O algoritmech

Výňatek z diplomové práce

Programovací jazyk pro podporu výuky algoritmů

Univerzita Hradec Králové - Fakulta informatiky a managementu - Katedra informatiky a kvantitativních metod

duben 2006

Autor: Ing. Petr Voborník

Vedoucí diplomové práce: Doc. RNDr. Eva Milková, Ph.D.

Obsah

Glosář **1**

O algoritmech **2**

1. Podporované příkazy 2

1.1. Uvození bloku příkazů 2

1.2. Příkazy větvení 3

1.3. Příkazy cyklu 3

1.4. Jednoduché příkazy 4

1.5. Konstanty, operátory, textové řetězce a komentáře 6

1.6. Středníky 9

2. Správné formátování kódu 10

3. Ukázkové algoritmy 16

Seznam použité literatury **21**

Glosář

Matice Dvojrozměrné pole.

Nadpříkaz Blokový příkaz algoritmu (příkaz cyklu či větvení), který pod sebou obsahuje jeden či více (blok) *podpříkazů*, pro které je pak tento nadpříkazem.

Originální kód Kód algoritmu napsaný v módu originálního kódu, kterým se rozumí kód s klíčovými slovy převzatými z jazyka Pascal. Jejich znění je neměnné.

Panel nástrojů Lišta pod hlavním menu, obsahující ikony, které zastupují příkazy z menu. Tyto panely nástrojů je možné libovolně přesouvat v rámci okrajů hlavního okna, tažením je uvolňovat do *plovoucích oken*, skrývat je či objevovat, měnit jejich obsah a vytvářet nové.

Podpříkaz Jeden či více (blok) příkazů, které jsou součástí cyklu či větve podmínky algoritmu, jejíž příkaz je pro tyto *nadpříkazem*.

Pole Strukturovaná proměnná, obsahující více hodnot. Pole je definováno svým názvem a hodnoty v něm pak indexem.

Posloupnost Jednorozměrné pole.

Pseudokód Kód algoritmu napsaný v módu pseudokódu, kterým se rozumí algoritmus s klíčovými slovy, která si uživatel může volitelně měnit.

Rezervovaná slova jsou slova vyhrazená programem pro definici algoritmů a nemohou tedy sloužit jako názvy proměnných a polí. Patří mezi ně především klíčová slova a konstanty.

Syntaxe Formální správnost zapsání algoritmu, na úrovni slov a znaků. Syntakticky správný algoritmus ještě nemusí být algoritmem správně funkčním.

Výraz Aritmetický výraz, který lze početně vyhodnotit a získat tak jednu jedinou hodnotu.

O algoritmech

Program Algoritmy, jak již ostatně jeho název napovídá, je primárně určen především k pohodlnému psaní, kontrolování, spouštění a krokování algoritmů. Slouží tedy jako učební pomůcka k problematice algoritmů.

V této kapitole se zaměříme právě na ně. Probereme zde každý podporovaný příkaz, jaké musí mít náležitosti, jaká je jeho podstata a správný formální zápis. Ukázky algoritmů zde budou uvedeny ve znění výchozí verze českého pseudokódu.

1. Podporované příkazy

Na začátku je třeba upozornit, že program Algoritmy je *case insensitive*, tedy nerozlišuje malá a velká písmena, načež proměnná s názvem **proměnná** je považována za stejnou jako například **ProMěNNá**.

Nejprve si probereme jednotlivé příkazy podporované programem Algoritmy.

1.1. Uvození bloku příkazů

Blokem algoritmu se rozumí jeden či více libovolných příkazů jdoucích za sebou v daném pořadí, kteréžto jsou vymezeny klíčovými slovy **začátek** (před prvním příkazem bloku) a **konec** (po posledním příkazu bloku).

```
začátek
  příkaz;
  ...
  příkaz;
konec
```

Do bloku je vždy třeba uzavřít celý algoritmus. Jinými slovy prvním slovem algoritmu musí být **začátek** a posledním **konec**. Do bloku se také uzavírají skupiny příkazů u větvení a cyklů (zpravidla pouze v případě, že jich je více než jeden, ale není to podmínkou). Tedy následuje-li pod těmito typy příkazů slovo **začátek**, platí jejich funkce (podmínka větvení či multiplicita cyklů) na celý tento blok až po slovo **konec**.

Počet všech začátků v algoritmu musí odpovídat počtu všech konců!

Klíčová slova **začátek** a **konec** sama o sobě nejsou příkazy, jelikož nevykonávají žádnou činnost, mají pouze vymezovací funkci.

