# Programovanie a algoritmy

**Gabriela Andrejková**

Ústav informatiky

Prírodovedecká fakulta

Univerzita P. J. Šafárika, Košice

e-mail: andrejk@kosice.upjs.sk

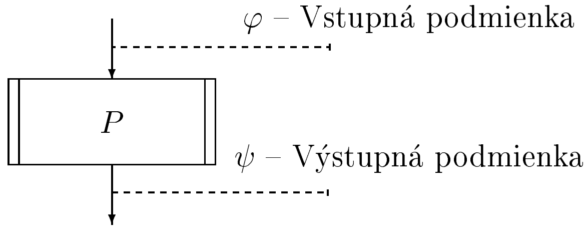
October 2, 2002

# Obsah

### 0.0.1 Jazyk vývojových diagramov

Na popis algoritmov je možné použiť graficky názorný jazyk, ktorý nazývame *jazyk vývojových diagramov.* Tento jazyk je použiteľný aj pri dôkaze správnosti algoritmov, čo je tiež veľmi názorné.

Skutočnosť (tvrdenie), že príkaz $P$ je zápisom algoritmu riešenia problému daného vstupnou podmienkou $\varphi$ a výstupnou podmienkou $\psi$, môžme v jazyku vývojových diagramov symbolicky vyjadriť takto:




V lineárnom zápise, čiže v programe je to možné zapísať:

$$\{\varphi\}\ \mathrm{P};\ \{\psi\ \}$$

**Význam:**

Príkaz (program) $P$ je zápisom (popisu) akcie, ktorá transformuje vstupné údaje splňajúce podmienku $\varphi$ na výstupné údaje, ktoré vyhovujú výstupnej podmienke $\psi$.

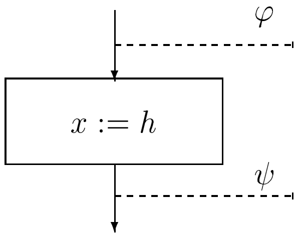
V ďalšom uvedieme spôsob zápisu jednotlivých povolených príkazov spolu s ich sémantikou. Definovať sémantiku príkazu $P$ znamená definovať túto akciu.

**Priraďovací príkaz**

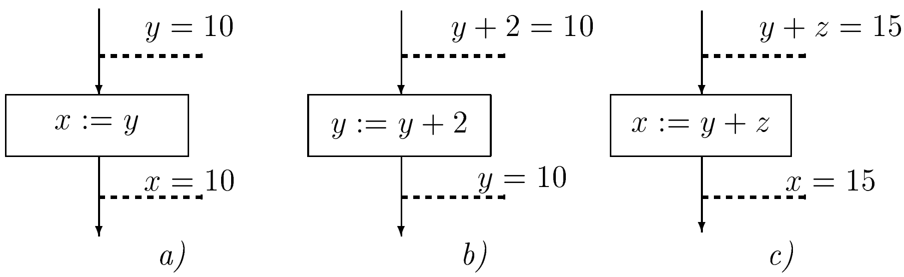
Nech je príkaz $P$ tvaru: $x := h.$

Akcia, ktorú tento príkaz popisuje sa nazýva **priradenie**: hodnota výrazu $h$ je priradená premennej $x$.

Ak podmienku $\varphi$ získame tak, že v podmienke $\psi$ substituujeme všetky výskyty $x$ výrazom $h$, potom je možné písať



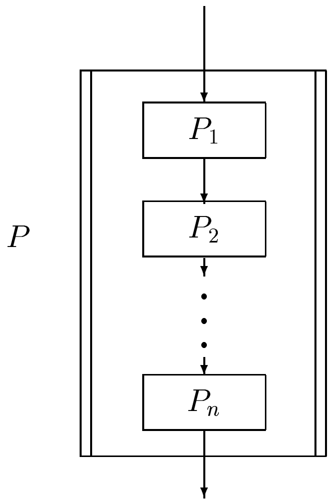

**Príklad 0.1** *V nasledujúcich troch príkladoch a) zápis znamená, že ak po vykonaní príkazu $x := y$ je hodnota $x$ rovná 10, tak pred jeho vykonaním hodnota $y$ bola rovná 10. Zápis b) znamená, že ak po vykonaní príkazu $y := y + 2$ je hodnota $y$ rovná 10, tak pred jeho vykonaním hodnota $y$ bola rovná 8. Zápis c) znamená, že ak po vykonaní príkazu $x := y + z$ je hodnota $x$ rovná 15, tak pred jeho vykonaním hodnota $y + z$ bola rovná 15.*



