N+})

dle mezinárodně dohodnutých pravidel poskládány a vytváří kód, např. ASCII kód (u osobních počítačů – viz str. 14–15), ISO kód, EBCDIC kód aj. Na osobních počítačích rozšířený ASCII kód je dále uveden. Tomuto kódu jakožto transformaci množiny znaků do množiny celých čísel odpovídá standardní funkce *ord*, inverzní transformaci, která celému číslu přiřazuje znak, jenž je tímto číslem kódován, odpovídá standardní funkce *chr*. Obecně tedy

celé číslo = ord(znak) a znak = chr(celé číslo)

Tedy dle ASCII kódu za předpokladu deklarací např.

```
var I : byte;
    Z : char;
```

můžeme psát

```
I:= ord('A')   nebo   Z:= 'A';
                      I:= ord(Z)
```

a proměnná I bude obsahovat celočíselnou hodnotu 65, neboť znak 'A' je v tabulce kódu ASCII na 65. místě (počítáno od nuly).
Obdobně můžeme psát

```
Z:= chr(49)   nebo   I:= 49;
                     Z:= chr(I)
```

a proměnná Z bude obsahovat hodnotu znaku '1', neboť znak '1' je na 49. místě v tabulce kódu ASCII.

Jako pro každý ordinální typ, tak i pro typ char jsou definovány funkce předchůdce *pred* a následníka *succ*:

Příkaz Z:= succ('A') popř. Z:= succ(chr(65))

bude mít za důsledek přiřazení hodnoty 'B' do proměnné Z, neboť v ASCII kódu je následníkem znaku 'A' znak 'B'.

1.4.1.4 Typ Boolean (logická hodnota)

Booleovská proměnná může nabývat jedné ze dvou logických pravdivostních hodnot (logických konstant), jež jsou označovány identifikátory *false* (tj. nepravda) a *true* (pravda). Implicitní (již existující) definici typu Boolean můžeme vyjádřit tvarem

type Boolean = (false, true);

Jako každá množina ordinálních hodnot je i množina logických hodnot uspořádaná a platí, že

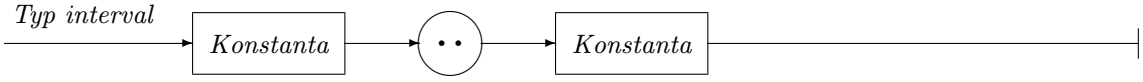
- false < true
- succ(false)= true a pred(true)= false
- ord(false)=0 a ord(true)=1

Z logických proměnných, logických konstant, logických funkcí pomocí logických operátorů tvoříme logické výrazy (způsob uveden v odstavci 1.2 — též jednoduché logické výrazy (pomocí aritmetických výrazů a relačních operátorů)).

Z logických standardních funkcí si uvedeme nyní pouze funkci odd(X), jejíž argument X je výraz celočíselného typu. Funkce nám vrací hodnotu *true*, jestliže argument je liché číslo, jinak vrací hodnotu *false*.

1.4.1.5 Typ interval

Je jeden ze dvou jednoduchých (ordinálních) datových typů, jejichž definici provádí uživatel na základě intervalu z nějakého již dříve definovaného (např. standardního) ordinálního typu.



♡ **Příklad:**

```
type   MESICE     = 1..12;
       MALEPISM   = 'a'..'z';
       NARODEN    = 1890..RZ;
```

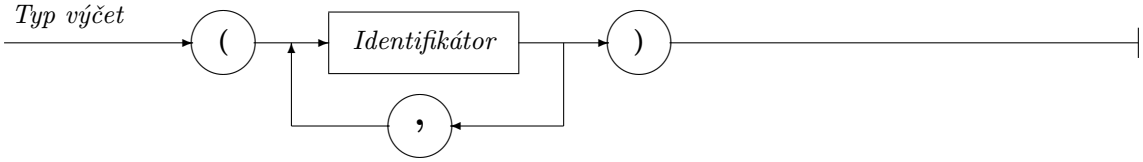
přičemž je zřejmé, že RZ musela být dříve definovaná konstanta např. způsobem

```
const  RZ    = 1995;
```

♠ **Konec příkladu.**

1.4.1.6 Typ definovaný výčtem

Uspořádaná a souvisla množina hodnot typu definovaného výčtem je dána výčtem identifikátorů. Konstantou typu definovaného výčtem nemůže být tedy nic jiného, než identifikátor (tedy ani číselná, ani znaková a ani řetězcová konstanta jako je tomu u typu interval). Je-li počet identifikátorů v definovaném typu menší nebo roven 256, pak proměnná tohoto typu obsazuje v paměti 1B, jinak 2B.



♡ **Příklad:**

```
type   NAPOJ          = (Rum,Vino,Pivo,Kava,Caj,Limonada,Voda);
       DEN            = (pondeli,utery,streda,ctvrtek,patek,sobota,nedele);
       PRACDEN        = pondeli..patek;
var    PIJI           : NAPOJ;
       PRACUJI        : PRACDEN;
       VEK            : byte;
begin
       .
       .
       VEK:=29;
       PIJI := Kava;
       PRACUJI := streda;
       .
       .
end.
```

♠ **Konec příkladu.**

Jak si řekneme v odstavci 5.1.2, nelze hodnoty do proměnné typu definovaného výčtem ani načítat z klávesnice, ani její obsah vytisknout na obrazovku. V těchto začátcích se zdá, že se s takovouto proměnnou snad ani nedá pracovat. Alespoň tedy naznačíme její možnosti, když již uvedenou výhodou je její malá paměťová náročnost a skutečnost, že její použití zčitelní program (viz níže uvedený příklad).