1.2. Příkazy větvení

Větvením se rozumí omezení vykonání určitého příkazu (či bloku příkazů) jistou podmínkou. V programu, v souladu s výukou algoritmizace na FIM UHK rozlišujeme dva typy příkazu větvení: úplné a neúplné. Příkazy podmínkou uvozené jsou tedy provedeny pouze v případě jejího splnění (platnosti výrazu s výslednou logickou hodnotou *true* - pravda).

Příkazem neúplného větvení je zde příkaz začínající klíčovým slovem **jestliže**. Za tímto klíčovým slovem musí následovat podmínka (např. **x > 3**), poté klíčové slovo **pak**, a poté již příkaz, který bude v případě splnění podmínky vykonán a v případě opačném přeskočen. Je-li těchto příkazů více, je třeba je uvést do bloku.

Příkaz větvení tedy může vypadat takto:

```
jestliže x > 3 pak
  příkaz;
```

Podmínka může být samozřejmě libovolně složitá, nezbytné však je, aby jejím výsledkem byla logická hodnota reprezentovaná buď *true* / *false* (získaná např. porovnáním dvou hodnot) nebo číselně (0 pak zastupuje *false* a vše ostatní je *true*).

Úplným větvením je možnost použití klíčového slova **jinak**. To může následovat pouze těsně za příkazem (či za slovem **konec** v případě bloku), jenž je omezen podmínkou větvení a za ním musí následovat příkaz další, který bude vykonán pouze v případě, že podmínka splněna nebude. Může to tedy vypadat například takto:

```
jestliže x > y pak
  vetší := x
jinak
  větší := y;
```

1.3. Příkazy cyklu

Rozlišujeme dva způsoby zápisu příkazu cyklu: příkazy začínající klíčovým slovem **dokud** a příkazy začínající klíčovým slovem **pro**, přičemž příkaz **pro** je speciálním případem příkazu **dokud**, ale v určitých případech je výhodnější a úspornější používat spíše příkaz **pro**.

Pro

Příkaz **pro** je určen pro případy, kdy chceme použít nějaké příkazy ve známém počtu opakování (např. 10x). Parametrem za klíčovým slovem **pro** je proměnná, v jejíž hodnotě se bude počítat kolikáté je to opakování, za klíčovým slovem **od** následuje počáteční hodnota této proměnné a za klíčovým slovem **do** hodnota, kterou když proměnná přesáhne, tak cyklus končí. Příkaz je pak uzavřen klíčovým slovem **opakuj**, za nímž již následuje rovnou příkaz (či blok příkazů), jež má být opakován. V každém zopakování je tedy k proměnné automaticky přičteno +1. Pro deset opakování by tedy tento příkaz mohl vypadat takto:

```
pro i od 1 do 10 opakuj
  příkaz;
```

Zde **i** je proměnná, z jejíž hodnoty se dá u příkazů v cyklu kdykoli zjistit, kolikáté je toto opakování. Obě hodnoty (od a do) mohou být samozřejmě definovány jinou proměnnou či výrazem, přičemž pokud je hodnota **od** větší než **do**, cyklus není vykonán ani jednou. Cyklu **pro** se využívá nejvíce při procházení hodnot polí.

Dokud

Příkaz **dokud** je obecnější než příkaz **pro**, a to především co do podmínky pro ukončení cyklu. Příkaz **dokud** totiž má za parametr logický výraz, který se před každým zopakováním takto uvozených příkazů znovu otestuje a pouze v případě, že jeho výsledkem je logická pravda (*true*) je cyklus znovu zopakován, v opačném případě je ukončen. Za touto podmínkou následuje klíčové slovo **opakuj** a za ní již příkaz (či blok příkazů), jež bude opakován. Cyklus by tedy mohl vypadat například takto:

```
dokud x > 0 opakuj
  x := x - 2;
```

Je zde třeba dbát na to, aby nedošlo k tomu, že podmínka tohoto cyklu bude splnitelná vždy, neboť v tom případě by cyklus nebyl nikdy ukončen a probíhal tak donekonečna (např. při 1 = 1).

1.4. Jednoduché příkazy

Jednoduché funkční příkazy jsou takové, které jsou vykonány pouze samy o sobě a neobsahují již žádné další podpříkazy, za kterézto se však nepovažuje

vyhodnocování výrazů. Vezměme si každý z nich podporovaný programem Algoritmy jednotlivě.