**Štruktúrované príkazy**

- **SEKVENCIA**

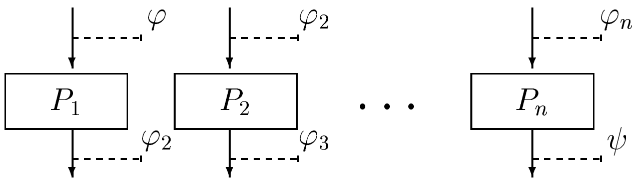
  Sekvenciu príkazov (postupnosť príkazov) je možné zapísať pomocou vývojových diagramov nasledovne:



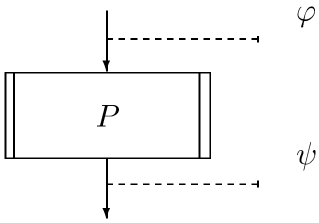

V lineárnom zápise:

$$\underbrace{P_1; P_2; \dots P_n}_{P}$$

Ak dokážeme platnosť nasledujúcich príkazov




potom pre celú sekvenciu platí

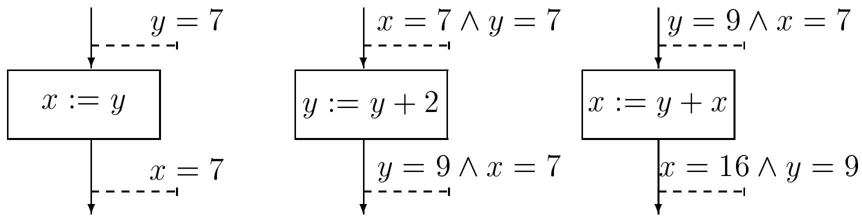



$\{\varphi\ \}$ P; $\{\psi\ \}$

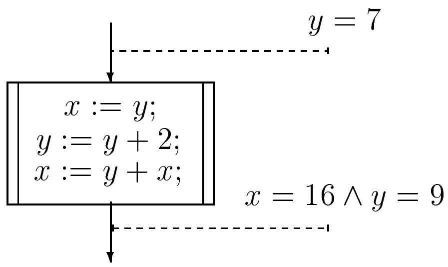
V lineárnom zápise: $\{\varphi\}P_1;\ P_2;\ \dots P_n; \{\psi\}$

Sekvencia príkazov sa dá v programovacom jazyku vyjadriť aj ako zložený príkaz.

**Príklad 0.2** *Ak je dokázaná platnosť nasledujúcej sekvencie (postupnosť) príkazov*



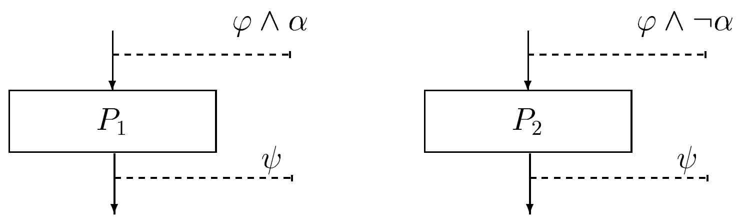

*potom môžme písať*



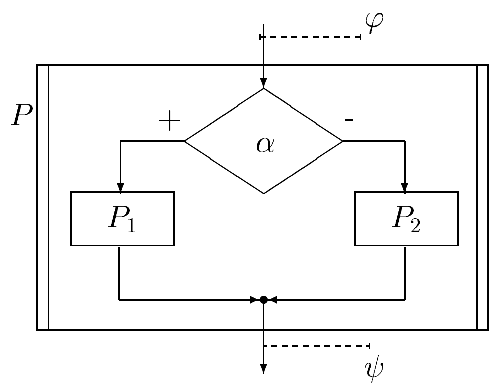

*V lineárnom zápise:* $\{y = 7\} \quad x := y;\ y := y + 2;\ x := y + x; \quad \{x = 16 \wedge y = 9\}$

