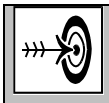
Úvod do
ALGORITMIZACE
a
PROGRAMOVÁNÍ

Autor: Radim Štefan

2006



Cíle předmětu:

po prostudování textu a příkladů budete znát a umět:

- **algoritmizovat jednoduché úlohy**
- **zobrazovat algoritmy ve formě blokových schémat**
- **nejdůležitější příkazy a klíčová slova programovacího jazyku Pascal**
- **převádět algoritmy do jazyka Pascal**
- **napsat jednoduché programy v prostředí Borland Pascal**

Získáte:

- **základní znalosti analýzy problémů**
- **prohloubení logického myšlení**

Budete schopni:

- **přesněji formulovat požadavky**

Čas potřebný k prostudování učiva předmětu: 24 vyučovací hodiny

Obsah

1 Vysvětlivky k používaným symbolům

Průvodce studiem – komunikace autora se studentem. Navrhuje způsob práce s textem, povzbuzuje studujícího a doplňuje text o další informace.

Příklad – slouží k objasnění probíraného tématu, nebo k jeho procvičování.

Pojmy k zapamatování – upozorňuje na důležité informace, které je potřeba si zapamatovat pro pochopení dalších témat.

Shrnutí – souhrn základních informací z předcházející látky. Většinou se týká shrnutí kapitoly.

Literatura – použitá ve studijním materiálu.

Kontrolní otázky a úkoly – prověřují, do jaké míry byly text a problematika pochopeny a zda student dokáže nové znalosti vhodně aplikovat při řešení problémů.

Korespondenční úkoly – představují úkoly, které studenti samostatně řeší a posílají ke kontrole vyučujícímu.

Úkoly k zamyšlení – slouží především k aplikování získaných znalostí.

Cíl – stanovuje cíl, kterého by mělo být dosaženo po nastudování a procvičení příslušné části.

Čas – časový odhad pro nastudování příslušné části. Je jen orientační.

2 Krátký úvod

Vážená studentko, vážený studente,

předmět Algoritmizace a programování spojuje dva samostatné předměty v jeden. Podle mého názoru, který vychází ze zkušenosti výuky programování, je vhodnější oba předměty sloučit. Vazba mezi oběma předměty je příliš těsná, než aby se učily odděleně.

Často se mezi vyučujícími ICT diskutuje otázka, který programovací jazyk je pro výuku nejvhodnější. To ale není nejdůležitější otázka. Pokud studenti pochopí a zvládnou princip algoritmizace a programování, nebude pro ně žádný problém naučit se další programovací jazyky.

Pro výuku programování jsem zvolil „Borland Pascal“, který díky své struktuře a čitelnosti je pro výukové účely velmi vhodný. K učebnímu textu je přiloženo i CD, které obsahuje elektronickou podobu (forma www stránek). Na CD jsou k dispozici také zdrojové kódy cvičení, jako soubory s příponou **.PAS**. Každý soubor obsahuje zadání a okomentovaný zdrojový kód.

Pro studium tohoto předmětu je důležité pochopit a zapamatovat si základní principy algoritmizace a aplikovat je pro řešení složitějších problémů. Je nutné studovat opravdu systematicky a uvedené příklady (řešené i neřešené) se snažit samostatně řešit. V případě nejasností, je důležité se k příkladům vracet i vícekrát. Tento učební text je vlastně vaším základem pro další studium. Můžeme to přirovnat k základům domu. Pokud nemáme základy, dům nepostavíme. Pokud nepochopíte základy, které jsou v tomto učebním textu, nemůžete sami ani uvažovat o psaní programu.

Mnozí si myslíte, že je zbytečné se učit programovaní, protože z vás nikdy programátoři nebudou. Jak se říká „nikdy, neříkej nikdy“. A i když se třeba opravdu programováním nebudete profesně zabývat, přesto zvládnutí tohoto předmětu je vám pomůže prohloubit vaše logické myšlení.

[1] KOSTOLÁNYOVÁ, K. *Algoritmizace a řešení problémů*. Ostrava: OU v Ostravě, 2002, ISBN 80-7042-227-0

[2] MIKULA, P. *Pascal 7.0 od příkladů k příkazům*. Praha: Grada, a.s., 1993, ISBN 80-85623-91-9

[3] MILDA, M. *Jak na to v Pascalu*. České Budějovice: Kopp, 1998, ISBN 80-7232-014-9

[4] RYCHLÍK, J. *Programovací techniky*. České Budějovice: Kopp, 1993, ISBN 80-85828-05-7

[5] ŠTEFAN, R. *Programovaní.* Ostrava: OU v Ostravě, 2002, ISBN 80-7042-254-8

Přeji Vám mnoho úspěchů ve studiu.

Radim Štefan

3 Algoritmizace a programování

3.1 Úvod

Cíl

Po prostudování oddílu 3.1 budete umět vysvětlit pojem algoritmus a objasnit, čím se vyznačuje.

Klíčová slova

Algoritmus, elementárnost, determinovanost, konečnost, rezultativnost, hromadnost.

Čas potřebný k prostudování učiva oddílu

24 vyučovací hodiny. Do tohoto počtu hodin nejsou zahrnuty hodiny pro řešení úkolů a opakování látky. Celkový odhad je asi 34 vyučovacích hodin.

Algoritmus – postup, který určuje postup, jak vyřešit libovolnou úlohu.

Algoritmus můžeme zapsat:

a. **slovně** – používá se jen pro jednoduché postupy. Pro složitější postupy může být slovní popis nepřehledný a nesrozumitelný. Příkladem algoritmů zapsaných slovně jsou návody k použití, kuchařské recepty, apod.

b. **graficky** – tento způsob zápisu využívá grafické symboly, které mají předem definovaný význam. Nejrozšířenější formy grafického zápisu jsou vývojové diagramy[1] a strukturogramy[2]. V dalším výkladu se budeme zabývat pouze vývojovými diagramy.

S algoritmy, jak už jsem se zmínil, se běžně setkáváme. Algoritmy nalezneme například v kuchařkách, jenže je nenazýváme algoritmy, ale recepty. Postup jak ovládat video kamery, fotoaparáty, mikrovlnky, automatické myčky, … jsou ve své podstatě taky algoritmy. Teoreticky lze každá činnost převést na algoritmy.

Vlastnosti algoritmů[3]:

Elementárnost	algoritmus je složen z konečného počtu jednoduchých kroků.
Determinovanost	v každé fázi zpracování existuje další postup. V jakékoliv situaci (krok algoritmu) musí být stanoveno, jak pokračovat dál.
Konečnost	činnost algoritmu musí skončit v reálném čase.
Rezultativnost	libovolná vstupní data musí vést k výsledkům. I informace: „nelze vypočítat“ je výsledek.
Hromadnost	algoritmus musí být aplikovatelný pro všechny úlohy stejného typu.

[1] Bude vysvětleno v dalších kapitolách

[2] Strukturální zápis kroků – může to být i zápis v symbolickém jazyce (programovací jazyk)

[3] URL: < http://home.tiscali.cz/~ca080987/FEI/_private/IVT/algoritmy.htm> [cit. 2006-03-19]

V tomto učebním textu se naučíte psát algoritmy jak ve formě vývojových diagramů, tak ve formě zdrojového kódu programovacího jazyka Pascal.

Úkoly k zamyšlení:

Napište nebo řekněte slovy algoritmus popisující:

- Uvaření čaje
- Přejití ulice se semafory
- Přejití ulice po přechodu bez semaforů
- Ranní vstávání
- Vymyslete si další vhodné náměty na algoritmy

Na vytvořených příkladech si ukažte vlastnosti algoritmů.

Příklad:

Navrhněte algoritmus – umytí rukou.

1. Jdu do koupelny.
2. Zastavím se před umyvadlem.
3. Otočím kohoutkem na studenou vodu.
4. Teče voda?
5. Ano
6. Je voda moc studená?
7. Ano.
8. Otočím kohoutkem na teplou vodu.
9. Namočím si ruce.
10. Namydlím si ruce.
11. Opláchnu si ruce.
12. Zavřu kohoutek s teplou vodou.
13. Zavřu kohoutek se studenou vodou.
14. Utřu si ruce do ručníku.
15. Odejdu z koupelny.

5. Ne.
6. Otočím kohoutkem víc.
7. Teče voda?
8. Ne.
9. Otočím kohoutkem zpět.
10. Odcházím z koupelny.
11. Počkám.
12. Jdu na první bod.

7. Ne.

Uvedený algoritmus je značně zjednodušený. Nejsou brány do úvahy další situace (stavy), které mohou nastat. Např. teče jen studená voda, voda je příliš horká, chybí mýdlo, schází ručník, v koupelně je tma, … Ale mohou nastat i různé naprosto neočekávané situace, které by měl algoritmus také řešit: praskne vodovodní trubka, bude zemětřesení, …

To co vypadá na první pohled, jako legrace jí přestane být, když programátor má zajistit, aby program fungoval za jakékoliv situace. Ale to je ještě hodně vzdálený bod programování.

3.2 Vývojové diagramy

Cíl

Po prostudování oddílu 3.2 budete schopni použít a nakreslit značky vývojového diagramu.

Klíčová slova

Vývojový diagram, koncová značka, příkaz, přiřazení, vstup dat, výstup dat, rozhodování, cyklus, podprogram.

Průvodce studiem

V této kapitole je vysvětlen zápis algoritmu ve formě vývojového diagramu.

Čas potřebný k prostudování učiva oddílu

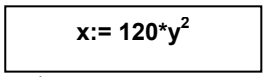
2 vyučovací hodiny.

Slovní zápis algoritmu má jen velmi omezené možnosti použití. Aby náš algoritmus byl srozumitelný a použitelný pro širší okolí uživatelů, můžeme ho zapisovat ve formě vývojových diagramů.