Čti

Tento příkaz umožňuje uživateli definovat hodnoty některých proměnných v průběhu běžícího algoritmu. Tímto příkazem definované proměnné se nazývají vstupy algoritmu, jelikož za předpokladu, že by byl tento algoritmus samostatně spustitelným programem, by pak tento příkaz byl jedinou možností, jak získávat vstupy zvenčí.

Příkaz **čti** má v závorkách pouze jediný parametr a tím je proměnná, do níž bude přečtená hodnota zadána. Zápis příkazu vypadá takto:

```
čti(x);
```

Tento příkaz se tedy zeptá uživatele na hodnotu, kterou po jejím zadání přiřadí do proměnné **x**.

Napiš

Příkaz **napiš** je naopak jedinou možností, jak realizovat výstupy z programu (sledování aktuálních hodnot proměnných, které program Algoritmy umožňuje nelze považovat za plnohodnotný výstup). Díky němu lze vypsat libovolný text, hodnotu kterékoli proměnné nebo pole či přímo výrazu.

Příkaz **napiš** má tedy v závorkách jeden či více parametrů oddělených čárkami. Jeho zápis pak může vypadat například takto:

```
napiš('Násobkem hodnot', x, ' a ', y, ' je hodnota ', x * y, '.');
```

Tento příkaz tedy v případě že hodnota **x** je 3 a hodnota **y** je 4 vypíše následující text: **Násobkem hodnot 3 a 4 je 12.**

Přiřazení

Příkaz přiřazení není uvozen žádným klíčovým slovem ale obsahuje přiřazující dvojznak „:=" (dvojtečka a rovná se). Prvním slovem je však vždy název proměnné, do které bude hodnota za tímto dvojznakem přiřazena. Následuje := a název proměnné, pole, hodnota či výraz, jehož výsledek bude proměnné před dvojznakem přirazen.

Zápis přiřazení pak může vypadat například jako jeden z následujících příkladů:

```
x := 1;
y := x + 1;
z := 10 * (2 * x) ^ y - 74 * (x - y / 2);
```

Po provedení těchto tří příkazů tedy bude platit, že **x = 1**, **y = 2** a **z = 40**.

Při práci s kteroukoli proměnnou je třeba dbát na to, aby její hodnota byla v některém z předchozích příkazů definována (přiřazením nebo příkazem **čti**). V opačném případě totiž dojde k chybě a běh algoritmu je automatiky ukončen.

1.5. Konstanty, operátory, textové řetězce a komentáře

Zbývá ještě probrat slova, která nejsou přímo klíčovými slovy, ale pouze slovy rezervovanými. To znamená, že je, stejně jako klíčová, nelze použít pro názvy proměnných, neboť jejich využití je především při definici jiných proměnných a vyhodnocování výrazů.

Konstanty

Konstanty jsou zde rezervovaná slova, mající stejný význam jako hodnota, kterou zastupují. Program Algoritmy zná tato:

- **true** - Logická pravda (shodná například s výrazem **1 = 1**).
- **false** - Logická nepravda (shodná například s výrazem **1 = 2**).
- **pi** - Ludolfovo číslo (pí, 3.14159265358979).
- **exp** - Eulerovo číslo (e, 2.71828182845905).
- **random** - Není konstantou, ale pokaždé jiným náhodným reálným číslem mezi 0 a 1 z rovnoměrného rozdělení R(0,1).

Operátory

Dalším případem rezervovaných slov jsou operátory. Většina z nich však ani nejsou slova, ale znaky, jejichž význam je na první pohled zřejmý. Přesto si tu však každý z nich pro jistotu osvětlíme. Pořadí operátorů je řazeno dle jejich priority.

- **not** – Negace logického výrazu. Je-li hodnota následující za **not** logická pravda, bude výsledkem nepravda a naopak. Např. platí, že **(not true) = false**.
- **^** - Mocnina. Hodnota před tímto operátorem je umocněna mocnitelem, který následuje za tímto operátorem. Např. platí, že **2 ^ 3 = 8**.
- ***** - Násobení. Vynásobí hodnoty před a za tímto operátorem. Např. platí, že **2 * 3 = 6**.