- **VETVENIE**

  Príkaz vetvenia obsahuje podmienku, pri splnení ktorej dochádza k vykonaniu požadovaného príkazu a ak podmienka nie je splnená, tak je vykonaný iný príkaz alebo sa nevykoná žiadny príkaz. Preto je potrebné príkaz vetvenia skúmať ako dve sekvenčné časti, ktoré sa líšia v podmienke (podmienka a jej negácia) a vo vykonávaných príkazoch. Teda ak dokážeme platnosť nasledujúcich príkazov

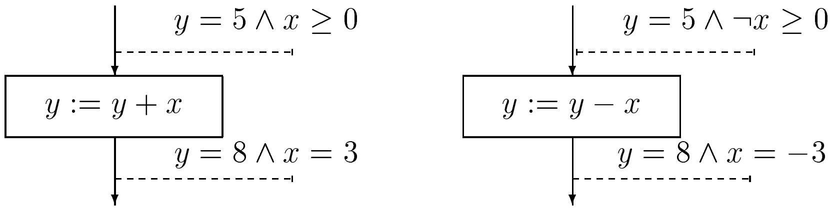



  potom je možné písať

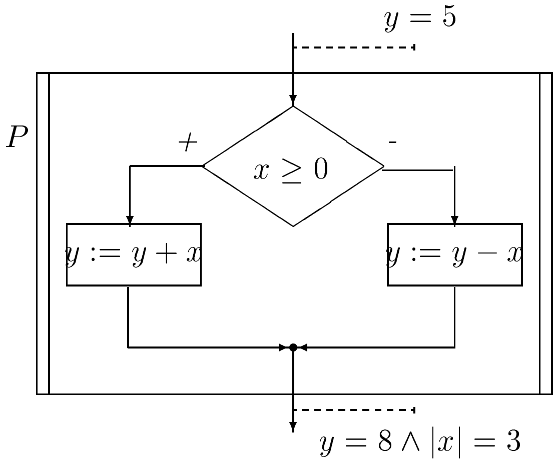



V lineárnom zápise: $\{\varphi\}$ IF $\alpha$ THEN $P_1$ ELSE $P_2$; $\{\psi\}$

**Príklad 0.3** *Ak sa nám podarí dokázať*




*potom môžme písať*




*V lineárnom zápise:*

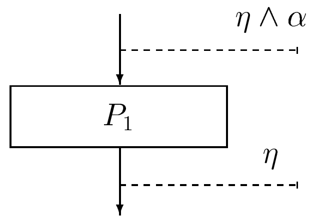
$\{y = 5\}$ *IF* $x \geq 3$ *THEN* $y := y + x$ *ELSE* $y := y - x$ $\{y = 8 \wedge |x| = 3\}$

- **CYKLUS WHILE**

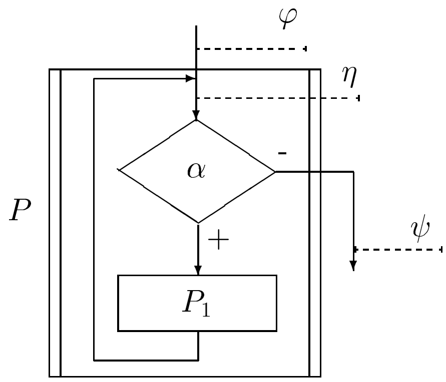
  Cykly obyčajne spôsobujú probémy, pretože sa skupina príkazov v tele cyklu opakuje. Preto cykly rozložíme na 3 časti, jedna z nich je práve takaá, ktorá obsahuje príkazy z tela cyklu a na tejto skuupine príkazov musí byť splnený invariant.

  Ak dokážeme platnosť nasledujúcich dvoch implikácií a platnosť nasledujúceho príkazu

  1. $\varphi \Longrightarrow \eta$
  2. $\eta \wedge \neg\alpha \Longrightarrow \psi$