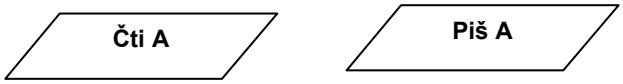
Ke kreslení vývojových diagramů používáme standardní grafické symboly. Tyto symboly naleznete v programu MS Word pod „Automatickými tvary“, část „Vývojové diagramy“ a „Spojovací čáry“.

Koncová značka. 

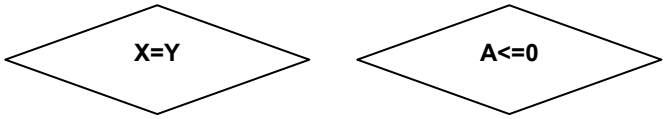
Každý zápis vývojového diagramu musí začínat koncovou značkou **Start** a končit značkou **Konec**. V diagramu se musí, vyskytnou obě značky vždy jen jednou a právě jednou.

Značka příkazu (např. přiřazení). 

V obecném pojetí to může být jakýkoliv příkaz – např. JDI, ZASTAV SE, OTOČ SE, …

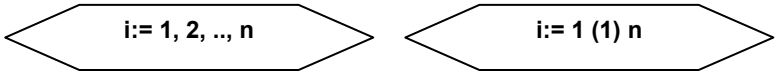
Vstup nebo výstup dat. 

Data jsou digitalizované informace. Mohou to být čísla, texty, …

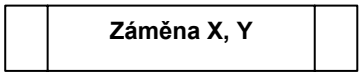
Značka rozhodování. 

Lze použít jen výraz, na který je odpověď ANO nebo NE. Např. porovnání dvou proměnných, ale i obecné dotazy jako např.: „Je teplá voda?“.

Příkaz cyklu



Značka podprogramu

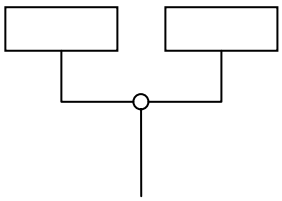


Podprogram je algoritmus, který v programu řeší nějaký úkol.

Jednotlivé symboly spojujeme čárami a spojkami. Ve vývojovém diagramu dodržujeme směr shora dolů, proto není nutné svislé čáry kreslit se šipkou. Čáru zakončujeme šipkou v případě, kdy se směr mění, např. při naznačení cyklu.

Spojky budeme používat v případě, že vývojový diagram nevejde na jednu stránky a pokračuje na straně další. Pak je vhodné pomocí spojek označit návaznosti pomocí čísla.

Dále spojku použijeme, pokud se v jednom bodě stýká více čar.

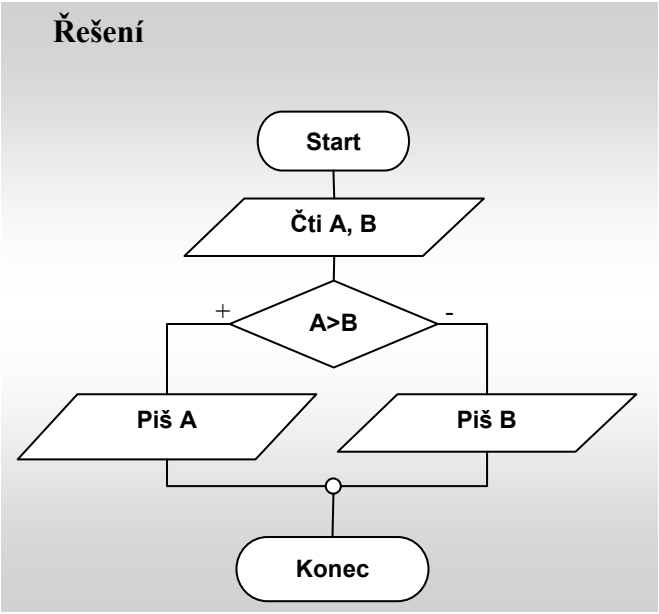


Ukázka jednoduchého algoritmu:

Zadání:

Vytiskněte větší číslo ze dvou.

Řešení



Ve vývojovém diagramu místo odpovědí **Ano** a **Ne**, se často používá plus **(+)** a mínus (**-**). Budeme to používat i my.

Pojem algoritmus již známe. Musíme se nyní tedy naučit zapsat algoritmus v přehledné a srozumitelné podobě, tedy ve formě vývojových diagramů. Ve vývojových diagramech již budeme používat zápis v souladu s programovacím jazykem, který v další etapě zajistí provedení (zpracování) algoritmu. Budeme používat programovací jazyk BORLAND PASCAL, který má přímou vazbou na programování v jazyce Delphi.

Vývojové diagramy při výuce však jsou důležité pro názornost a pochopení. Při skutečném programování se kreslení vývojových diagramů používá jen výjimečně. Je to dáno úrovní výpočetní techniky a úrovní programovacích a ladicího prostředí. V určitém stádiu programování vám bude stačit navrhnout jen hlavní body (procedury a funkce) a jejich vzájemné vazby.

Napsat podprogram nebo funkci bude rychlejší a snazší přímo v programovacím jazyku a vývojovém prostředí[4]. Současně využijeme ladicího prostředí a každou proceduru a funkci odladíme – otestujeme její funkčnost.

Vývojové diagramy zařazené v dalším textu řeší velmi jednoduché problémy a měly by vám umožnit pochopení základních programátorských struktur.

Úkol:

Naučte se v programu MS Word, pomocí „automatických tvarů“, kreslit vývojové diagramy.

[4] Program, ve kterém zapisujeme zdrojový kód, provádíme překlad tohoto kódu do programového kódu, syntaktickou kontrolu a ladění.

4 Sestavení algoritmu

Cíl

Po prostudování oddílů 4.1 až 0 budete schopni zakreslit algoritmus formou vývojového diagramu a napsat tento algoritmus v jazyce Pascal. Dokážete vysvětlit rozdíly mezi základními strukturami algoritmu a budete umět aplikovat různé typy větvení a cyklů.

Klíčová slova

Posloupnost, větvení, cyklus.

Průvodce studiem

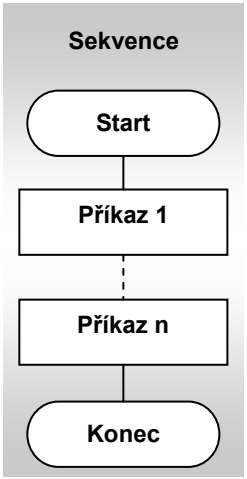
V této kapitole jsou vysvětleny všechny základní struktury algoritmu, které nás budou provázet během programování, již stále  .

Čas potřebný k prostudování učiva oddílu

12 vyučovacích hodin.

Algoritmus je sestaven na základě tří základních struktur:

1. **Posloupnost (sekvence)**
2. **Větvení (rozhodování)**
3. **Cykly (opakování)**



4.1 Posloupnost (sekvence)

Cíl

Po prostudování oddílu 4.1 budete schopni zakreslit sekvenční algoritmus formou vývojového diagramu a napsat tento algoritmus a jazyce Pascal.

Klíčová slova

Posloupnost, sekvence, matematické operandy, logické operandy, čtení, zápis.

Průvodce studiem

V této kapitole jsou vysvětleny základní struktury algoritmu a matematické a logické operandy používané v programování.

Čas potřebný k prostudování učiva oddílu

2 vyučovací hodiny.

Posloupnost je řada za sebou navazujících kroků, jejichž pořadí je předem pevně dáno. Posloupnost má svůj začátek a konec. Žádný krok nemůže být vynechán. Posloupnost se v algoritmech objevuje samostatně nebo jako součást složitějších struktur (větvení, cykly).

Vazba na programovací jazyk BP

Vzhledem k tomu, že je vhodné vývojové diagramy propojit s formální zápisem v jazyce BP, budeme si postupně uvádět způsoby zápisu v PB. Zde jsou první:

Matematické operandy:

`+` sčítaní

`-` odčítání

`*` násobení

`/` dělení

`()` závorky

Pro dosazení hodnoty, nebo matematického výrazu použijeme tzv. „dosazovací příkaz“.

`:=` dvojtečka, za kterou bezprostředně následuje rovnítko. Přestože jsou to dva znaky, vyhodnocují se jako jeden!!!!

`;` středník – každý příkaz musí být ukončen středníkem!

Pro vyhodnocování matematického výrazu platí stejná pravidla jako v matematice – zleva doprava – nejdříve násobení a dělení, a potom sčítání a odčítání. Pokud jsou ve výrazu závorky, vyhodnocuje se nejprve výraz v závorce.

Mějte na paměti, že pokud chceme s nějakou proměnnou pracovat, musíme nejprve ji „naplnit“ nějakou hodnotou!

Příklad 1 – zjisti, zda je výraz v pořádku a spočítej hodnotu proměnné Z:

```
X:= 10;            hodnota X je 10, hodnota Y je zatím nedefinována (neznámá)
Y:= x*x;           hodnota x zůstává 10, hodnota Y je 100
Z:= 100/(x+y-10);  hodnoty x, y se nemění, hodnota z je 1
```

Příklad 2 – zjisti, zda je výraz v pořádku a spočítej hodnotu proměnné C:

```
A:= 10;
C:= A*B;           řešení
```

Logické operandy (porovnávání):

< menší (např. **X<10**)

> větší (např. **A>0**)

= rovno (např. **A=X**) – všimněte si, že samostatné rovnítko se používá v případě porovnávání. Pro dosazování se musí použít **:=**

<> nerovná se (např. **i<>1**)

<= menší a rovno (např. **i<=1**). Pořadí operandů se nesmí zaměnit!

>= větší a rovno (např. **k>=1**). Pořadí operandů se nesmí zaměnit!

NOT negace

Úkol:

Porovnejte operandy jazyka BP s operandy, které se používají v matematice.