- **/** - Dělení. Vydělí hodnotu před tímto operátorem hodnotou za ní. Např. platí, že **6 / 3 = 2**.
- **div** - Celočíselné dělení. Vrátí kolikrát se hodnota za tímto operátorem celá vejde do hodnoty před ním. Např. platí, že **7 div 3 = 2**.
- **mod** - Zbytek po dělení. Vrátí zbytek po dělení hodnoty před tímto operátorem hodnotou za ním. Např. platí, že **7 mod 3 = 1**.
- **-** - Odečítání. Odečte od hodnoty před tímto operátorem hodnotu za ním. Např. platí, že **3 – 2 = 1**.
- **+** - Sčítání. Vrátí součet hodnot před a za tímto operátorem. Např. platí, že **2 + 3 = 5**.
- **<=** - Menší nebo rovno. Vrátí logickou hodnotu `true` v případě, že hodnota před tímto operátorem je menší nebo rovna než ta za ním. V opačném případě je výsledkem `false`. Např. platí, že **(4 <= 3) = false**.
- **>=** - Větší nebo rovno. Vrátí logickou hodnotu `true` v případě, že hodnota před tímto operátorem je větší nebo rovna než ta za ním. V opačném případě je výsledkem `false`. Např. platí, že **(4 >= 3) = true**.
- **<** - Menší než. Vrátí logickou hodnotu `true` v případě, že hodnota před tímto operátorem je menší než ta za ním. V opačném případě je výsledkem `false`. Např. platí, že **(4 < 3) = false**.
- **>** - Větší než. Vrátí logickou hodnotu `true` v případě, že hodnota před tímto operátorem je větší než ta za ním. V opačném případě je výsledkem `false`. Např. platí, že **(4 > 3) = true**.
- **=** - Porovnání dvou hodnot. Vrátí logickou hodnotu `true` pouze v případě, že hodnoty před a za tímto operátorem jsou naprosto stejné, jinak je výsledkem `false`. Např. platí, že **4 = 4**.
- **<>** - Nerovná se. Vrátí logickou hodnotu `true` pouze v případě, že hodnoty před tímto operátorem není stejná jako hodnota za ním, jinak je výsledkem `false`. Např. platí, že **4 <> 3**.
- **and** - A zároveň. Vrátí logickou hodnotu `true` pouze v případě, že logické hodnoty před i za tímto operátorem jsou `true`. Pokud je jen jedna z nich či obě `false`, výsledkem je `false`. Např. platí že **(true and false) = false**.

- **or** - Nebo. Vrátí logickou hodnotu `true` pouze v případě, že alespoň jedna logická hodnota (případně obě) před či za tímto operátorem je `true`. Např. platí že **(true or false) = true**.

Ne všechny operátory však lze použít na všechny typy hodnot. Program Algoritmy rozlišuje totiž tři typy hodnot: logické, číselné a textové. Pro každý z nich lze použít pouze tyto operátory:

- logické hodnoty: not, =, <>, and, or
- číselné hodnoty: not, ^, *, /, div, mod, -, +, <=, >=, <, >, =, <>, and, or
- textové řetězce: +, =, <>

Operátory `not`, `and` a `or` lze u číselných hodnot použít díky tomu, že 0 je brána jako `false` a všechny ostatní číselné hodnoty jako `true`. Výsledkem však již bude logická hodnota `true` či `false` (např. **not 3 = false**).

Textové řetězce

Textový řetězec je typ hodnoty. Takovéto hodnoty mohou být přiřazovány proměnným a vykonávány s nimi i některé operace (sčítání a přímé porovnávání).

Veškeré textové řetězce přímo použité v algoritmu musí být umístěny mezi apostrofy či uvozovky (např. **'text'** a **"text2"** jsou textové řetězce). Vše co bude umístěno mezi jedním z těchto párů znaků je považováno za textový řetězec a jako s takovým je s ním i nakládáno. Dva druhy vymezení textových řetězců jsou použity mimo jiné i pro to, aby bylo možné do řetězců zahrnout i oba tyto znaky (např. "McDonald's" nebo 'Řekl: "Ahoj!"').

Jsou-li sčítány dvě hodnoty tohoto typu operátorem +, nejedná se o sčítání v pravém slova smyslu, nýbrž o sloučení těchto dvou textů do jednoho (např. **'Ahoj' + 'Karle' = 'AhojKarle'**). Sčítáme-li textový řetězec a číselnou hodnotu, bude tato hodnota převedena také na textový řetězec, což bude i typem výsledku takovéhoto součtu (např. **250 + ' Kč' = '250 Kč'**).