Hodnotě výčtového typu (jako každé hodnotě ordinálního typu) přináleží její ordinální číslo podle pořadí hodnoty (tj. v tomto případě identifikátoru) v definičním seznamu hodnot v definici typu. Tato skutečnost nám umožňuje provádět konverzi hodnoty jednoho ordinálního typu na hodnotu jiného se stejným ordinálním číslem obecně příkazem ve tvaru

proměnná 1.ord.typu := typ 1.ord.typu (výraz 2.ord.typu)

♡ **Příklad:** Na základě deklarací a definicí z výše uvedeného příkladu je možno formulovat příkazy:

```
.
.
PIJI := NAPOJ(streda); {popř. PIJI := NAPOJ(2); tj. vlastně PIJI := Pivo;}
VEK := byte(Kava); {popř. VEK := byte(ctvrtek); tj. vlastně VEK := 3; }
.
.
a v proměnné PIJI bude hodnota Pivo
a v proměnné VEK bude hodnota 3
```

♠ **Konec příkladu.**

Pozorný čtenář již jistě postřehl, že standardní funkce *ord* popř. *chr*, jež používáme ke konverzi znakových popř. číselných hodnot, lze za předpokladu deklarace

var Z : char; I : integer;

použít ve tvaru

I := ord(Z); Z := chr(I);

což je totéž, jako když napíšeme

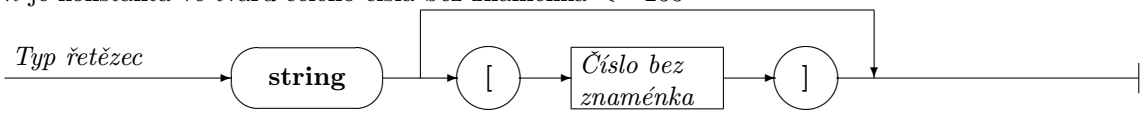
I := integer(Z); Z := char(I);

1.4.2 Strukturované datové typy

Strukturovaný datový typ je určen typem strukturování a typem složek, z nichž je daný strukturovaný typ složen. Poněvadž typ složek může být také strukturovaným typem, lze vytvářet složité strukturované datové typy s hierarchickým uspořádáním.

1.4.2.1 Typ řetězec

Proměnná typu řetězec slouží k uložení až n znaků. V paměti zabírá $n + 1$ bytů,
n je konstanta ve tvaru celého čísla bez znaménka <= 255



♡ **Příklad:**

```
   type   RETEZ = string[12];
   var    JMENO, PRIJMENI : RETEZ;
begin
.
.
          JMENO := 'Monika';
          ReadLn(PRIJMENI);
.
.
end
```

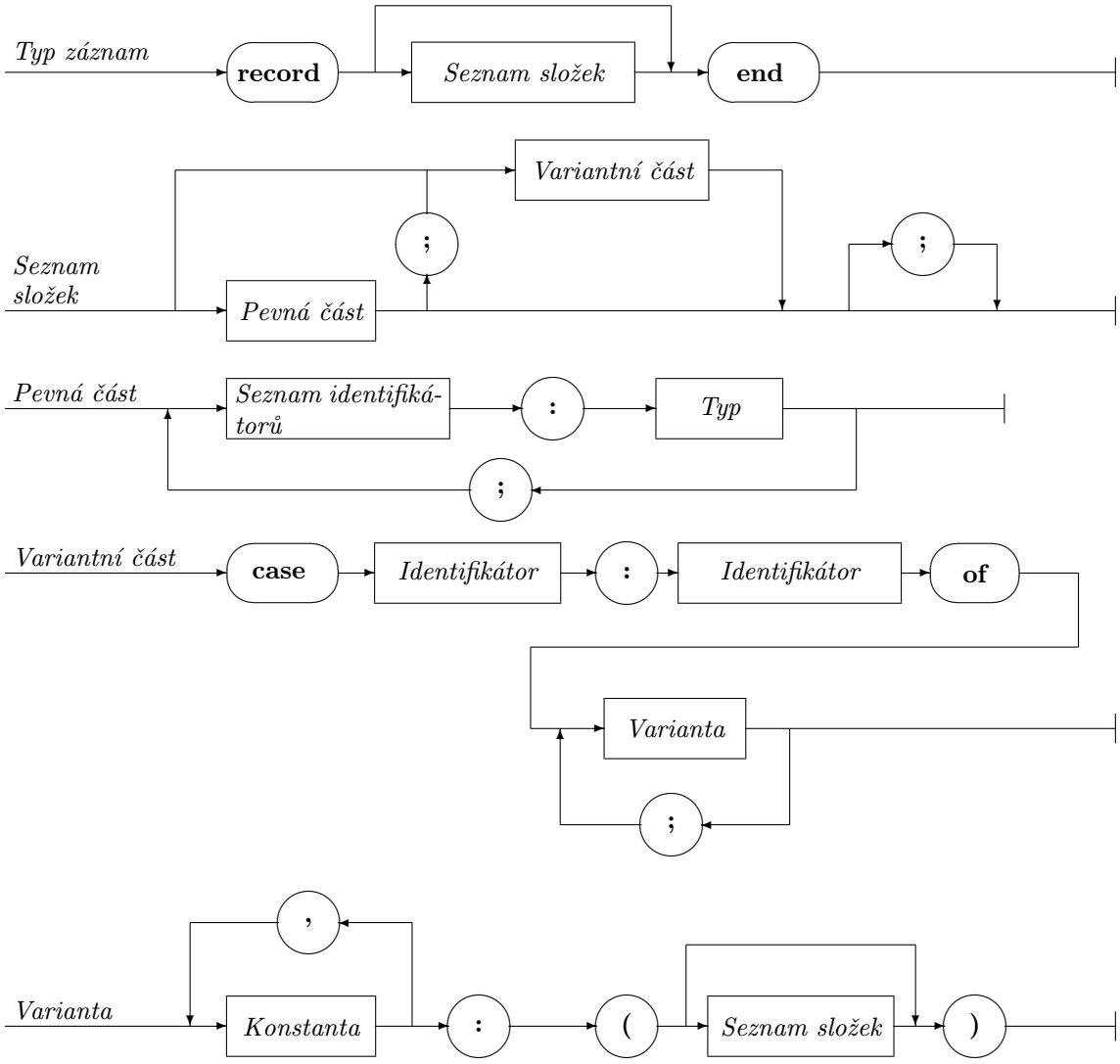
♠ **Konec příkladu.**

1.4.2.2 Typ soubor

Problematice práce se soubory (textovými i netextovými) je v tomto učebním textu věnována větší pozornost a je jí věnována kapitola 3.