Funkce pro čtení a zápis

Read(); čtení proměnných (např. **Read(a,b);** - načtení proměnných a, b)

Write(); výpis proměnných nebo konstant (např. **Write(x,y);** - zobrazení obsahu proměnných x, y)

Logické závorky

Begin

End;

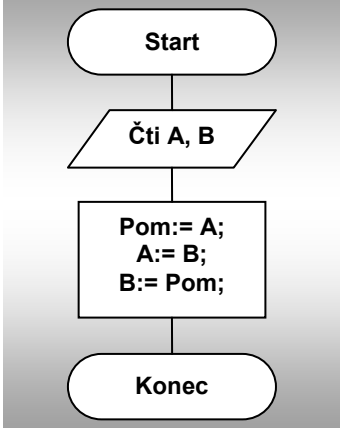
Logické závorky např. uvozují a ukončují algoritmy

Zadání:

Navrhněte algoritmus záměny obsahu proměnných (pomocí vývojového diagramu).

Na počátku algoritmu jsou proměnné A=1 a B=2. Po skončení algoritmu je A=2 a B=1.

Řešení



Zkuste si představit situaci: V každé ruce máte jeden velký a křehký předmět. Nelze mít v ruce oba předměty současně. Jakým způsobem je bezpečně zaměníte?

Pokud vás napadne, že předměty přehodíme vzduchem, pak toto řešení určitě nebude příliš bezpečné. Jediný optimální způsob je, že předmět z jedné ruky položíme na pomocnou podložku (např. na stůl). Předmět z druhé ruky přeložíme do prázdné ruky. Nakonec prázdnou druhou rukou vezmeme předmět ze stolu. Stejným způsobem to provedeme v algoritmu.

```
begin
  Read(A, B);
  Pom:= A;
  A:= B;
  B:= Pom;
end;
```

Pro vyřešení tohoto úkolu musíme zavést pomocnou proměnnou, do které uložíme hodnotu z proměnné A (první ze zaměňovaných hodnot). Pak můžeme hodnotu proměnné A přepsat hodnotou proměnné B. Po tomto kroku je obsah obou proměnných (A i B) stejný, což ale nevadí. V pomocné proměnné máme uloženou původní hodnotu A, proto nyní obsah pomocné proměnné uložíme do B.

Úkoly k zamyšlení:

Navrhněte algoritmus počítající:

- Délku dráhy na základě dané rychlosti a času pohybujícího se objektu.
- Obsah a obvod obdélníka a kruhu.
- Délku strany trojúhelníka podle Pythagorovi věty. Pro výpočet odmocniny použijte funkci s názvem SQRT (`a:= SQRT(x);` – odmocnina x).

4.2 Větvení (alternativa, rozhodování)

Cíl

Po prostudování oddílu 4.2 budete schopni zakreslit algoritmus s různým typem větvení formou vývojového diagramu a napsat tento algoritmus a jazyce Pascal.

Klíčová slova

Úplné větvení, neúplné větvení, vnoření větvení.

Průvodce studiem

V této kapitole jsou vysvětleny různé způsoby větvení algoritmu na základě různých logických operací.

Čas potřebný k prostudování učiva oddílu

4 vyučovací hodiny.

Větvení použijeme tam, kde podle okolností mají být některé kroky v posloupnosti vynechány, přidány nebo nahrazeny jinými.

Větvení obsahuje obvykle tři části. První částí je otázka, na kterou existuje kladná nebo záporná odpověď. Druhou částí je krok, který se provede v případě kladné odpovědi na otázku. Třetí částí je krok, který se provede v případě záporné odpovědi na otázku. První část větvení (otázka) je povinná, zbylé dvě části jsou nepovinné. Pokud však současně chybí druhý i třetí krok, ztrácí větvení smysl.

Ve vývojovém diagramu se tedy může objevit:

- **Úplné větvení**
- **Neúplné větvení**
- **Vnořené větvení**

4.2.1 Úplné větvení

Při úplném větvení jsou zařazeny kroky pro kladnou i zápornou odpověď. Podle toho, jak je sestavena podmínka, budou zařazeny další činnosti. Je nutné označit, která větev se provádí v případě, že podmínka je pravdivá a která větev se provádí, když je podmínka nepravdivá. V dalším textu je vždy pravdivost podmínky označena symbolem + (plus) a nepravdivost symbolem – (mínus).

Vazba na programovací jazyk BP

Pro větvení v BP se požívá příkaz **IF**. Používá se ve dvojím tvaru:

Pro úplné větvení:

```
IF podmínka THEN příkaz 1 ELSE příkaz 2;
```

Důležité:

***Za klíčové slovo* THEN *i* ELSE *lze použít jen jeden příkaz! Pokud je potřeba použít více příkazů, je nutné využít tzv. složeného příkazu pomocí logických závorek:* begin end.**

```
Begin
 Příkaz 1
 Příkaz 2
 …..
End;
```

Složený příkaz se jeví pro BP jako jeden příkaz!

Další možnosti příkazu Write.

Protože budeme potřebovat zobrazovat různé textové informace, ukážeme si další možnosti příkazu **Write**. Když potřebujeme zobrazit text, použijeme jednoduché uvozovky na začátku a na konci textového řetězce: `'Nelze spočítat'`. Lze kombinovat texty, proměnné i matematické výrazy, přičemž je budeme oddělovat čárkou.

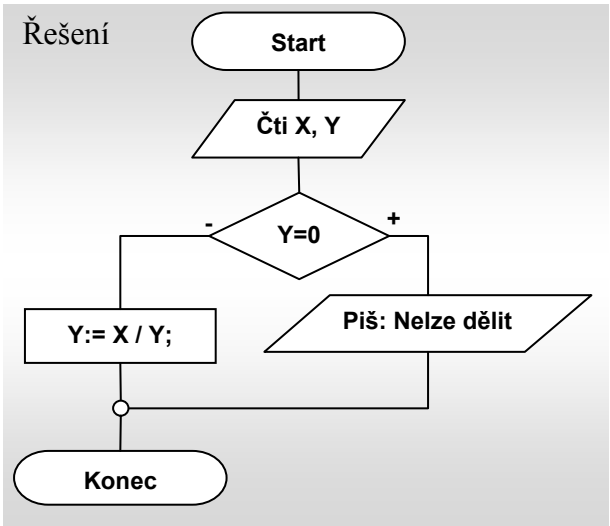
Příklad

`Write('Nelze spočítat y=',y);`	*Zobrazení textu a hodnoty proměnné* **y**
`Write('Podíl: ',x/y);`	*Zobrazení textu a výsledku matematické operace* **x/y**

Zadání:

Popište pomocí vývojového diagramu algoritmus načtení dvou proměnných X, Y a výpočet podílu těchto dvou čísel uložte do proměnné Y. Zajistěte, aby nedošlo k dělení nulou!!

Řešení




```
begin
 Read(X, Y);
 if Y=0
  then Write('Nelze dělit')
  else Y:= X/Y;
end;
```

4.2.2 Neúplné větvení

Při neúplném větvení chybí krok pro kladnou nebo pro zápornou odpověď (častější je situace, kdy chybí krok pro zápornou odpověď). V logickém sledu činností sice chybí krok pro zápornou odpověď, ale ve vývojovém diagramu je nutné tuto skutečnost zakreslit.

Vazba na programovací jazyk BP:

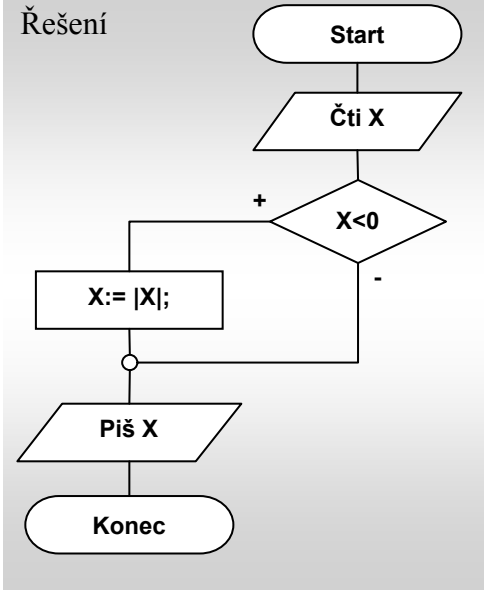
Pro neúplné větvení se neuvádí klíčové slovo **ELSE**:

```
IF podmínka THEN příkaz;
```

Zadání:

Popište pomocí vývojového diagramu algoritmus, který v případě, že číslo X je záporné, vypočítá jeho absolutní hodnotu. Obsah proměnné X vytiskne.

Řešení



Pro výpočet absolutní hodnoty v BP se použije funkce `Abs()`.

Příklad:

```
X:= Abs(X);
```

Zápis, který je zdánlivě nelogický, je zcela v pořádku. V BP je možné zapsat do proměnné hodnotu sebe sama (`X:= X;`). V tomto příkladu se nic nestane, protože se hodnota X nezmění.

```
X: = X+1;
```

V tomto případě se zvýší hodnota X o 1. Před tím, ale musíme zajistit, že v X bude vložena nějaká hodnota!