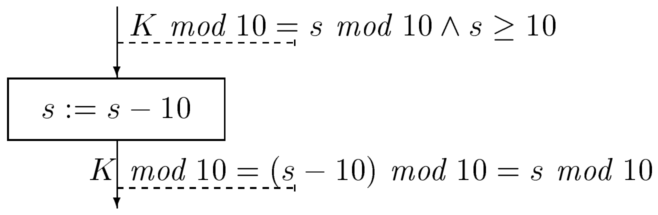

potom je možné písať



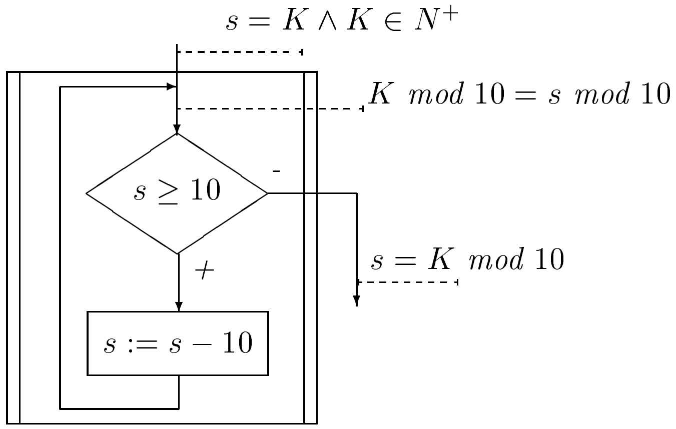

V lineárnom zápise: $\{\varphi\}$ WHILE $\alpha$ DO $P_1$;$\{\psi\}$

**Príklad 0.4** *Ak dokážeme platnosť*

1. $s = K \wedge K \in N^+ \Longrightarrow K\ mod\ 10 = s\ mod\ 10$
2. $K\ mod\ 10 = s\ mod\ 10 \wedge \neg(s \geq 10) \Longrightarrow s = K\ mod\ 10$
3.




*potom môžme písať*




*V lineárnom zápise:*

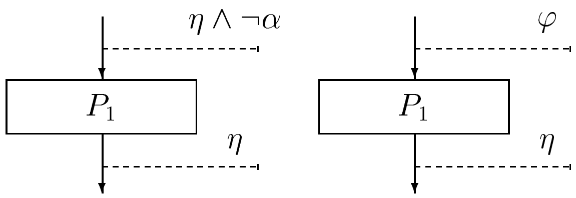
$\{s = K \wedge K \wedge K \in N^+\}$ *WHILE* $s \geq 10$ *DO* $s := s - 10;\{s = K\ mod\ 10\}$

*Je evidentné, že tento cyklus počíta zvyšok po delení 10.*

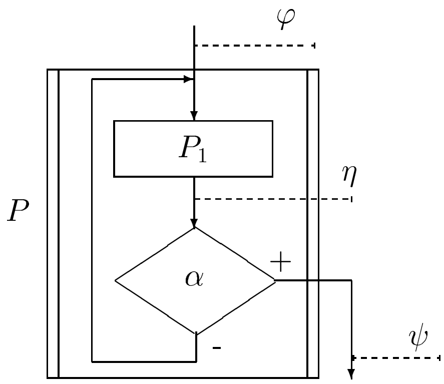
- **CYKLUS REPEAT**

  Ak dokážeme platnosť nasledujúcej implikácie a platnosť nasledujúcich dvoch príkazov

$$\eta \wedge \alpha \Longrightarrow \psi$$



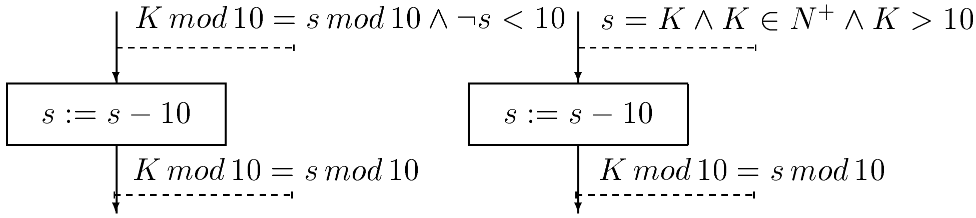

potom je možné písať



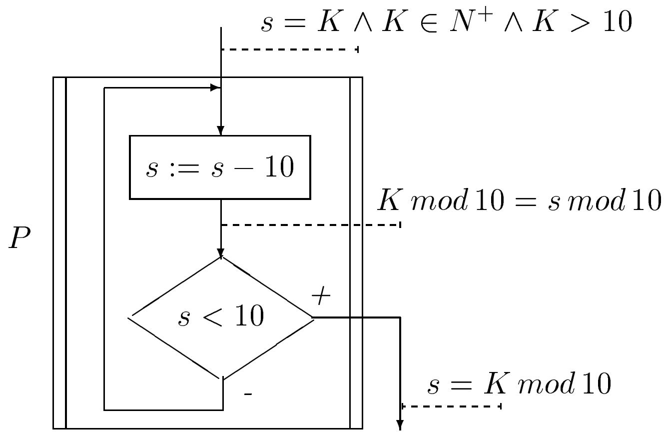

V lineárnom zápise: $\{\ \varphi\ \}$ REPEAT $P_1$;$\{\ \eta\ \}$ UNTIL $\alpha$ $\{\psi\ \}$