V případě porovnávání je srovnáván každý jednotlivý znak obou textových řetězců (při *case sensitive*, tedy rozlišování malých a velkých znaků jako různých) a výsledkem je logická hodnota určující, jsou či nejsou-li tyto dva řetězce stejné (např. výsledkem výrazu **'honza' = 'jan'** je **false**, stejně jako u výrazu **'honza' = 'Honza'**). Při srovnávání číselné a textové hodnoty je opět ta číselná nejprve převedena na textovou a až poté dojde k porovnání (tedy např. **platí** že **2 = '2'** ale **neplatí** že **2 = 'dvě'**). To samé, ale s opačnými výsledky platí i pro operátor **<>**.

Komentáře

Je velmi dobrým zvykem, ne-li přímo nepsaným pravidlem, do kódu algoritmů vkládat také komentáře. Jedná se o vymezené části obsahující libovolný text, který nemá žádný vliv na běh algoritmu. Komentáře jsou však velmi užitečné pro celkové pochopení algoritmu, ať třetí osobou, tak samotným autorem algoritmu, vrátí-li se k němu po čase.

Komentáře mohou být v programu Algoritmy dvojího typu: blokové a řádkové. Blokové jsou vymezeny složenými závorkami **{** a **}**. Všechno mezi těmito závorkami, včetně jich samotných, je považováno za komentář a jako takové při zpracování kódu zcela ignorováno. Takto vymezený blok komentáře může být jak přes několik řádků, tak i pouze uprostřed řádku s jinak platným příkazem, viz následující ukázka:

```
{
  Toto je komentář
  na více řádcích.
}
a := 3 * 4 { + 5};
```

V této ukázce by hodnota proměnné a byla 12, neboť **{+ 5}** je pouhý komentář.

Řádkové komentáře na rozdíl od blokových mají pouze dvojznak který je začíná, neboť jejich konec je vymezen koncem řádku. Tímto dvojznakem jsou dvě lomítka za sebou (**//**). Vše za nimi následující až do konce daného řádku je také považováno za komentář a jako takové při zpracování algoritmu ignorováno.

```
a := 3 * 4; // a bude od teď 12
```

1.6. Středníky

Nyní si ještě vysvětlíme problematiku psaní či nepsaní středníků v algoritmu. Jsou totiž jejich důležitou součástí a pokud nejsou uvedeny správně, neprojde algoritmus ani základní kontrolou. Středníky v algoritmech určují konec příkazu a v zásadě platí, že by měly následovat za každým z nich. Existují ovšem i výjimky od tohoto pravidla, takže to vezměme pěkně popořádku.

Za jednoduchými příkazy (**čti**, **napiš** a přiřazení) se středníky píší vždy, pokud za nimi nenásleduje klíčové slovo **jinak**. Za příkazy cyklu (tedy za klíčovým slovem **opakuj**) a větvení (za klíčovým slovem **pak**) se středníky nepíší nikdy, neboť za nimi následuje další příkaz či klíčové slovo **začátek**. Za klíčovým slovem

začátek se středník také nepíše, neboť není příkazem. Za klíčovým slovem **konec** se naopak středník píše vždy (nenásleduje-li **jinak**), protože jím v podstatě končí příkaz, který stojí nad tímto blokem (cyklus či větvení). Za posledním klíčovým slovem **konec**, jež zakončuje celý algoritmus se nepíše buď nic nebo tečka.

V podstatě tedy platí, že středník se píše za každým příkazem, nenásleduje-li klíčové slovo **jinak**, přičemž „nadpříkazy" (cyklus a větvení) končí až za svým posledním „podpříkazem". Před (ani za) **jinak** se středník nepíše nikdy, protože příkaz větvení, jehož je **jinak** pokračováním, končí až za blokem této alternativní podmínky.

Následuje názorný příklad:

```
{ Největší ze tří }
začátek
  // Načtení tří hodnot
  čti(x);
  čti(y);
  čti(z);
  // Porovnání
  jestliže x > y pak
    jestliže x > z pak
      největší := x
    jinak
      největší := z
  jinak
    jestliže y > z pak
      největší := y
    jinak
      největší := z;
  // Vypsání výsledku
  napiš('Největší zadané číslo je ', největší);
konec
```

2. Správné formátování kódu

Formátování kódu algoritmu je v podstatě pouze jeho formální úprava, která nemá na jeho běh žádný vliv a díky tomu si každý může psát algoritmy takovým stylem, jaký mu nejlépe vyhovuje. Nicméně vhodné formátování kódu mnohdy až několikanásobně zvyšuje jeho přehlednost a dobře naformátované algoritmy je poté celkem snadné „číst", studovat jejich podstatu, vyhledávat v nich chyby, opravovat

je a učit se z nich. Z těchto důvodů si zde nastíníme jeden z (v dlouhodobé praxi prověřených) příhodný způsob, jak algoritmy vhodně formátovat.