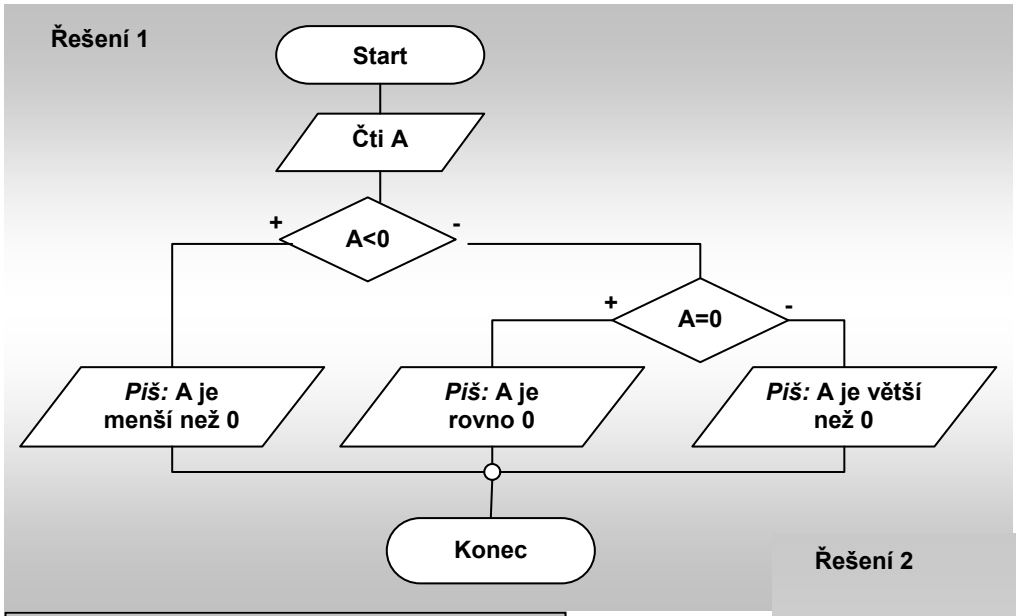
```
begin
 Read(X);
 if X<0 then X:= Abs(X);
 Write(X);
end;
```

4.2.3 Vnořené větvení

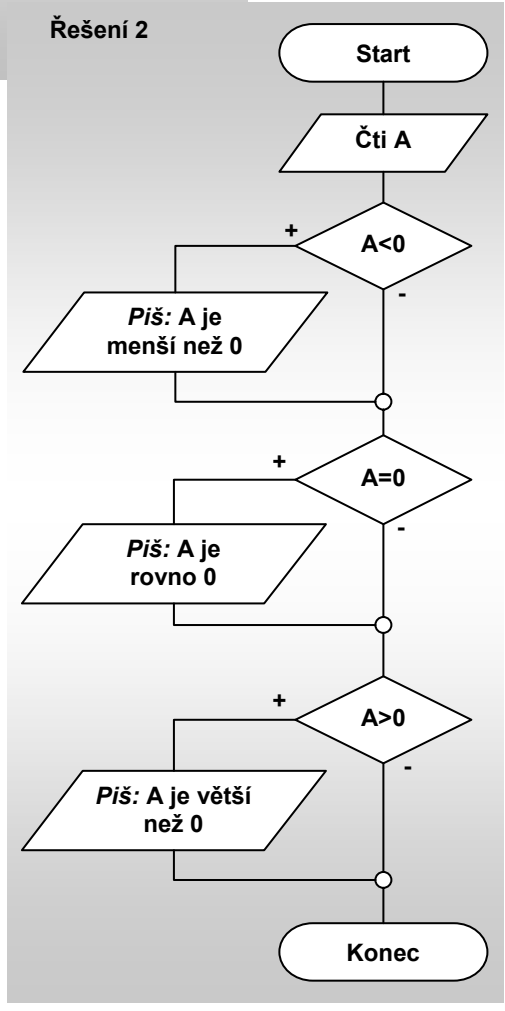
Vnořené větvení krok pro kladnou nebo pro zápornou odpověď (nebo pro obě odpovědi) je tvořen opět větvením.

Zadání:

Popište pomocí vývojového diagramu algoritmus, který zjišťuje, zda číslo A je menší nebo větší nebo rovno nule. Výsledné zjištění vypište.




```
begin
 Read(A);
 if A<0
  then Write('A je menší než 0')
  else
   if A=0
    then Write('A je rovno 0')
    else Write('A je větší než 0');
end;
```



Řešení 1 – používá vnořeného větvení.

Řešení 2 – používá neúplného větvení.

```
begin
 Read(A);
 if A<0 then Write('A je menší než 0');
 if A=0 then Write('A je rovno 0');
 if A>0 then Write('A je větší než 0');
end;
```

V předcházejícím příkladu jsme si uvedli dvě řešení k jednomu zadání. To není nic zvláštního. Pro všechny složitější zadání dokážeme většinou nalézt více řešení. Při výběru řešení je vždy důležitá jednoduchost a přehlednost.

Existuje tvrzení, že v každém programu je aspoň jeden nadbytečný příkaz, který lze odstranit, aniž by došlo k narušení funkčnosti  .

Takže si to vyzkoušíme v praxi.

Korespondenční úkol:

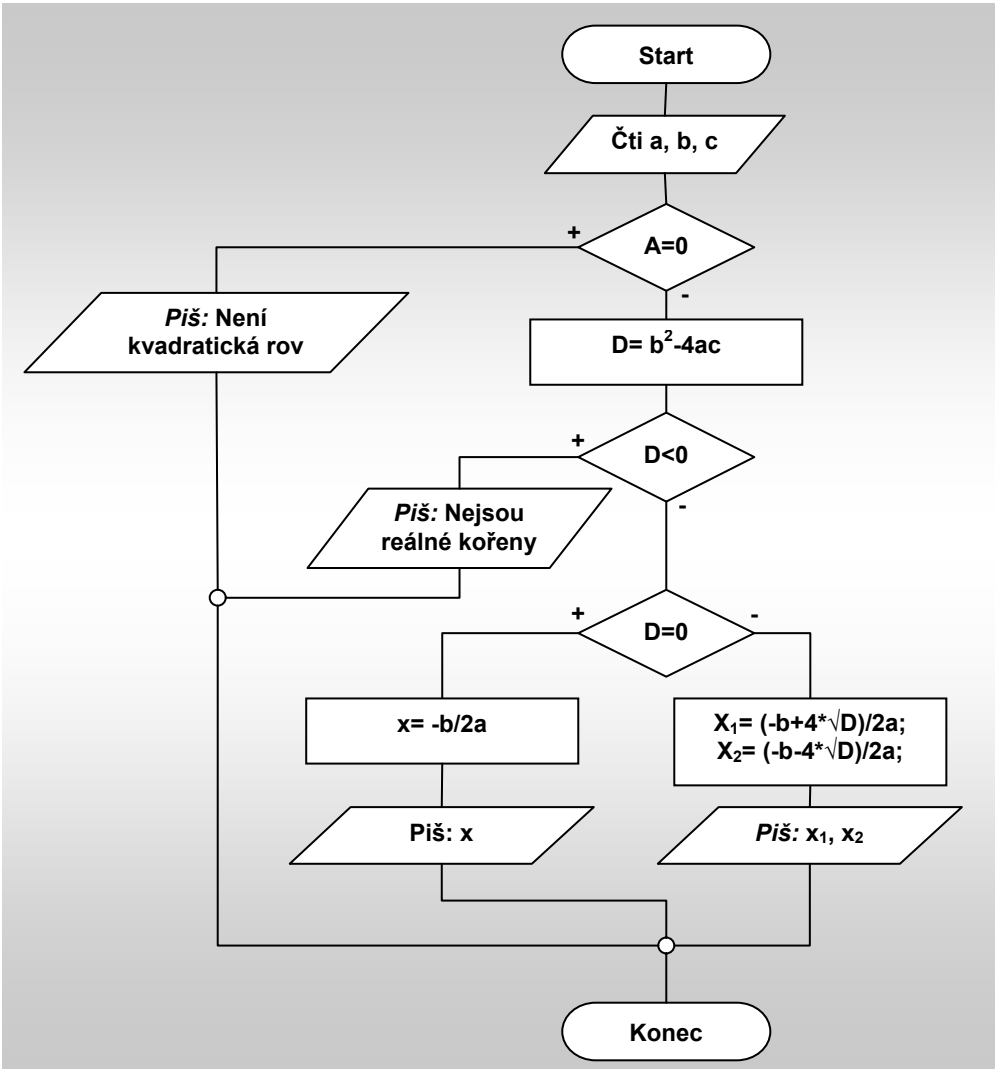
Optimalizujte řešení 2 – tj. odstraňte minimálně jeden příkaz a řešení pošlete na E-mail vyučujícího.

Příklady na procvičování:

Navrhněte algoritmus

1. Pro výpočet odmocniny, pokud má smysl.
2. Pro výpočet obvodu a obsahu obdélníka, pokud má smysl.
3. Pro výpočet obvodu a obsahu kružnice, pokud má smysl.
4. Pro výpočet délku strany trojúhelníka podle Pythagorovi věty, pokud má smysl.
5. Pro výpočet dráhy za známé rychlosti a času, pokud má smysl.
6. Pro výpočet kořenů kvadratické rovnice $ax^2+bx+c=0$.

Algoritmus pro výpočet kořenů kvadratické rovnice.




```
begin
 Write('Zadej a, b, c: ');
 Read(a, b, c);
 if A=0
  then Write('Není kvadratická rovnice')
  else
   begin
    D:= b*b-4*a*c;
    if D=0 then
     begin
      x:= -b/(2*b);
      Write('Jeden kořen: ',x);
     end
    else
     begin
      x1:= (-b+SQRT(D))/(2*a);
      x2:= (-b-SQRT(D))/(2*a);
      Write('x1: ',x1,'x2: ',x2);
     end
   end
end;
```

Kontrolní otázky:

1) Popište vlastnosti algoritmu.
2) Vysvětlete, jaká jsou omezení při použití pouze posloupnost příkazů.
3) Jaký má smysl větvení v algoritmu?
4) Popište rozdíl mezi úplným a neúplným větvením.
5) Analyzujte výhody a nevýhody použití vnořeného větvení.

Zadání korespondenčního úkolu:

Popište pomocí vývojového diagramu algoritmus, který zjišťuje, zda načtené číslo X leží uvnitř nebo mimo nebo na hranici intervalu (10, 20).