**Príklad 0.5** *Ak dokážeme platnosť*

$$K\,mod\,10 = s\,mod\,10 \wedge s < 10 \Longrightarrow s = K\,mod\,10$$




*potom môžme písať*

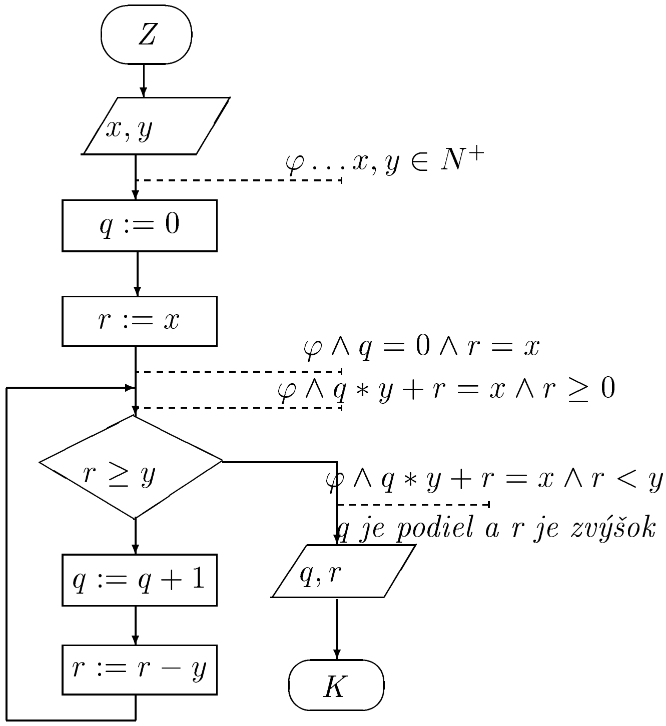



*V lineárnom zápise:*

$$\{s = K \wedge K \in N^+ \wedge K > 10\ \}\ REPEAT\ s := s - 10\ \ UNTIL\ s < 10\ \{s = Kmod10\ \}$$

V nasledujúcom príklade budú aplikované takmer všetky typy predchádzajúcich príkazov.

**Príklad 0.6** *Vytvorte algoritmus (program), ktorý pre vstupné hodnoty $x, y \in N^+$ vypočíta celočíselný podiel a zvýšok po delení.*




*V lineárnom zápise (platné podmienky sú uvedené v kučeravých zátvorkách):*

```
readln(x,y);          {nacitanie dvoch cisel zo vstupu}
q:= 0;                {pocet odcitani je doposial 0}
r:= x;                {pracovna premenna nastavena na hodnotu x}
WHILE r>= y DO        {pokial je mozne odcitat y}
BEGIN
     q:= q+1;         {zvysenie poctu odcitani o~1}
     r:= r-y;         {znizenie hodnoty r o~odcitavanu hodnotu y}
END;
writeln(q,r);         {q*y+r=x, q je podiel, r je celociselny zvysok}
```