1.4.2.3 Typ záznam

Záznam je heterogenní datový typ, tj. skládá se ze složek, které mohou být různého typu. Záznam jako celek a složky jako jeho části jsou pojmenovány různými identifikátory. Přístup ke složce záznamu je možný buď s využitím příkazu **with** (odst. 1.2), nebo zápisem tzv. tečkové notace, tvořené identikátory proměnné typu zaznam a identifikátorem příslušné složky, jež jsou odděleny tečkou.



♡ **Příklad:**

```
type   RETEZ = string[12];

       OSOBA  =  record   JMENO,PRIJM   : RETEZ;
                          VEK           : byte;
                          HMOTNOST      : real;
                 end;
```

```
  var   CLOVEK : OSOBA;
begin
.
.
        CLOVEK.JMENO := 'Jan';
        CLOVEK.HMOTNOST := 82.5;
        Read(CLOVEK.PRIJM,CLOVEK.VEK);
.
.
end
```

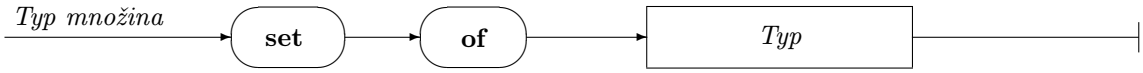
Příkazová část může být s využitím příkazu **with** také ve tvaru:

```
  var     CLOVEK : OSOBA;
begin
.
.
        with CLOVEK          do
          begin              JMENO := 'Jan';
                             HMOTNOST := 82.5;
                             Read(PRIJM,VEK);
          end
.
.
end
```

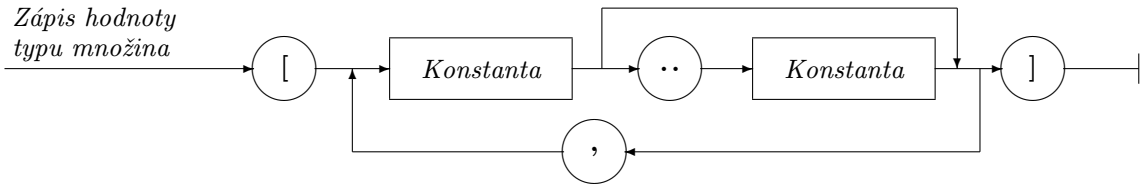
♠ **Konec příkladu.**

1.4.2.4 Typ množina

Množina jako typ reprezentuje množinu všech podmnožin bázového typu, jímž může být pouze takový ordinální typ, jehož hodnoty mají ordinální čísla v intervalu < 0..255 > (tedy ne např. integer, word apod.):



Zápis množiny sestává z prvků množiny bázového typu oddělených čárkami a uzavřených do hranatých závorek. Zápis [ ] označuje množinu prázdnou.



♡ **Příklad:**

```
type   TM1 =   set of char;
       TI =    0..7;
       TM2 =   set of TI;
       TM3 =   set of byte;
               TM4 = set of word; ⇒ chybné, neboť některé hodnoty typu word
               mají ordinální čísla > 255

var    A : TM1; B : TM2;
       C : TM3;
       Z : char; I : byte;
.
.
```

```
C:= [11,17,23,111,153,207];
A:= [ ]; B:=[ ];
for I:=1 to 5 do begin   Readln(Z);
                         A := A + [Z];
                         B := B + [I];
                 end;
.
.
```

Proměnné A i C zabírají v paměti po 256 bitech, tj. 32B (to je maximum, co může proměnná typu množina zabírat), proměnná B zabírá 1B (max osm prvků 0..7).

♠ **Konec příkladu.**

Pro práci s množinami existují tři množinové operace:

a) sjednocení množin — používá se operátor +

b) průnik množin — používá se operátor ∗

c) rozdíl množin — používá se operátor −

Z relačních operátorů je možno využít

a) operátor = pro test rovnosti

b) operátor <> pro test nerovnosti

c) operátor <= pro test podmožiny

Pro datové objekty typu množina je definován operátor **in** (klíčové slovo), pomocí kterého zjišťujeme příslušnost prvku k množině. Je-li proměnná Z typu char, pak příkaz

```
if  Z  in  ['a','e','i','o','u','y']
    then Write('je samohláska')
    else Write('není samohláska');
```

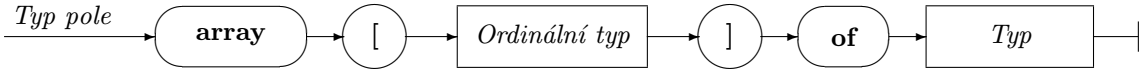
zjišťuje, zda znak uložený v proměnné Z je malou samohláskou. Obecně je přípustný zápis (výraz)

p **in** m,

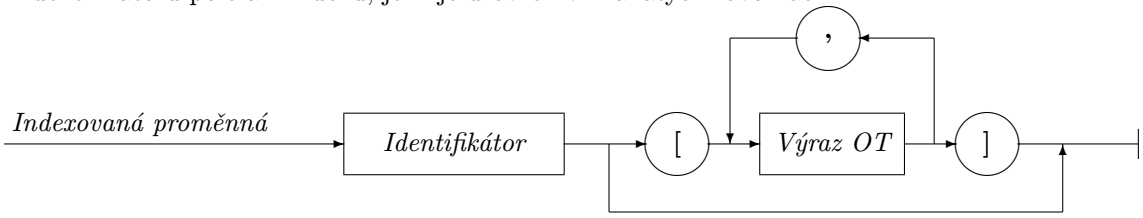
kde p je výraz typu T (bázového typu množiny)
m je výraz typu množina nad bázovým typem T,
který poskytuje hodnotu *true* pokud je p prvkem množiny m, jinak dává hodnotu *false*.

1.4.2.5 Typ pole

Pole je strukturovaný typ, sestávající z pevného počtu složek stejného typu. Pole je tedy homogenní datový typ.



Složka proměnné typu pole (tzv. indexovaná proměnná) je zpřístupněna zápisem, který sestává z identifikátoru pole a z indexu, jenž je uzavřen v hranatých závorkách :



Třeba poznamenat, že v uvedeném grafu syntaxe je identifikátorem rozuměn identifikátor proměnné typu pole a výrazem pak pouze výraz ordinálního typu (viz dále).