Základním pravidlem správného formátování je, že každý příkaz by měl být na zvláštním řádku. To zvyšuje nejen přehlednost, ale také jedině v tomto případě je možné algoritmus efektivně krokovat.

Správně

```
začátek
  příkaz;
  příkaz;
  příkaz;
konec
```

Nesprávně

```
začátek
  příkaz;  příkaz;  příkaz;
konec
```

Dalším podstatným pravidlem je odsazování podpříkazů. Odsazováním se rozumí přidání mezer na začátek každého řádku s příkazem tak, aby tento začínal až na příslušné úrovni. Tato úroveň je určena tím, kolik nadřazených příkazů (mohou jimi být pouze příkazy cyklu či větvení) je nad tím právě řešených. Počet mezer je pak násobkem čísla této úrovně a odsazovací jednotky, která je v programu Algoritmy stanovena na dvě mezery. Hlavní blok programu, který sám začíná na nulté úrovní, se do toho také počítá.

Správně

```
začátek
  příkaz;
  jestliže x > 3 pak
    příkaz
  jinak
    příkaz;
konec
```

Nesprávně

```
začátek
příkaz;
jestliže x > 3 pak
příkaz
jinak
příkaz;
konec
```

Klíčová slova uvozující blok (`začátek` a `konec`) mají úroveň stejnou jako příkaz nadřazený tomuto bloku.

Správně

```
začátek
  pro i od 1 do 5 opakuj
  začátek
    příkaz;
    příkaz;
  konec;
konec
```

Nesprávně

```
začátek
  pro i od 1 do 5 opakuj
    začátek
    příkaz;
    příkaz;
    konec;
konec
```

Pokud je nějaký řádek příliš dlouhý, ve vhodném místě jej zalomíme a pokračujeme v něm na řádku dalším, který bude odsazen o dvě jednotky (čtyři mezery) více, než je příkaz, jehož je toto pokračováním.

Správně

```
začátek
  x := ((3 * 4) - 2 + 1) *
       (10 + 1) * 12 / 2 ;
konec
```

Nesprávně

```
začátek
  x := ((3 * 4) - 2 + 1) *
  (10 + 1) * 12 / 2 ;
konec
```

U zalamování řádků je však více než kde jinde vhodné neřídit se zcela tímto pravidlem a přizpůsobit si úroveň odsazení tak, aby nějakým způsobem logicky začínala na co nejvhodnějším místě, usnadňujícím orientaci v tomto dlouhém příkazu. Například aby začínala na úrovni závorky, v níž se nachází.

```
začátek
  x := ((((3 + 4) * (7 + 2)) /
          (((5 + 2) * (4 - 3)) * 2)) ^
         (3 + 2));
konec
```

U hodně dlouhých příkazů se však doporučuje si je rozdělit na více kratších a ty pak najednou vyhodnotit.

Dobrým zvykem je také vkládání mezer mezi operátory ve výrazech. V některých případech je samozřejmě vhodnější tyto mezery vynechat (například u mocnění), ale v zásadě je spíše lepší tyto mezery dodržovat. V každém případě to však platí u dvojznaku přiřazení „:=“.

Správně

```
začátek
  čti(d);
  S := pi * d^2 / 4;
konec
```

Nesprávně

```
začátek
  čti(d);
  S:=pi*d^2/4;
konec
```

U čárek v příkazu `napiš` a v souřadnicích pole platí to samé, jako při psaní obyčejného textu, tedy že před ní mezera není a za ní naopak ano.

Správně

```
začátek
  pole[1, 1] := 1;
  napiš('První je ', pole[1, 1]);
konec
```

Nesprávně

```
začátek
  pole[1 ,1] := 1;
  napiš('První je ',pole[1,1]);
konec
```

Co se týče komentářů, tak ty je vhodné psát několika různými způsoby. Jedním z nich je při nadpisu části algoritmu odsadit jej na stejnou úroveň jako má následující příkaz, který je prvním ze série příkazů tímto komentářem popisovaných.

```
začátek
  // Načtení hodnot
  čti(a);
  čti(b);
  // Porovnání
  jestliže a > b pak
    větší := a
  jinak
    větší := b;
  // Vypsání výsledku
  napiš('Větší je ', větší);
konec
```