Poznámka:

Pro testování můžete použít následující podmínky:

```
(X>10) AND (X<20)
(X<10) OR (X>20)
(X=10) OR (X=20)
```

4.3 Cykly (opakování)

Cíl

Po prostudování oddílu 4.3 až budete schopni zakreslit algoritmus s různými typy cyklu a napsat tento algoritmus a jazyce Pascal. Dokážete zvolit vhodnou variantu cyklu pro dané řešení.

Klíčová slova

Cyklus, podmínka na počátku, podmínka na konci, řízená proměnná.

Průvodce studiem

V této kapitole jsou vysvětleny různé typy cyklů i s příklady použití.

Čas potřebný k prostudování učiva oddílu

6 vyučovací hodin.

V algoritmech velmi často nastává situace, kdy musíme některé činnosti zopakovat. To, zda se opakování provede a kolikrát, závisí vždy na vyhodnocení určité podmínky. Pokud přesně víme, kolikrát se má činnost opakovat, pak podmínkou kontrolujeme, zda již bylo opakování provedeno v potřebném počtu. V jiném případě opakování závisí na vzniku určité situace, např. když při výpočtu dojde k překročení určité extrémní hodnoty[5], pak podmínkou kontrolujeme vznik této situace.

Existují dva základní typy cyklů. Cyklus, který nejdříve vykoná určité příkazy, a pak teprve vyhodnocuje podmínku opakování. Druhý případ je cyklus, který nejdříve vyhodnocuje podmínku opakování**,** a pak provádí příkazy. Cyklus s řídící proměnnou je určitou variantou cyklu s podmínkou na začátku. Tento cyklus se používá velice často, protože zpřehledňuje práci programátora.

Budeme tedy rozlišovat 3 typy cyklů (všechny uvedené cykly jsou v dalším textu podrobně vysvětleny):

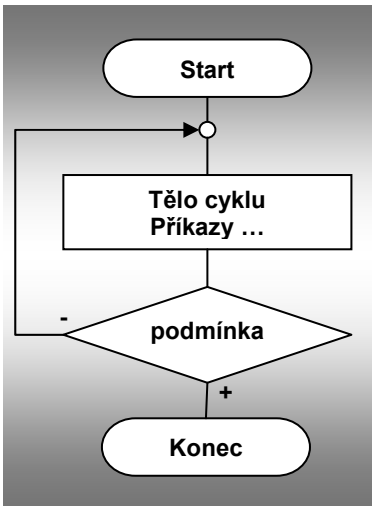
a) cyklus s podmínkou na konci

b) cyklus s podmínkou na začátku

c) cyklus s řídící proměnnou

[5] Ukončení cyklu může nastat i překročení počtu kroků nebo zvoleného časového intervalu trvání cyklu

4.3.1 Cyklus s podmínkou na konci (s výstupní podmínkou)



Postup provádění kroků cyklu:

Nejdříve se vykonají příkazy těla cyklu, pak se provede vyhodnocení podmínky. Má-li podmínka hodnotu FALSE (není pravdivá), provede se návrat na začátek těla cyklu, provedou se příkazy těla cyklu a opět se vyhodnotí podmínka. Tato činnost se opakuje tak dlouho, až podmínka nabude hodnotu TRUE (je pravdivá). Pak je cyklus ukončen.

Při použití tohoto cyklu si musíte zapamatovat, že se příkazy těla cyklu provedou vždy alespoň jednou.

Zde vás musím upozornit na to, že neplatí obecně pro každý programovací jazyk, že cyklus s výstupní podmínkou končí při hodnotě TRUE. Toto platí pro BORLAND PASCAL, Delphi a v celém dalším textu je toto pravidlo dodrženo. Například pro programovací jazyk C platí, že cyklus končí při hodnotě FALSE.

Vazba na programovací jazyk BP

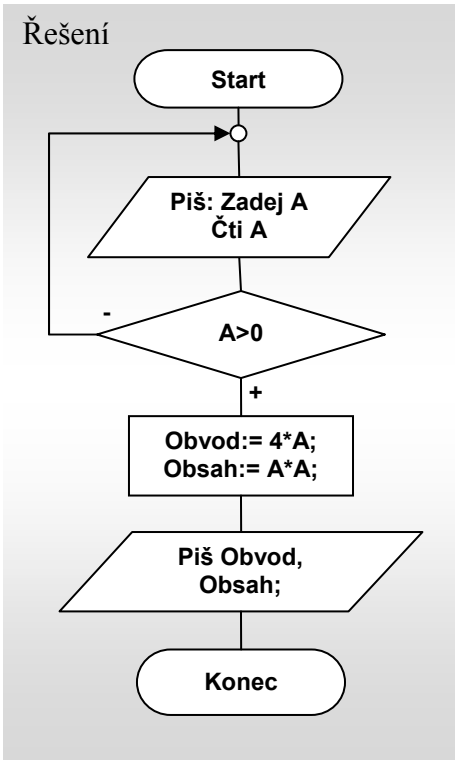
Pro cyklus s podmínkou na konci se používá příkaz:

```
Repeat
 Příkaz 1
 Příkaz 2
 . . . . .
Until podmínka;
```

Zadání:

Popište pomocí vývojového diagramu algoritmus výpočtu obvodu a obsahu čtverce o straně A. Zajistěte, aby výpočet proběhl pouze tehdy, jestliže strana A bude mít velikost větší než nula (v opačném případě nemá smysl výpočet provádět). Pro řešení použijte cyklus s podmínkou na konci.

Řešení




```
begin
 repeat
  Write('Zadej stranu čtverce');
  Read(A);
 until A>0;
 Obvod:= 4*A;
 Obsah:= A*A;
 Write('Obvod: ',Obvod,' Obsah: ',Obsah);
end;
```

Cyklus, který je zařazen ve vývojovém diagramu, je ukázkou cyklu s výstupní podmínkou. V uvedeném příkladu se jedná o situaci, kdy nevíte, kolikrát se činnost bude opakovat. Opakování činnosti má končit při zadání hodnoty větší než nula. Tomu tedy odpovídá sestavená podmínka.

4.3.2 Cyklus s podmínkou na začátku (se vstupní podmínkou)

Postup provádění kroků cyklu:

Nejdříve dojde k vyhodnocení podmínky. Má-li podmínka hodnotu TRUE, provede se tělo cyklu (jeden nebo více příkazů) a pak se provede automaticky návrat k podmínce, ta se opět vyhodnotí. Pokud má podmínka opět hodnotu TRUE, celá činnost se opakuje. Cyklus je ukončen až tehdy, když podmínka nabude hodnoty FALSE. U tohoto cyklu se může stát, že se tělo cyklu neprovede ani jednou! To nastane v případě, že podmínka má hodnotu FALSE již při prvním vyhodnocení.

Vazba na programovací jazyk BP

Pro cyklus s podmínkou na konci se používá:

```
While podmínka do Příkaz;
```

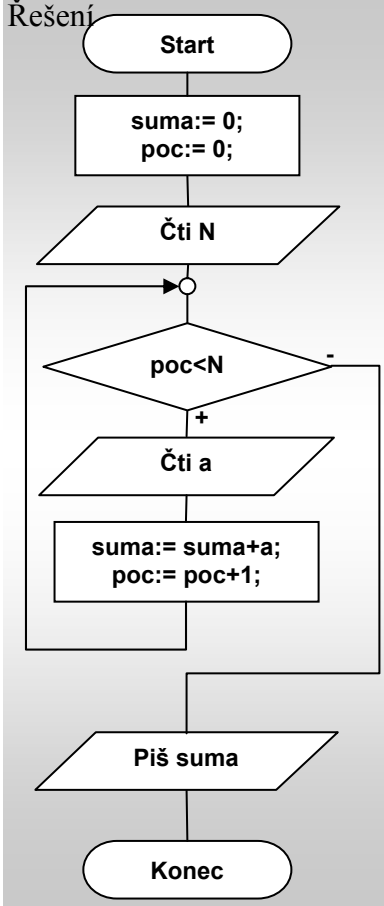
Za „**do**“ lze použít jen jeden příkaz. V případě, že je potřeba použít více příkazů, je opět nutné využít složeného příkazu **begin – end**.

Zadání:

Popište pomocí vývojového diagramu algoritmus sečtení souboru čísel. Vyřešte situaci, kdy víme, že v souboru je N čísel. Pro řešení použijte příkaz cyklu s podmínkou na začátku.

```
begin
 pom:= 0; poc:= 0;
 Write('Zadej N');
 Read(N);
 while poc<N do
  begin
   Write('Zadej číslo: ');
   Read(a);
   Pom:= pom+a;
   Poc:= poc+1;
  end;
 Write('Suma: ',suma);
end;
```

Řešení



4.3.3 Cyklus s řídící proměnnou

Cyklus s řídící proměnnou je poněkud zjednodušený cyklus se vstupní podmínkou. Lze jej použít pouze tehdy, jestliže počet opakování je dán explicitně a nezávisí na činnosti prováděné v těle cyklu.

Postup provádění:

Cyklus je řízen řídící proměnnou. Hodnoty řídící proměnné jsou omezeny počáteční a koncovou hodnotou cyklu. Na začátku provádění cyklu se do řídící proměnné uloží počáteční hodnota. Pokud je hodnota řídící proměnné menší nebo rovna koncové hodnotě, provedou se tyto činnosti:

- vykoná se tělo cyklu
- pak se provede návrat na začátek cyklu
- pak se automaticky zvýší hodnota řídící proměnné o 1
- pokud je hodnota řídící proměnné menší nebo rovna koncové hodnotě, celá činnost se opakuje
- cyklus končí tehdy, když řídící proměnná dosáhne hodnoty vyšší než je hodnota koncová.