*V lineárnom zápise s podmienkami:*

*readln(x,y);* $\{\varphi \ldots x, y \in N^+\}$

*q:= 0; r:= x;* $\{\varphi \wedge q = 0 \wedge r = x\}$

*WHILE* $r >= y$ *DO*

*BEGIN* $\{\varphi \wedge q * y + r = x \wedge r \geq 0\}$

*q:= q+1; r:= r-y;*

*END;* $\{\varphi \wedge q * y + r = x \wedge r < y\}$

*writeln(q,r); { q*y+r=x, q je podiel, r je celociselny zvysok}*

# Kapitola 1

# Programovací jazyk Pascal

Programovacie jazyky prechádzajú postupným vývojom od jazykov blízkych strojovému kódu, tzv. *strojovo-orientovaných jazykov* k jazykom blízkym, t. j. čo najviac zrozumiteľným človeku. V súčasnosti sa rozvíjajú objektovo-orientované programovacie jazyky, v ktorých je možné fyzické objekty popísať ako jeden celok, ktorý obsahuje aj atribúty objektov aj metódy, ktoré sú pre daný objekt použiteľné. Vznikajú tak nové programovacie jazyky (Java), ktoré už majú nové prvky alebo staršie spoľahlivé jazyky sú rozširované o nové prvky. Medzi tieto staršie ale veľmi spoľahlivé jazyky patrí programovací jazyk Pascal. Tento jazyk sa stal základom pre objektovo-orientovaný systém Delphi, v ktorom je možné vytvárať celkom zaujímavé a zložité aplikácie.

## 1.1 Vlastnosti vyšších programovacích jazykov

Program napísaný v strojovo-orientovanom jazyku je postupnosť elementárnych príkazov, ktoré nazývame *inštrukcie* a ktoré počítač môže priamo vykonávať (jedna inštrukcia zodpovedá jednej operácii procesora). Počítač taký program zvláda bez problémov, spracováva inštrukciu po inštrukcii mechanicky a jednoducho. Ťažšiu prácu v tomto prípade má programátor, ktorý musí tieto rozsiahle postupnosti vytvoriť a odladiť. Programátor, ktorý študuje program vytvorený niekym iným môže mať značné problémy s pochopením tohto programu.

Nezrozumiteľnosť týchto programov je hlavne v tom, že sú neštruktúrované, že sú to len postupnosti inštrukcií. Tento nedostatok je odstránený vo *vyšších programovacích jazykoch.* Z ich postupného vývoja môžme vidieť, že boli hľadané vhodné *operačné, údajové a riadiace štruktúry*, ktoré by pomohli zvýšiť zrozumiteľnosť programov pre človeka. V súvislosti s týmto vývojom programovacích jazykov vyššieho typu je potrebné vyvíjať tiež programy, ktoré sa nazývajú *prekladače* a ktoré umožňujú preklad programu z vyššieho programovacieho jazyka do strojového kódu počítača.

Základné rysy vyšších programovacích jazykov môžme zhrnúť v nasledujúcich bodoch:

1. **Typy údajov**

   Každý vyšší programovací jazyk má základné typy údajov, ktoré určujú obory hodnôt premenných a prípustné operácie nad týmito množinami. Najbežnejšie typy údajov sú, napríklad, typ celých čísel, typ reálnych čísel, typ logických hodnôt a typ znakov. Tieto typy sú tiež nazývané *jednoduché typy.* Pomocou jednoduchých typov je možné vytvárať *štruktúrované údajové typy* podľa predpísaných pravidiel. Medzi štruktúrované typy patrí predovšetkým typ pole, záznam, trieda, a ďalšie. V najnovších programovacích jazykoch sa kladie dôraz hlavne na podporu objektovo-orientovaného princípu programovania.

2. **Konštanty**

   Konštanta je údajový objekt, ktorého hodnota sa v priebehu výpočtu nemení. Konštanty sú dvoch typov, a síce v tvare *priamych konštánt - literálov* alebo v tvare *identifikátorov konštánt.* Priamou konštantou je priamy zápis hodnoty, napríklad,

   $$227 \qquad -45.234 \qquad 'Povrch \quad gule \quad je'.$$

   Identifikátory konštánt musia byť definované a je možné sa na ne odvolávať podľa ich mena. Napríklad, je možné zaviesť identifikátor konštanty POCET ako meno hodnoty 100.

3. **Premenné**

   Premenná je údajový objekt, ktorého hodnota sa môže v priebehu výpočtu meniť. *Pomenované premenné* sú najčastejšie zavedené pomocou identifikátorov. Identifikátory premenných sú v programe *deklarované* pomocou pravidiel jazyka. Pri deklarácii premennej je určené jej meno, a väčšinou aj jej typ, pomocou ktorého je určená množina prípustných hodnôt premennej a operácie, pri ktorých je možné túto premennú použiť.