Pokud chceme komentovat pouze jeden příkaz je vhodnější tento komentář napsat za něj na tentýž řádek. Toto je zvlášť vhodné, komentujeme-li takto několik řádků za sebou, každý samostatně. V tom případě, je-li to z kapacitních důvodů možné, je vhodné i tyto komentáře zarovnat pod sebe.

```
začátek
  čti(r);                          // Načteme poloměr
  o := 2 * pi * r;                 // Vypočítáme obvod
  S := pi * r ^ 2;                 // Vypočítáme obsah
  napiš('o = ', o, ', S = ', S);   // Vypíšeme výsledky
konec
```

Je také dobrým zvykem ještě před samotný hlavní blok algoritmu na první řádek (či více) vložit blokový komentář s textem stručně charakterizujícím každý algoritmus. Díky tomu je pak možné si jeho význam přečíst zde a ne se po něm složitě pídit ve struktuře samotného algoritmu. Je tu také prostor pro jméno autora algoritmu, a to buď přímo v jeho úvodním popisu, nebo dole za finálním slovem konec.

```
{ Součet dvou čísel. }
začátek
  čti(a);
  čti(b);
  c := a + b;
  napiš('Součtem čísel ', a, ' a ', b, ' je číslo ', c);
konec
{ Autor: Jan Novák }
```

Výjimečné postavení má někdy klíčové slovo **jinak**. V základu na něj lze samozřejmě aplikovat všechna výše uvedená pravidla, nicméně existuje několik celkem častých případů, kdy je vhodné k němu přistupovat individuálně.

Jedním z těchto případů je vícenásobné větvení, kdy je například potřeba jednu hodnotu otestovat na několik (více než dvě) možností, přičemž jen jedna z nich může platit. Pokud by se tedy tyto příkazy větvení odsazovali klasicky stále dál a dál, vznikla by velmi nepřehledná a z okrajů obrazovky doprava mizející „jitrnice“. Proto je vhodné vždy rovnou za slovo **jinak** napsat na tentýž řádek **jestliže** s další podmínkou.

```
začátek
  čti(x);
  jestliže x > 0 pak
    jestliže x < 10 pak
      napiš('Zadané číslo má jednu cifru.')
    jinak jestliže x < 100 pak
      napiš('Zadané číslo má dvě cifry.')
    jinak jestliže x < 1000 pak
      napiš('Zadané číslo má tři cifry.')
    jinak jestliže x < 10000 pak
      napiš('Zadané číslo má čtyři cifry.');
konec
```

Další specialitou klíčového slova **jestliže** je možnost ušetření jednoho až dvou řádků, což u delší algoritmů může být přínosné. Vejdou-li se totiž díky tomu celé na jednu obrazovku či papír, stávají se lépe čitelnými. Konkrétně jde o případ, kdy je před klíčovým slovem **jinak** slovo **konec**, nebo za ním následuje slovo **začátek**, či obojí. V těchto případech je možné tato tři slova (případně jen dvě) napsat na jediný řádek.

```
začátek
  čti(x);
  jestliže x = 0 pak
  začátek
    napiš('Zadané číslo je nula.');
    sign := 0;
  konec jinak
    jestliže x > 0 pak
    začátek
      napiš('Zadané číslo je kladné.');
      sign := 1;
    konec jinak začátek
      napiš('Zadané číslo je záporné.');
      sign := -1;
    konec;
konec
```

Program Algoritmy disponuje funkcí, která dokáže na aktuální kód algoritmu aplikovat základní pravidla. Je však určitě lepší rovnou psát algoritmus dle těchto pravidel, což umožňuje i aplikovat ony drobné výjimky, jež program identifikovat nedokáže.

3. Ukázkové algoritmy

V této kapitole je uvedeno pouze několik hotových algoritmů, jakožto ukázka syntaktické i formální správnosti.

Průměr ze zadaných čísel

```
začátek
  suma := 0;
  počet := 0;
  x := 1; // Aby to poprvé prošlo podmínkou cyklu dokud
  dokud x <> 0 opakuj
  začátek
    x := 0; // Výchozí hodnota, aby pro konec stačilo jen Enter
    čti(x);
    suma := suma + x;
    počet := počet + 1;
  konec;
  počet := počet - 1;      // Poslední byla 0, která se nepočítá
  průměr := suma / počet;  // Výpočet průměru
  napiš('Průměr ze zadaných čísel je ', průměr);
konec.
```