♡ **Příklad:**

```
type  RADEK = array[−1..1] of shortint;
      MATICE= array['a'..'d'] of RADEK;
var   M : MATICE;
      Z : char;
      I : integer;
```

Deklarací vznikla v operační paměti počítače proměnná M, která je složena ze čtyř řádků ($'a'$ až $'d'$), tří sloupců ($-1, 0a1$) a lze do ní uložit hodnoty typu shortint. Zabíra tedy $4 * 3 * 1B = 12B$.
Poznamenejme, že ke stejné proměnné jsme mohli přijít i při použití zkráceného zápisu definice typu ve tvaru

type MATICE = **array**['a'..'d',-1..1] **of** shortint;

a následující deklarace v již uvedeném tvaru

var M : MATICE;

V příkazové části programu pak můžeme psát příkazy:

for I:=0 **to** 1 **do** M['d',I] := I+4;

M['b',1] := −12;

for Z:='a' **to** 'c' **do** M[Z,-1] := 0;

Poté obsah matice M je následující:

M	-1	0	1
'a'	0	nedef.	nedef.
'b'	0	nedef.	–12
'c'	0	nedef.	nedef.
'd'	nedef.	4	5

♠ **Konec příkladu**

1.4.2.6 Typ ukazatel

Datový typ ukazatel ani práce s dynamickými datovými objekty není obsahem této učební pomůcky.

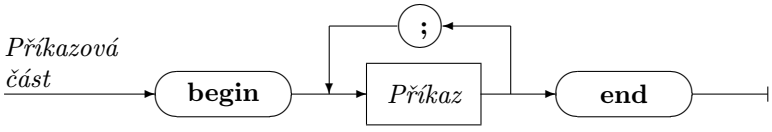
1.4.2.7 Typ podprogram

Podprogramům je věnována zvláštní kapitola (kap. 2.), ve které se zmiňujeme i o typu podprogram.

1.5 Část příkazová

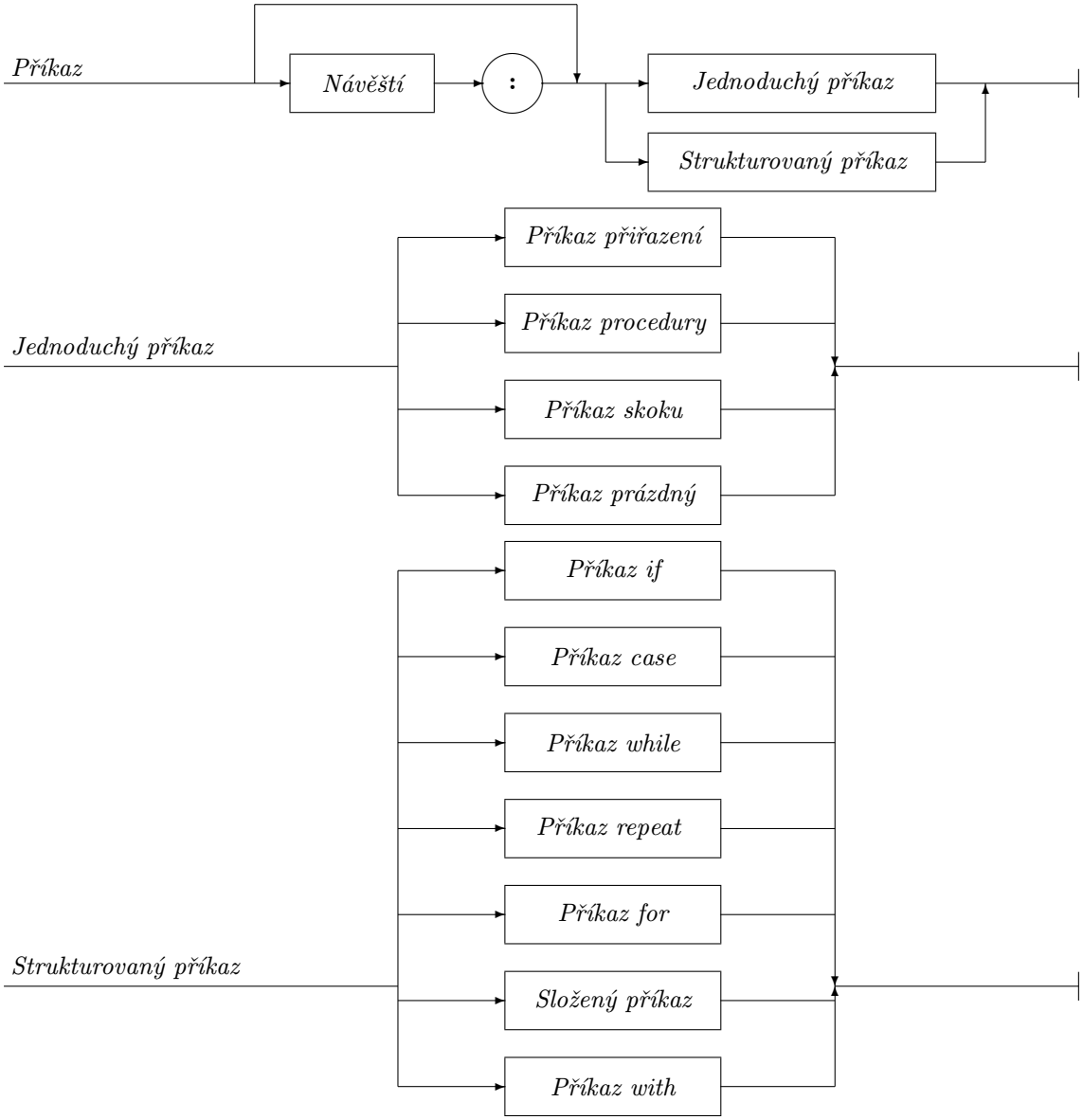
V příkazové části používáme syntakticky i sémanticky pevně definované pascalovské konstrukce — *příkazy* —, pomocí nichž se snažíme zapsat řešení našeho problému. V Pascalu použitelných příkazů je poměrně malý počet a jejich úkolem je pracovat s datovými objekty uvedenými v *části definicí a deklarací.*

Příkazová část bloku začíná klíčovým slovem **begin** a končí klíčovým slovem **end**. Mezi těmito slovy — příkazovými závorkami — se nacházejí jednotlivé příkazy, vzájemně oddělované znakem ; (středník).