Vazba na programovací jazyk BP

Pro cyklus s řídící proměnnou:

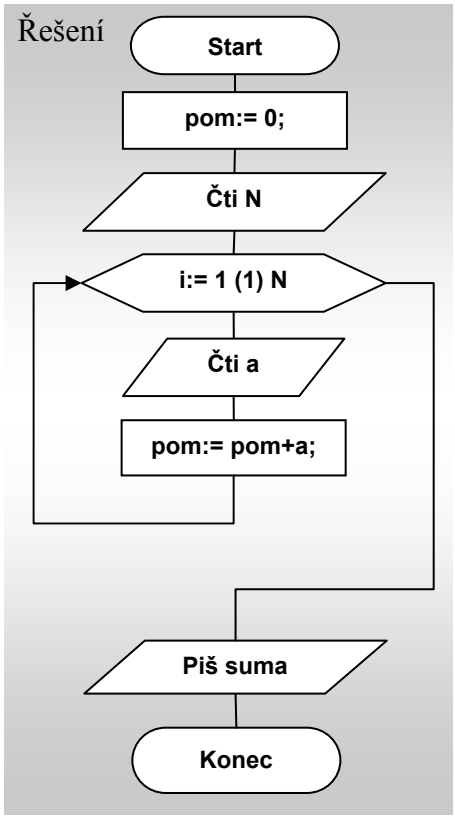
```
For i:=1 to N do Příkaz;
```

Kde N je počet kroků cyklu. Za „**do**“ lze použít jen jeden příkaz. V případě, že je potřeba použít více příkazů, je opět nutné využít složeného příkazu.

Zadání:

Stejné zadání jako v předcházejícím případu. Popište pomocí vývojového diagramu algoritmus sečtení souboru čísel. Řešte situaci, kdy víte, že v souboru je N čísel. Pro řešení použijte příkaz cyklu s řídící proměnnou.

```
begin
 pom:= 0;
 Write('Zadej N');
 Read(N);
 for i:=1 to N do
  begin
   Write('Zadej číslo: ');
   pom:= pom+a;
  end;
 Write('Suma: ', pom);
end;
```



Korespondenční úkol:

Zadání:

Popište pomocí vývojového diagramu algoritmus zjištění aritmetického průměru souboru čísel. Vyřešte situaci, kdy neznáte počet čísel v souboru, ale víte, že soubor je ukončen číslem 0, které ale do zpracování nesmíte zahrnout.

Takto zadaný úkol lze řešit pomocí cyklu s podmínkou na začátku i pomocí cyklu s podmínkou na konci. Nelze využít cyklus s řídící proměnnou, protože neznáme počet vstupních hodnot (počet opakování cyklu). Zvolte vhodný cyklus, napište algoritmus a pošlete na E-mail vyučujícího.

Poznámka: při tomto řešení nesmíme zapomenout na to, že ukončující hodnota 0 se nemá zpracovávat.

Kontrolní otázky:

1. Vysvětlete využití cyklů v algoritmu.
2. Analyzujte rozdíly mezi jednotlivými typy cyklů.
3. Popište omezení a výhody cyklus s řídící proměnnou.
4. Uveďte příklady použití jednotlivých cyklů.

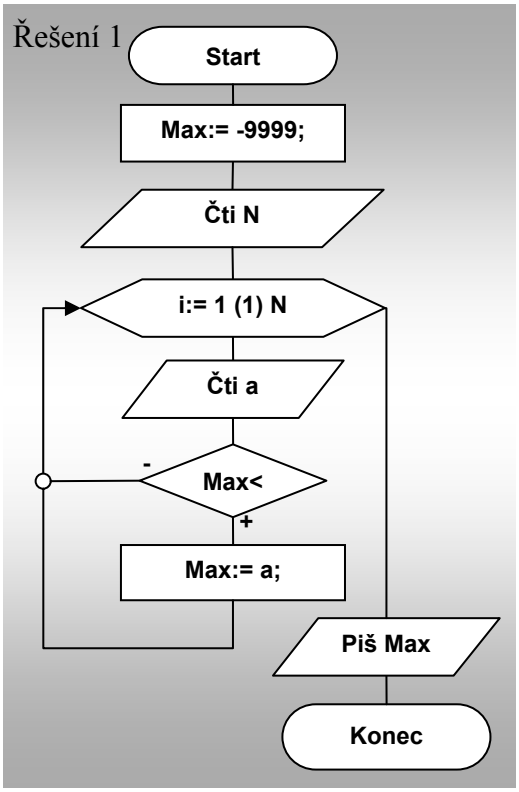
4.3.4 Příklady na použití cyklů

Zadání:

Nalezněte největší číslo z řady N čísel.

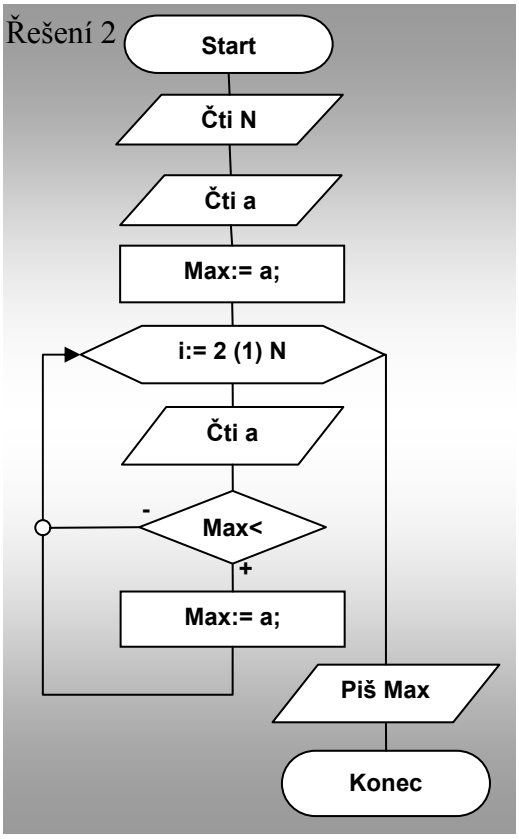
Řešení 1

```
begin
 Max:= -9999;
 Write('Zadej N');
 Read(N);
 for i:=1 to N do
  begin
   Write('Zadej číslo: ');
   Read(a);
   if Max<a then Max:= a;
  end;
 Write('Maximum: ', Max);
end;
```



Řešení 2

```
begin
 Write('Zadej N');
 Read(N);
 Write('Zadej číslo: ');
 Read(a);
 Max:= a;
 for i:=2 to N do
  begin
   Write('Zadej číslo: ');
   Read(a);
   if Max<a then Max:= a;
  end;
 Write('Maximum: ', Max);
end;
```



Úkol k zamyšlení:

Zdůvodněte, které z uvedených řešení je vhodnější a vysvětlete proč.

Korespondenční úkoly:

Vytvořte obdobné algoritmy (blokové schéma i v jazyce Pascal) a upravte pro hledání:

- nejmenšího čísla;
- nejmenšího čísla v absolutní hodnotě;
- největšího čísla v absolutní hodnotě.

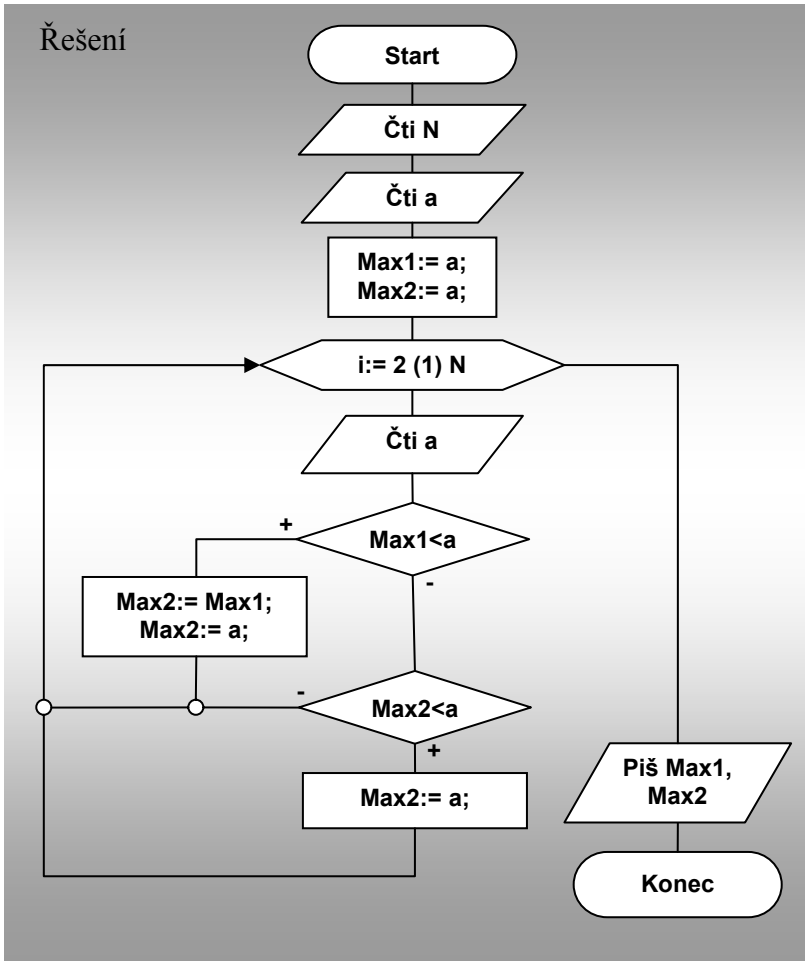
Řešení zašlete na E-mail vyučujícího.