Fibonacciho generátor náhodných čísel

```
začátek
  počet := 15; // Požadovaný počet hodnot
  M := 100;    // Rozsah generovaných hodnot
  // Libovolně zvolené první dvě hodnoty
  x[1] := 32;
  x[2] := 63;
  // Vygenerování "náhodných" čísel od 0 do M-1
  pro i od 3 do počet opakuj
    x[i] := (x[i-1] + x[i-2]) mod M;
konec.
```

Lehmerův generátor pseudonáhodných čísel

```
začátek
  počet := 15;  // Požadovaný počet hodnot
  // Parametry generátoru (nastavena volba ESSL)
  a := 16807;
  M := 2^31-1;
  // Libovolně zvolím první hodnotu
  x[1] := 123456;
  // Vygenerování "náhodných" čísel od 0 do M-1
  pro i od 2 do počet opakuj
    x[i] := (a * x[i-1]) mod M;
konec.
```

Vypsání posloupnosti

```
začátek
  n := 10; // Počet hodnot
  // Definice posloupnosti náhodných hodnot (od 1 do 99)
  pro i od 1 do n opakuj
    a[i] := ((random * 100) div 1) mod 100;
  // Převedení hodnot posloupnosti do jednoho textového řetězce
  s := '';
  pro i od 1 do n opakuj
    jestliže s = '' pak
      s := a[i]
    jinak
      s := s + ', ' + a[i];
  // Vypsání textového řetězce
  napiš(s);
konec.
```

Algoritmus řazení - Buble sort

```
začátek
  n := 15; // Počet hodnot
  // Vygenerování pseudo náhodných hodnot (od 1 do 30)
  a[1] := 19; // První libovolně zvolená hodnota
  pro i od 2 do n opakuj
    a[i] := (3*a[i-1]) mod 31;
  // Seřazení hodnot
  c := n - 1;
  změna := true;
  dokud změna opakuj
  začátek
    změna := false;
    pro i od 1 do c opakuj
      jestliže a[i] > a[i+1] pak
      začátek
        pomocná := a[i];
        a[i] := a[i+1];
        a[i+1] := pomocná;
        změna := true;
      konec;
  konec;
konec.
```

Odmocnina z X (celá část jejího čísla)

```
začátek
  čti(x);
  s := 0;
  c := 1;
  dokud s <= x opakuj
  začátek
    s := s + c;
    c := c + 2;
  konec;
  s := c div 2 - 1;
  napiš('Odmocnia z ', x, ' = ', s, '. (Skutečná honota je ',
      x^0.5, '.)');
konec.
```

Nalezení minimální hodnoty v matici

```
začátek
  // Definice hodnot v matici
  m := 4;
  n := 5;
  pro i od 1 do m opakuj
    pro j od 1 do n opakuj
      a[i, j] := ((random * 100) div 1) mod 100;
  // Nalezení minimální hodnoty v poly a jejího počtu
  min := a[1, 1];
  počet := 0;
  pro i od 1 do m opakuj
    pro j od 1 do n opakuj
      jestliže a[i, j] <= min pak
        jestliže a[i, j] = min pak
          počet := počet + 1
        jinak začátek
          min := a[i, j];
          počet := 1;
        konec;
  // Vypsání vásledků
  napiš('Minimální hodnota = ', min, '.');
  napiš('Počet minimální prvků = ', počet, '.');
konec.
```

Násobení matic

```
začátek
  // Rozměry matic A(m x n), B(n x r), C(m x r)
  n := 4;
  m := 5;
  r := 3;
  // Nastavení generátoru pseudonáhodných hodnot
  x := 19;
  rozsah := 31;
  // Definice hodnot matice a pseudonáhodnými hodnotami
  pro i od 1 do m opakuj
    pro j od 1 do n opakuj
    začátek
      x := (3*x) mod rozsah;
      a[i, j] := x;
    konec;
  // Definice hodnot matic B pseudonáhodnými hodnotami
  pro i od 1 do r opakuj
    pro j od 1 do n opakuj
    začátek
      x := (3*x) mod rozsah;
      b[j, i] := x;
    konec;
  // Vynásobení matic C = a x B
  pro i od 1 do m opakuj
    pro j od 1 do r opakuj
    začátek
      s := 0;
      pro k od 1 do n opakuj
        s := s + a[i, k] * b[k, j];
      c[i, j] := s;
    konec;
konec.
```

Seznam použité literatury

1. **Milková, E.:** *Algoritmy, typové konstrukce*. Gaudeamus, Hradec Králové, 2001

2. **Milková, E.:** *Algoritmy v příkladech*. Gaudeamus, Hradec Králové, 2004