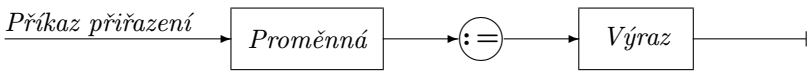
Příkaz je pascalovská jazyková konstrukce, která má svoji jednoznačnou syntaktickou a sémantickou definici. V Pascalu rozlišujeme

- příkazy jednoduché, jsou čtyři a jeden z nich (příkaz skoku **go to**) ani nepoužíváme,
- příkazy strukturované, kterých je celkem sedm.



1.5.1 Jednoduché příkazy

1.5.1.1 Příkaz přiřazovací



Přiřazovací příkaz obecně zapsaný ve tvaru

$$P := V$$

se skládá ze tří částí:

- Z levé strany, kterou tvoří identifikátor proměnné,
- z pravé strany, tzv. výrazu a
- operátoru přiřaď (dva znaky :=).

V závislosti na typu proměnné, jejíž identifikátor tvoří levou stranu přiřazovacího příkazu, hovoříme o

číselném (aritmetickém) výrazu, logickém výrazu apod.

Výraz (např. aritmetický) tvoříme z aritmetických proměnných číselných či reálných typů, z aritmetických konstant, zápisů aritmetických funkcí a s použitím aritmetických operátorů.Pro zvýraznění priority vyčíslování jednotlivých částí výrazu používáme kulatých závorek tak, jak to běžně známe z matematiky.

Aritmetický výraz za předpokladu že A, B a C jsou proměnné číselného typu je např. zápis:

B*(C-3)/(2*sin(A/B)) nebo (A+B+C)/2 nebo 4 nebo B.

Zpracování přiřazovacího příkazu počítačem:

Počítač vyčíslí nejdříve hodnotu výrazu. Je jasné, že obsah všech proměnných, které se na formulaci výrazu podílejí, musí být v okamžiku vyčíslování výrazu definován. Hodnota, která vyčíslením výrazu vznikne, je uložena do proměnné, jejíž identifikátor tvoří levou stranu přiřazovacího příkazu. Tato proměnná musí být identického (stejného) nebo kompatibilního (slučitelného) typu s typem hodnoty získané výpočtem výrazu.

Je-li deklarce

var A,B,C : integer;

a v příkazové části části se vyskytne příkaz

C:=A+B;

pak říkáme, že hodnota výrazu A+B je identického typu s proměnnou C.

Proměnná a vyčíslená hodnota je kompatibilního (slučitelného) typu tehdy, jestliže vyčíslená hodnota je typu, který můžeme považovat za podmnožinu typu proměnné. Za předpokladu deklarační části

```
var   R : real;
      I : integer;
      B : byte;
      L : longint;
```

můžeme psát příkazy	avšak ne příkazy
R:=I;	I:=R;
I:=B;	B:=I;
L:=I;	I:=L;
R:=I/I;	L:=I/I;

Aritmetické operátory jsou + | − | ∗ | / | div | mod

Tabulky výsledků aritmetických operací v závislosti na typu hodnot obou operandů:

operátor je	operand 1.	
+ , - , *	integer	real
integer	integer	real
operand 2.		
real	real	real

operátor je	operand 1.	
/	integer	real
integer	real	real
operand 2.		
real	real	real

operátor je	operand 1.	
div , mod	integer	real
integer	integer	nedef.
operand 2.		
real	nedef.	nedef.

Výsledkem standardních aritmetických funkcí *abs* a *sqr* je hodnota stejného typu jako argument funkce.

Výsledkem standardních aritmetických funkcí *sin, cos, ln, exp, sqrt, arctan* je vždy hodnota typu real.

Obdobně jednoduchý logický výraz tvoříme z aritmetických výrazů navzájem spojených pomocí relačního operátoru, složitější logické výrazy pak z jednoduchých logických výrazů, logických proměnných, logických funkcí s použitím logických operátorů.

Relační operátory jsou <|<=|>|>=|=|<>| **in**

Logické operátory jsou **and** | **or** | **not** | **xor**

Následující tabulka přináší definice logických operací:

op1	op2	op1 **and** op2	op1 **or** op2	op1 **xor** op2	**not** op1
false	false	false	false	false	true
false	true	false	true	true	true
true	false	false	true	true	false
true	true	true	true	false	false

1.5.1.2 Příkaz procedury

Vzhledem k tomu, že procedurám bude v tomto textu věnována velká pozornost (Kapitola 2) poznamenejme zde pouze, že příkaz procedury — volání procedury — sestává ze zápisu identifikátoru procedury, který případně doprovází seznam skutečných parametrů uvedený mezi kulatými závorkami bezprostředně za identifikátorem procedury.

Zde by již čtenář měl vědět o procedurách pro čtení vstupních hodnot *read* resp. *readln* a pro tisk výstupních hodnot *write* resp. *writeln*. Tyto procedury (podprogramy) jsou standardní (někdo je pro nás udělal) a umožňují přenos dat např. z klávesnice do operační paměti, či z operační paměti na obrazovku.

Již při této příležitosti upozorňujeme, že pomocí klávesnice (z textového souboru) je možno číst postupně vždy do jedné proměnné typu:

char, celočíselného i reálného a řetězec.

♡ **Příklad:** Předpokládejme deklaraci

```
var   A,B,C   : real;
      I,J     : integer;
      Z1,Z2   : char;
```

a tři řádky vstupních dat ve tvaru(psáno od prvního sloupce):

```
7  -14.2  6     3
8  -5     -1.1  5
3  3      3     3
```

(mezi jednotlivými číselnými hodnotami musí být alespoň jeden číselný oddělovač, kterým je pouze mezera. Mezičíselným oddělovačem v Pascalu není ani čárka, ani středník.)

Pak několik příkazů pro čtení

a)
```
Read(I,A);
Readln(J);
Readln(B,C);
Read(Z1,Z2);
```

b)
```
Readln(I,A);
Read(J);
Readln(B,C);
Readln(Z1);
Readln(Z2);
```

má za důsledek definování obsahu příslušných proměnných takto:

	a)	b)
A	-14.2	-14.2
B	8.0	-5.0
C	-5.0	-1.1
I	7	7
J	6	8
Z1	'3'	'3'
Z2	' '	chyba

Čtecí hlava bude stát

a) na druhé trojce třetího,

b) na začátku čtvrtého řádku vstupních dat (neexistujícího) řádku.