Zadání:

Nalezněte dvě největší čísla z řady N čísel.

```
begin
 Write('Zadej N');
 Read(N);
 Write('Zadej číslo: ');
 Read(a);
 Max1:= a;
 Max2:= a;
 for i:=2 to N do
  begin
   Write('Zadej číslo: ');
   Read(a);
   if Max1<a then
    begin
     Max2:= Max1;
     Max1:= a;
    end
   else
    if Max2<a then Max2:= a;
  end;
 Write('Maxima: ',Max1,Max2);
end;
```

Řešení



Úkol k zamyšlení:

Algoritmy upravte pro hledání dvou nejmenších čísel

4.4 Procedury

Cíl

Po prostudování oddílu 4.4 budete schopni převést algoritmy na procedury.

Klíčová slova

Procedura, parametr.

Čas potřebný k prostudování učiva oddílu

2 vyučovací hodiny.

Napsat program se neobejde bez použití procedur a funkcí. Jsou to ucelené algoritmy, které provedou určitou činnost. Jak z algoritmu vytvoříme proceduru? Zcela jednoduše – použijeme slovo **procedure** a přidáme název. S funkcí je to podobné, jen je potřeba uvést její typ – bude vysvětleno později.

```
procedure NactiPole;
begin
 Write('Zadej N');
 Read(N);
for i:=1 to N do
  begin
   Write('Zadej číslo: ',i);
   Read(A[i]);
  end;
end;
```

Naše procedura se jmenuje NactiPole a nemá žádné parametry. Z tohoto konstatování vyplývá, že procedura může mít parametry.

4.4.1 Procedury a jejich parametry.

Když definujeme proceduru (jinými slovy – když píšeme její algoritmus), často potřebujeme, aby do ní vstupovaly nějaké proměnné nebo konstanty. Dosud jsme vždy v každém algoritmu používali vstup pomocí příkazu „**Čti**“ (v BP – Read). Pokud tyto algoritmy chceme upravit na procedury, můžeme nahradit tyto vstupy parametry procedury.

Při definici se uvádějí tzv. **formální parametry**.

Příklad:

Zadání:

Vytvořte proceduru pro výpočet podílu X, Y.

Vycházíme z původního algoritmu, který jsme řešily v předcházejících kapitolách:

```
begin
 Read(X,Y);
 if Y=0
  then Write('Nelze dělit')
  else Y:= X/Y;
end;
```

Změna na proceduru proběhne velice jednoduše:

```
procedure Podil(x,y: real);
begin
 if Y=0
  then Write('Nelze dělit')
  else Y:= X/Y;
end;
```

Vrátíme se zpátky k parametrům procedury **Podil**. Říká se jim **formální** parametry a v proceduře s nimi pracujeme jako s normálními proměnnými. Kdykoliv definujeme proceduru (popř. funkci) a chceme použít parametry, vždy se uvádí formální parametry.

Jak se procedura použije v programu? Jednoduše. Napíšeme její jméno, a pokud má parametry, nahradíme formální parametry skutečnými. Podmínkou je, že skutečné parametry jsou stejného typu jako formální. O typech se dozvíme v následujících kapitolách.

4.5 Deklarace proměnných a konstant

Cíl

Po prostudování oddílu 4.5 dokážete deklarovat proměnné základních číselných typů a budete vědět**,** jak je použít.

Klíčová slova

Deklarace, identifikátor, typ, celočíselné proměnné, reálné proměnné, textový řetězec, znak, logická proměnná, konstanta.

Čas potřebný k prostudování učiva oddílu

2 vyučovací hodiny.

4.5.1 Deklarace číselných proměnných

V předcházející kapitole se jako formální parametr požily proměnné x, y. Co znamená u nich to slovo **`Real`** za dvojtečkou? To je typ proměnné. Když definujeme (deklarujeme) proměnnou, vždy za ní uvedeme dvojtečku a napíše její typ.

Nyní nastal okamžik, kdy je potřeba si vysvětlit pojmy jako je typ a deklarace proměnných a konstant. Bude to trochu teorie, ale bez ní bychom se nemohli dostat v našem programování dál.

Hlavní pravidlo pro práci s proměnnými, funkcemi a procedurami je, že s nimi lze pracovat jen v případě, že už byly deklarovány.

Proměnné (identifikátory) se deklarují po uvedení klíčového slova **`VAR`**. Za tímto slovem následuje seznam proměnných s uvedením jejich typu. Název proměnné nesmí obsahovat znaky s diakritikou, mezery, a dále uvedené znaky:

```
+ - * = < > [ ] ( ) : , . „ @ { } / \ $ # & %
```

Délka identifikátoru je omezena na 63 znaků.

Jako první si uvedeme typy celočíselné (obor celých čísel):

Tabulka základních celočíselných typů

Typ	*Minimální hodnota*	*Maximální hodnota*	*Počet bytů*
Integer	-32768	32767	2
Byte	0	255	1
Shortint	-128	127	1
Word	0	65535	2
Longint	-2147483648	2147483647	4
Comp	$-2^{63}+1$	$2^{63}-1$	8

Tabulka 1

Celočíselné typy nepracují s desetinnou čárkou! Pokud bychom do celočíselné proměnné vložili číslo s desetinnou čárkou, došlo by k závažné chybě programu. Takže teď jsou na řadě typy čísel s pohyblivou řádovou čárkou (obor reálných čísel).

Tabulka číselných typů s pohyblivou řádovou čárkou

Typ	*Prahová hodnota*	*Maximální hodnota*	*Počet významných číslic*	*Počet bytů*	*Poznámka*
Real	$2{,}9\times10^{-39}$	$1{,}7\times10^{39}$	11-12	6	*)
Single	$1{,}5\times10^{-45}$	$3{,}4\times10^{38}$	7-8	4	
Double	$5{,}0\times10^{-324}$	$1{,}7\times10^{308}$	15-16	8	
Extended	$3{,}4\times10^{-4932}$	$1{,}1\times10^{4932}$	19-20	10	
Comp[6]	0	$9{,}2\times10^{18}$	19-20	8	Celá čísla

Tabulka 2

*) Typ Real není podporován matematickým koprocesorem, a proto výpočty s tímto typem jsou značně pomalejší.

Prahová hodnota znamená, že číslo, které je v intervalu 0 až prahová hodnota je počítačem považováno za nulu. Platí to i pro záporná čísla.

Příklady deklarací proměnných:

```
Var
  i:     integer;
  x:     single;
  a,b:   double;
```

Seznam proměnných stejných typů lze oddělit čárkou. Za poslední proměnnou musí být uvedena dvojtečka, za kterou pak následuje typ. Mezery v textu jsou ignorovány.

[6] Typ Comp je uveden v obou tabulkách. Je to sice celé číslo, které používá jen matematický koprocesor. Nelze ho použít např. jako řízenou proměnnou cyklu.

4.5.2 Další deklarace

Musíme se ještě seznámit s dalšími potřebnými deklaracemi: char, string, boolean.

Char deklarace znaku. Má velikost 1 byte

String deklarace textového řetězce. Maximální délka je 255 znaků. V paměti zaujímá vždy 256 znaků, ale skutečná použitá délka může být od 0 do 255 znaků.
V případě, že nám stačí kratší délka řetězce, můžeme ji uvést za deklarací **string** v hranatých závorkách

String[10] maximální délka použitelného textu bude 10 znaků. V paměti bude však zaujímat vždy o jedničku víc – 11 byte

Boolean deklarace logické proměnné. Její hodnota může být jen **TRUE** nebo **FALSE**. V paměti zaujímá 1 byte

Příklady deklarací proměnných:

```
Var
s:        string;
s1:       string[30];
c, b:     char;
t, x:     boolean;

begin
 s:= 'pokusný text - tento text mít až 255 znaků';
 s1:= 'pokusný text - max 30 znaků';
 s:= '';      {prázdný řetězec - neobsahuje žádný znak}
 c:= 'd';     {vložení jednoho znaku}
 t:= TRUE;
 x:= FALSE;

 if t then Write(s);       {bude vytistknut obsah řetězce s, protože t má hodnotu
                           TRUE}
 if x then Write(s1);      {obsah řetězce s1 nebude vytistknut, protože x má hodnotu
                           FALSE}

end;
```

4.5.3 Deklarace konstant.

Při psaní programu je často potřeba zajistit, aby proměnná měla určenou počáteční hodnotu, nebo konstanta měla své jméno (identifikátor). V sekci, ve které to lze zabezpečit, je uvozena klíčovým slovem **const**.

Pokud chceme definovat počáteční hodnotu proměnné, napíšeme její identifikátor s dvojtečkou, za kterou následuje typ. Za typem se uvede rovnítko, za kterým uvedeme příslušnou hodnotu. Obsah proměnných s počáteční hodnotou lze v průběhu programu libovolně měnit.

```
Const
  i:     integer= 100;
  x:     single= 10.1;
  a:     double= 1,11E15;       E znamená exponent za kterým může následovat kladné
                                nebo záporné číslo
  s:     string= 'Text';
  c:     char= 'x';
  t:     boolean= TRUE;
```

Pokud potřebujeme konstantu pojmenovat, uvedeme jen identifikátor, za kterým následuje rovnítko a příslušná hodnota. Tento identifikátor není proměnná, a tak ho nelze během programu měnit.

```
Const
  N= 100;
  eps= 1E-10;
  jmeno= 'Jméno';
  A= 'A';
  Ano= TRUE;
```

Zamyslete se nad konstantami **jmeno**, **A** a **Ano**. Vysvětlete smysl a navrhněte příklad, ve kterém je použijeme.

Příklady na procvičování:

Algoritmy, které jste již řešili dříve, upravte jako procedury s formálními parametry. Zvolte vhodné typy parametrů.

1. Pro výpočet odmocniny, pokud má smysl.
2. Pro výpočet obvodu a obsahu obdélníka, pokud má smysl.
3. Pro výpočet obvodu a obsahu kružnice, pokud má smysl.
4. Pro výpočet délku strany trojúhelníka podle Pythagorovi věty, pokud má smysl.
5. Pro výpočet dráhy za známé rychlosti a času, pokud má smysl.
6. Pro výpočet kořenů kvadratické rovnice $ax^2+bx+c=0$.