   *Definíciou premennej* budeme rozumieť akciu priradenia hodnoty premennej, napríklad pomocou príkazu priradenia alebo prečítaním hodnoty zo vstupu do premennej. Nedefinovaná premenná je taká, ktorá nemá priradenú hodnotu. Vo vyšších programovacích jazykoch premenné tesne po deklarácii nie sú definované, to znamená, že pri deklarácii nemajú priradenú žiadnu hodnotu.

   Okrem pomenovaných premenných, ktoré sú zavedené deklaráciami, je možné pracovať aj s premennými, ktoré vznikajú v priebehu výpočtu. Tieto premenné budeme nazývať *dynamické premenné*, pri identifikácii ktorých sa používajú *smerníky.*

4. **Deklarácie**

   Deklarácie slúžia na zavedenie a pomenovanie premenných, procedúr funkcií, atď. v programovacích jazykoch. V tejto súvislosti je možné uvažovať o rozsahu platnosti deklarácie. Z tohto hľadiska rozlišujeme *globálne deklarácie*, ktoré sa vzťahujú na celý program, alebo *lokálne deklarácie*, ktoré sa vzťahujú len na časť programu. Túto vlastnosť majú jazyky s *blokovou štruktúrou.*

   Deklarácie sú rozlišované *dynamické a statické.* Statické deklarácie sú tie, u ktorých vlastnosti deklarovaného objektu nezávisia od hodnôt premenných, na rozdiel od dynamických, u ktorých sú niektoré vlastnosti deklarovaného objektu popísané pomocou premených. Typickou dynamickou deklaráciou je deklarácia poľa s hranicou, ktorá je určená premennou.

5. **Výrazy**

   Výraz je operačná štruktúra, ktorá určuje výpočet hodnoty. Výraz je popisom akcie aj označením hodnoty, ktorá je výsledkom akcie. Výrazy majú podobný tvar ako v matematike s tým rozdielom, že sú zapísané v lineárnej postupnosti (aj zlomky). Výrazy sa skladajú z *operandov a operátorov* a je povolené používanie zátvoriek. Operátory sa delia obyčajne na *aritmetické, logické a relačné* a zároveň určujú typ výrazu. Operandom môže byť konštanta, hodnota uložená v premennej, výsledok aplikácie funkcie.

   Vo väčšine programovacích jazykov výraz je súčasťou nejakej vyššej jazykovej konštrukcie, obyčajne príkazu, z ktorej vyplýva, čo sa má s výsledkom výrazu urobiť.

6. **Príkazy**

   Príkazy popisujú jednotlivé výpočtové akcie a vyjadrujú nadväznosti medzi príkazmi. Príkazy delíme na *jednoduché* a *štruktúrované*. Medzi jednoduché príkazy patria *priraďovací príkaz, príkaz skoku a príkaz volania procedúry.* Pomocou štruktúrovaných príkazov je možné vyjadriť postupné, podmienené alebo opakované vykonávanie niektorých iných príkazov. Sú to predovšetkým *zložený príkaz, podmienkové príkazy a príkazy cyklu.*

7. **Podprogramy**

   K algorimizácii zložitejších problémov je potrebné pristupovať systematicky, čo spočíva v tom, že problém rozložíme na jednoduchšie podproblémy a vytvárame algoritmy pre tieto jednoduchšie problémy. Toto sa vo vyšších programovacích jazykoch odrazí v tvare programu, ktorý môže byť zložený z podprogramov. Podprogram je popisom algoritmu pre podproblém. Výpočet podľa podprogramu sa môže uskutočniť na viacerých miestach programu, čo je vyjadrené zavolaním podprogramu podľa jeho mena.

   Podprogramy sa delia na *procedúry a funkcie.* Procedúra je popisom algoritmu, ktorý nejakým spôsobom transformuje údaje. Funkcia popisuje algoritmus pre výpočet jednej hodnoty, ktorá môže byť použitá ako operand vo výraze.

   Z procedúr a funkcií je možné vytvárať knižnice, ktoré môž byť použiteľné pri riešení viacerých problémov.