Rozdíl procedury *read* a *readln* je tedy v tom, že při použití *readln* se po načtení uvedených hodnot čtecí hlava přesouvá na začátek nového řádku (i když na čteném řádku nějaké hodnoty zůstávají nepřečteny – přeskakují se).

♠ **Konec příkladu**

Přehlednost programu zvyšuje použití výčtových typů. Máme-li definován typ:

type SKOLA=(základní,učňovská,střední,vysoká);

a deklarovánu proměnnou:

var VZDELANI : SKOLA;

pak je možno v programu psát dotazy v pochopitelném tvaru

if VZDELANI = *střední* **then** ...
místo dosud často obvyklého tvaru
if VZDEL = 62039 **then** ... (VZDEL je např. typu *word*)

Problém je, že nelze z klávesnice hodnotu výše definovaného typu SKOLA přímo načíst a ani ji na obrazovku přímo vytisknout. Pozorný čtenář si však s ohledem na konec odstavce 1.4.1.6 dovede poradit. Nadeklaruje ještě celočíselnou proměnnou např. I (stačí typu interval 0..4), do které lze s klávesnice načítat. Pak příkazem

Read(I); načítá ordinální hodnotu výčtového typu a příkazem

VZDELANI := SKOLA(I); ji převádí na potřebnou výčtovou hodnotu.
Načte-li např. I=2, pak v proměnné VZDELANI dostává hodnotu *střední*.

Za předpokladu uvedených deklarací a načtení hodnot dle příkazů uvedených v možnosti a) můžeme obsahy proměnných vytisknout pomocí příkazů procedur write nebo writeln na obrazovku.

Je třeba upozornit, že na obrazovku je možno zapisovat postupně vždy obsah jedné proměnné typu:

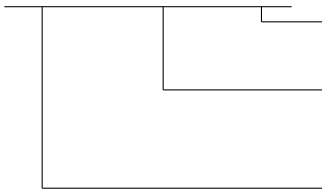
char, celočíselného i reálného, Boolean a řetězec.

Příkazy write(A,I); a writeln(Z1);
mají stejný význam jako příkaz writeln(A,I,Z1)
nebo příkazy write(A); write(I); writeln(Z1);
a způsobí celkem nepřehledný tisk ve tvaru:

-1.42000000E0173 (tzv. normalizovaný tvar)

Příkaz writeln(A:8:2,I:4,Z1:3); nám dá tisk ve tvaru

-14.20 7 3



tři pozice pro tisk obsahu znakové proměnné Z1

čtyři pozice (na nichž jsou zleva tři mezery a pak 7) pro tisk obsahu celočíselné proměnné I

osm pozic pro tisk obsahu reálné proměnné A ve tvaru se dvěma desetinnými místy. Specifikace např. A:8:3 vede k tisku -14.200 apod.

Necháme-li pro výstup obsahu proměnné málo místa,na které se nevejde ani jeho celá část, pak obsah proměnné vystupu je opět v normalizovaném tvaru.

Výstup desetinných míst je prováděn automaticky v zaokrouhleném tvaru na tolik míst, kolik požadujeme.

Příkaz *writeln*; bez parametrů se používá k vynechání prázdného řádku ve výstupní sestavě.

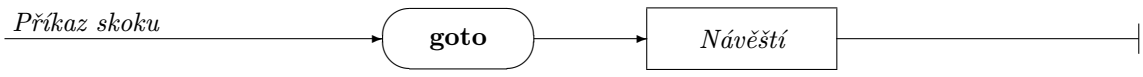
1.5.1.3 Příkaz prázdný

Syntaktická definice prázdného příkazu má tvar

Smysl poznáme vzápětí při používání příkazových struktur (usnadňuje život programátorovi v oblasti psaní příkazového oddělovače ;.

1.5.1.4 Příkaz skoku

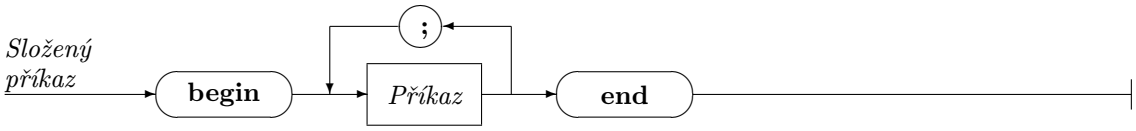
Jeho přítomnost v programu může pouze zhoršit jeho čitelnost. V Pascalu absolutně (téměř) nepotřebný. Je používán staršími programátory, jejichž první kroky se ubíraly jazykem Fortran, případně jejich žáky v jazyku Basic. Prosíme studenty, aby příkaz skoku ve svých programech nepoužívali, případně jeho použití konzultovali s vyučujícím.



1.5.2 Strukturované příkazy

1.5.2.1 Příkaz složený

Používá se tehdy, potřebujeme-li několik příkazů napsat na místě, kde syntaktická definice umožňuje psát příkaz jeden.

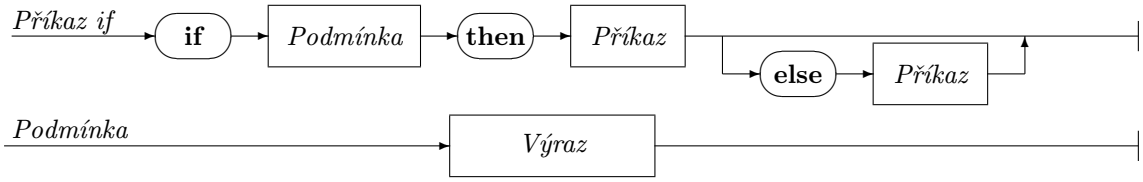


Čtenář nechť si laskavě všimne, že za posledním příkazem před klíčovým slovem **end** se dle syntaktické definice středník nepíše. Tento fakt by byl zdrojem častých chyb a (nejen) proto je zaveden příkaz prázdný, jenž psaní inkriminovaného středníku legalizuje (neboť za středníkem před **end** může být právě příkaz prázdný).

1.5.2.2 Příkaz podmíněný

V Pascalu rozlišujeme

- příkaz podmíněný úplný a
- příkaz podmíněný neúplný.



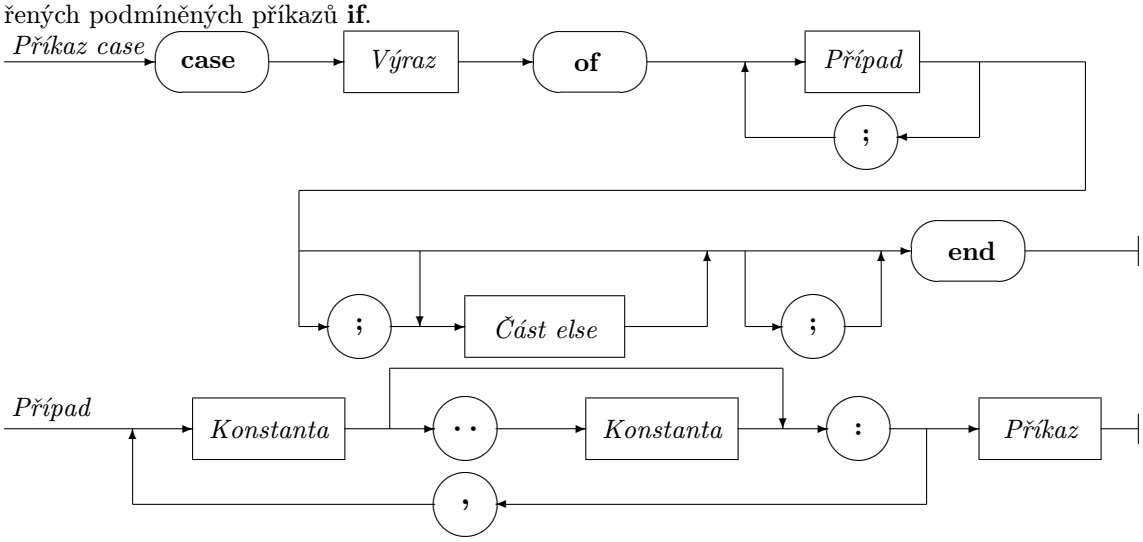
Příkaz používaný pro větvení programu. Podmínku tvoří nám již známý logický výraz. Jestliže podmínka je splněna, provádí se příkaz uvedený ve větvi **then**, není-li podmínka splněna, provádí se příkaz větve **else**.

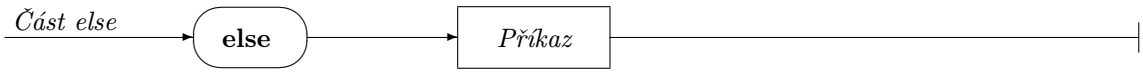
Je -li větev **else** vynechána, hovoříme o neúplném podmíněném příkazu.

Upozorňujeme čtenáře, že za klíčovým slovem **then** resp. **else** může být uveden příkaz, tj. jeden příkaz (může jim být např. příkaz složený).

1.5.2.3 Příkaz selektivní

Selektivní příkaz opět používáme pro větvení programu. Úspěšně nahrazuje několik do sebe vnořených podmíněných příkazů **if**.





Výraz za **case**, tzv. *selektor* je výraz ordinálního typu a stejného typu musí být i konstanty, které rozhodují o větvi, v níž bude příkaz proveden.

♡ **Příklad:** Máme určit, kolik je v programu jednotlivých aritmetických operátorů +, -, * nebo /.

S použitím příkazu **if** může mít část algoritmu tvar

```
.
.
if ZN='+'   then PP:=PP+1
            else if ZN='-'      then PM := PM+1
                                else if ZN='*'      then PK:=PK+1
                                                    else if ZN='/' then PD:=PD+1;
.
.
```

Použití selektivního příkazu **case** vede k jasnějšímu zápisu

```
.
.
case   ZN of
       '+': PP:=PP+1;
       '−': PM:=PM+1;
       '∗': PK:=PK+1;
       '/': PD:=PD+1;
end;
.
.
```

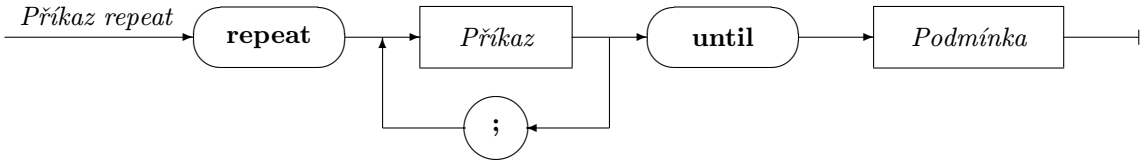
♠ **Konec příkladu**

1.5.2.4 Příkaz cyklu

Příkazem cyklu se zapisuje opakované provádění určité části algoritmu —- tzv. těla cyklu (opakující se část). Opakování je podmíněno opět splněním či nesplněním určité podmínky. Při špatně formulované podmínce, či při opomenutí modifikovat některou z jejích komponent v opakující se části velice snadno naprogramujeme tzv. nekonečný cyklus (pak často nezbývá nic jiného, než vypnout počítač se všemi důsledky z tohoto počinu vyplývajícími).

1.5.2.4.1 Příkaz cyklu repeat — until

Příkazy uvedené mezi klíčovými slovy **repeat** a **until** se opakovaně provádějí tak dlouho, dokud není podmínka uvedena za **until** splněna.



Z definice příkazu vyplývá, že tělo cyklu se provede alespoň jednou.

♡ **Příklad:** Sečtěte všechna čísla z intervalu $< A, B >$ s krokem C.
Hodnoty A, B, C mohou být reálné popř. celočíselné.
Platí, že hodnota $A <= B$.