## 1.2 Syntaktické diagramy na popis jazyka

Syntaktický diagram (nazývaný tiež Conwayov diagram) vyjadruje väzbu medzi jednotlivými symbolmi syntaxe jazyka, určenú vytvárajúcim pravidlom. Ak je daná syntax jazyka, tak syntaktický diagram je možné vytvárať nasledujúcim spôsobom:

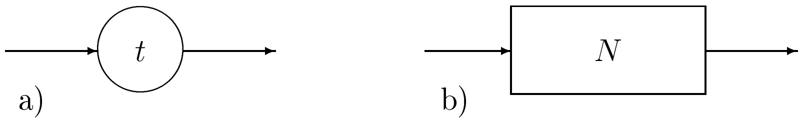
1. Každý neterminálny symbol (tj. symbol, ktorý je v jazyku ešte upravovaný) $A$, ktorý je definovaný vytvárajúcim pravidlom gramatiky

   $$A \to \alpha_1|\alpha_2|\ldots|\alpha_n$$

   je zobrazený v tvare syntaktického diagramu $G(A)$ pomocou nasledujúcich pravidiel 2.-5. Štruktúra syntaktického diagramu je určená pravou stranou vytvárajúceho pravidla.

2. Každému výskytu terminálneho alebo neterminálneho symbolu v $\alpha_i$ zodpovedá v syntaktickom diagrame vrchol ohodnotený daným symbolom. Sú zobrazované nasledovne:

   - terminálny symbol $t$ tak, ako je uvedené na Obr. 1.1,
   - neterminálny symbol $N$ tak, ako je uvedené na Obr. 1.1, pričom $\rightarrow$ vyjadruje väzbu medzi symbolmi.

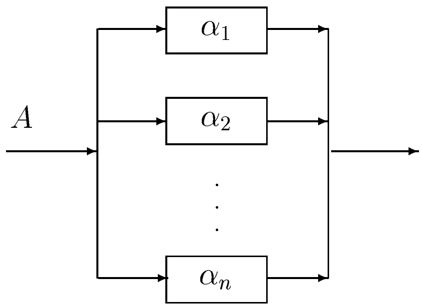



Obrázok 1.1: Syntaktický diagram a) terminálneho symbolu $t$ a b) neterminálneho symbolu $N$.

3. Vytvárajúce pravidlo tvaru

$$A \rightarrow \alpha_1|\alpha_2|\ldots|\alpha_n$$

   sa zobrazí v tvare syntaktického diagramu tak, ako je znázornené na Obr. 1.2, pričom každé $\alpha_i, i = 1, 2, \ldots, n$ je možné získať pomocou pravidiel 2.-5. To znamená, že terminálny symbol je definovaný svojím výskytom a pre zobrazenie jeho výskytu v diagrame použijeme zobrazenie podľa Obr. 1.1 a). Ak je to neterminálny symbol, zodpovedá mu vytvárajúce pravidlo, ktorým je daný symbol definovaný a podľa jeho pravej strany vytvorí preň syntaktický diagram. Syntaktický diagram k tomuto vytvárajúcemu pravidlu sa nachádza na Obr. 1.2.




Obrázok 1.2: Syntaktický diagram pre pravidlo $A \rightarrow \alpha_1|\alpha_2|\ldots|\alpha_n$.

4. Ak $\alpha_i$ má tvar

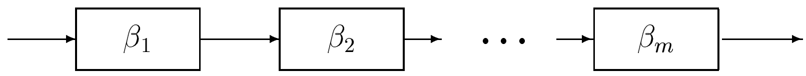
$$\alpha_i = \beta_1\beta_2\ldots\beta_m$$

   je zobrazené v tvare syntaktického diagramu na Obr. 1.3, pričom každé $\beta_i, i = 1, 2, \ldots, m$ je získané pomocou pravidiel 2.-5.

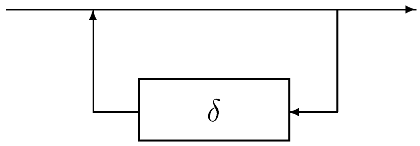
5. Ak $\alpha_i$ má tvar

$$\alpha_i = \{\delta\}$$

   je zobrazené v tvare syntaktického diagramu na Obr. 1.3, pričom $\delta$ je získané pomocou pravidiel 2.-5.



Obrázok 1.3: Syntaktický diagram pre pravidlo $\alpha_i = \beta_1\beta_2 \ldots \beta_m$.



Obrázok 1.4: Syntaktický diagram pre pravidlo tvaru $\alpha_i = \{\delta\}$.

Príklady syntaktických diagramov budú uvedené v nasledujúcich odsekoch pri popise údajových štruktúr a príkazov.