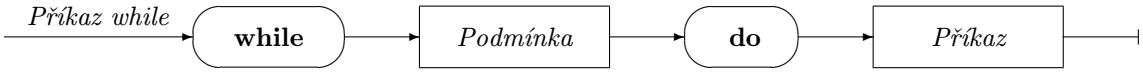
```
Program SOUCET;
  var   A,B,C,POM,VYSL : real;
```

```
begin Write('Zadej meze intervalu A,B a krok C:');
      ReadLn(A,B,C);
      VYSL := 0; POM := A;
      repeat VYSL := VYSL + POM;
             POM := POM + C
      until  POM > B;
      Writeln('Výsledna hodnota je ',VYSL:10:2);
end.
```

♠ **Konec příkladu**

1.5.2.4.2 Příkaz cyklu while - do

Příkaz uvedený za klíčovým slovem **do** se opakovaně provádí tak dlouho, dokud je podmínka uvedená za **while** splněna.



Z definice příkazu vyplývá, že tělo cyklu se nemusí provést ani jednou.

♡ **Příklad:** Výše uvedený příklad s použitím příkazu cyklu **while** – **do** bude mít tvar:

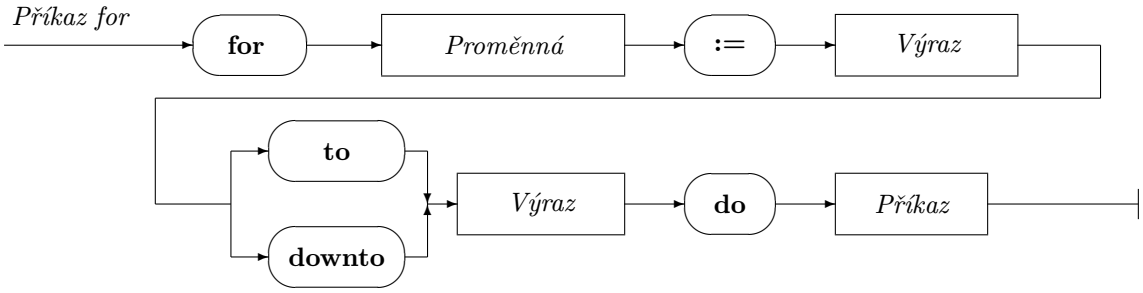
```
Program SOUCET;
  var   A,B,C,POM,VYSL : real;
  begin Write('Zadej meze intervalu A,B a krok C:');
        ReadLn(A,B,C);
        VYSL := 0; POM := A;
        while POM <= B do begin VYSL := VYSL + POM;
                                POM := POM + C
                          end;
        Writeln('Výsledna hodnota je ',VYSL:10:2);
  end.
```

♠ **Konec příkladu**

1.5.2.4.3 Příkaz cyklu for - to - do resp. for - down - to

Budeme jej používat až při práci s indexovanými proměnnými. Jeho uvedení na tomto místě je pouze z hlediska logické návaznosti, nikoli z hlediska praktické použitelnosti.

Opakovaně se provádí příkaz stojící za klíčovým slovem **do**. Počet opakování je pevný a je dán počtem hodnot, kterými je postupně definován obsah proměnné ordinálního typu — tzv. řídící proměnné cyklu. Hodnoty tvoří rostoucí resp. klesající posloupnost hodnot počínaje hodnotou přiřazenou proměnné cyk- lu příkazem uvedeným za **for** a konče hodnotou ordinálního typu uvedenou za klíčovým slovem **to** resp. **downto**.



♡ **Příklad:** Pokud by z výše uvedeného příkladu byly hodnoty A, B ordinálního typu (např. celočíselného), pak součet všech hodnot z intervalu $< A, B >$ bychom mohli získat programem:

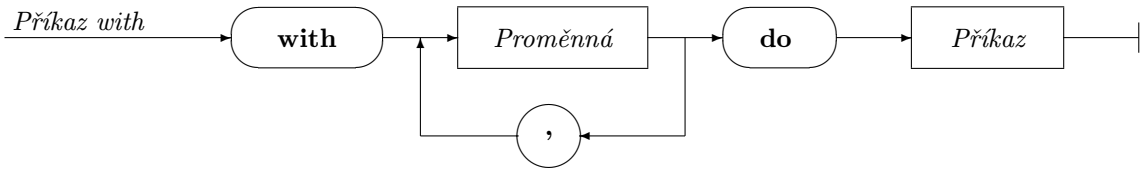
```
Program SOUCET;
  var   A,B,POM,VYSL : integer;
  begin Write('Zadej meze intervalu A,B :');
        ReadLn(A,B);
        VYSL := 0;
        for POM:=A to B do VYSL := VYSL + POM;
        Writeln('Výsledna hodnota je ',VYSL:10:2);
  end.
```

Pozn.: místo příkazu
for POM:=A **to** B **do** VYSL := VYSL + POM;
jsme mohli použít příkaz
for POM:=B **downto** A **do** VYSL := VYSL + POM;
se stejným výsledkem.

♠ **Konec příkladu**

1.5.2.5 Příkaz with

Příkaz má uplatnění při práci s proměnnými typu záznam. Umožňuje psát identifikátor složky proměnné typu záznam bez tečkové notace (příklad viz typ záznam v odstavci 1.4.2.3)



V odstavci 1.4.2.3 jsme definovali typy

```
type   RETEZ =   string[12];
       OSOBA =   record      JMENO,PRIJM : RETEZ;
                             VEK : byte;
                             HMOTNOST : real;
                 end;
```

a deklarovali proměnnou

```
var   CLOVEK : OSOBA;
```

Deklarací další proměnné, např. JMENO ve tvaru

```
JMENO : RETEZ;
```

se nezpronevěřujeme zásadě, že v Pascalu nesmí být použit stejný identifikátor k pojmenování dvou objektů, neboť iden- tifikátor jednoho datového objektu je JMENO a identifikátor jiného datového objektu je CLOVEK.JMENO .

Možné kolizní případy uvnitř příkazu **with** jsou řešeny takto:

V příkaze

```
with CLOVEK do   begin   JMENO := 'Jan';
                         HMOTNOST := 82.5;
                         Read(PRIJM,VEK)
                 end;
```

je použit identifikátor JMENO, který ovšem zpřístupňuje složku záznamu CLOVEK. Pokud bychom chtěli pracovat s proměnnou JMENO uvnitř příkazu **with**, pak je třeba použít tečkové notace, ve které jako první identifikátor uvedeme identifikátor programu.

Je-li hlavička našeho programu ve tvaru **Program** TEST; pak ve výše uvedeném příkaze **with** píšeme TEST.JMENO:='KAMIL'