4.6 Indexované proměnné a pole

Cíl

Po prostudování oddílu 4.6 budete schopni pracovat s indexovanými proměnnými. Budete je umět deklarovat a používat především v cyklech.

Klíčová slova

Pole, array, index, indexovaná proměnná.

Čas potřebný k prostudování učiva oddílu

2 vyučovací hodiny.

Velice často je nutné zpracovávat značné množství hodnot. V tomto případě je vhodné nebo spíš jedině možné, použít indexované proměnné. V matematice se s nimi také setkáváme, např. jako označení řady čísel A_1, A_2, …. A_i… A_n. V blokových schématech můžeme používat stejné značení.

V BP se používá jiný způsob označování indexů, a to v hranatých závorkách – např. A[i]. Index může být jen celé číslo[7] a může nabývat i záporných hodnot a 0. Indexem může být i matematický výraz, jehož výsledkem je celé číslo nebo celočíselná proměnná. V BP můžeme místo označení indexovaná proměnná, používat označení pole (Array). V tomto případě je to pole jednorozměrné – jeden index. Z toho lze logicky odvodit, že budou i pole vícerozměrná, ale o tom až později.

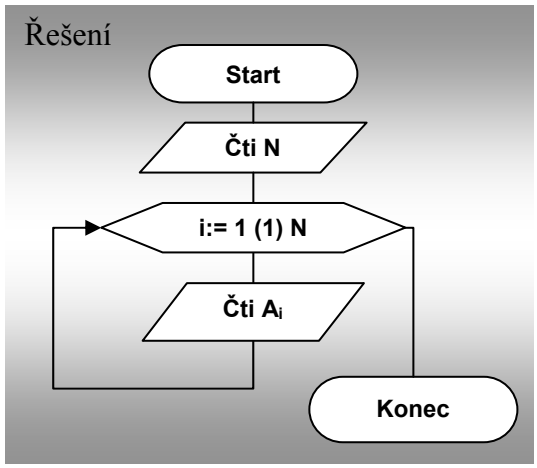
Zadání:

Načtěte N hodnot do pole A.

Algoritmus:

```
begin
 Write('Zadej N');
 Read(N);
for i:=1 to N do
  begin
   Write('Zadej číslo: ',i);
   Read(A[i]);
  end;
end;
```

Řešení



Procedura:

```
procedure NactiPole;
begin
 Write('Zadej N');
 Read(N);
for i:=1 to N do
  begin
   Write('Zadej číslo: ',i);
   Read(A[i]);
  end;
end;
```

[7] Může být i ordinální typ, ale o tom až v dalším díle.

4.6.1 Deklarace pole

Pokud chceme pracovat s indexovanou proměnnou, musíme deklarovat pole. Pro deklaraci pole požijeme klíčové slovo:

Array

Při deklaraci je nutné vždy uvést typ pole a rozsah. Rozsah se neuvádí pouze v případě formálních parametrů procedury nebo funkce. Rozsah se uvádí v hranatých závorkách. Vlevo je nejnižší hodnota indexu, následují 2 tečky a vpravo maximální hodnota indexu. Za hranatou závorkou následuje klíčové slovo **`of,`** za kterým je uveden typ.

```
Příklady:
Var
  Apole:  array[1..100] of integer;
  pole:   array[-100..1] of real;
```

Je to vlastně deklarace 100 čísel typu **`integer`** v prvním případě a **`real`** ve druhém. Ke každému z těchto čísel přistupujeme pomocí indexu, který je uveden v hranatých závorkách. Deklarovaný počet čísel v poli musí být rovný nebo větší, než skutečně použitý počet čísel.

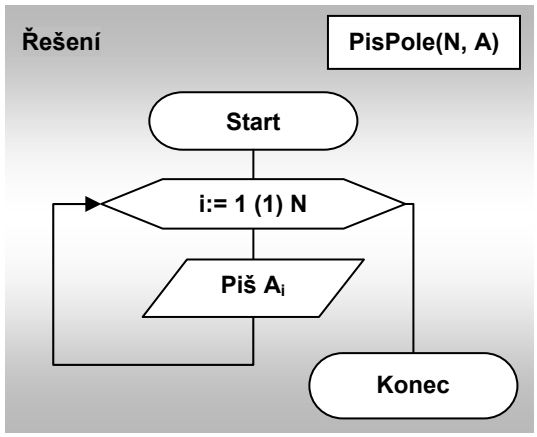
```
Apole[1]:= 1;
Apole[10]:= -100;
pole[-10]:= 999;
```

Jako index může být použita, jak jsem se již dříve zmínil, i celočíselná proměnná. Toho se využívá často v cyklech, jako je vidět v hned následujícím příkladu. Všimněte si, že při deklaraci pole jako formální parametr se neuvádí rozsah, ale jen typ.

Zadání:

Zobrazte N hodnot z pole A.

```
procedure PisPole(N: integer;A: array of integer);
var i: integer;
begin
 for i:=1 to N do Write(A[i]);
end;
```



Máme dvě procedury NactiPole (kap. 4.6) a PisPole. Teď je obě použijeme pro vytvoření algoritmu:

Zadání:

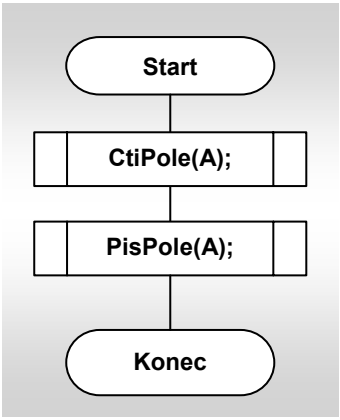
Načtěte a vytiskněte pole N čísel. Použijte procedury CtiPole a PisPole.

Abychom mohli algoritmus vyřešit, musíme ještě dořešit parametry u procedury NactiPole. Je sice pravda, že pole **A** v této proceduře načítáme, ale je potřeba načtené pole **A** „předat“ dál (jsou sice i jiné možnosti, ale ty si řekneme později). Formální parametry v případě, že potřebujeme hodnoty předat dál do skutečných proměnných, musí být uvozeny slovem **VAR**.

```
procedure NactiPole(var A: array of integer; var N: integer);
var i: integer;           {lokální proměnná}
begin
 Write('Zadej N');
 Read(N);
 for i:=1 to N do
  begin
   Write('Zadej číslo: ',i);
   Read(A[i]);
  end;
end;
```

Řešení:

```
begin
 NactiPole(A);
 PisPole(A);
end;
```

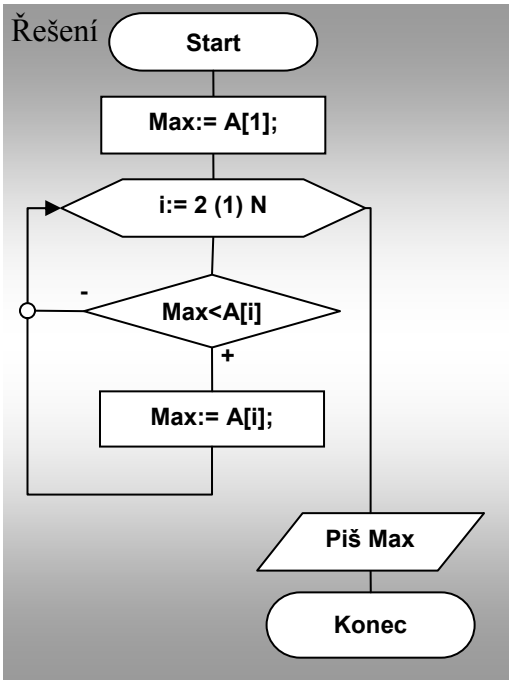


Příklady:

Opět se vrátíme k již řešeným algoritmům. Ukážeme si jejich řešení pomocí pole a nového prvku – funkce.

Zadání:

Nalezněte největší číslo z pole N čísel. Vytvořte algoritmus jako funkci.



Blokové schéma se nezmění. Jediná změna bude v jazyce Pascal. Místo slova **`procedure`** použijeme klíčové slovo **`function.`** Za ním musí následovat název (identifikátor) této funkce. Dále mohou následovat parametry stejně jako u procedury. Každá funkce musí být ukončená označením typu, stejně jako proměnná. Potom se s ní i jako s proměnnou pracuje, ale jen pro příkazy dosazení, v matematických výrazech a příkazech **`Write`**. Nelze ji přiřazovat hodnotu jako normální proměnné. Nejlépe to uvidíme na příkladě  . V první funkci používáme pomocnou proměnnou **`max`**. Ve druhém příkladě proměnnou **`max`** „ušetříme“, protože pracujeme přímo s identifikátorem funkce.

```
function NajdiMax(N: integer; A: array of integer): integer;
var i, max: integer;
begin
 max:= A[1];
 for i:=2 to N do
  begin
   if max<A[i] then max:= A[i];
  end;
 NajdiMax:= Max;
end;
```

Poslední příkaz v `NajdiMax:= Max;` způsobí, že výsledná nalezená hodnota bude vložena do identifikátoru (názvu) funkce. Algoritmus můžeme zjednodušit:

```
function NajdiMax(N: integer;var A: array of integer): integer;
var i: integer;
begin
 NajdiMax:= A[1];
 for i:=2 to N do
  begin
   if NajdiMax<A[i] then NajdiMax:= A[i];
  end;
end;
```

Korespondenční úkol:

Upravte již dříve napsané algoritmy tak, že využijete pole a procedury. Jedna procedura bude načítat řadu čísel a další bude řešit zadaný úkol.

- Nalezni největší číslo z řady N čísel.
- Nalezni nejmenší číslo v absolutní hodnotě z řady N čísel.

Označte křížkem správné deklarace:

☐ x: longint;	☐ MIN: extended;	☐ průměr: single;
☐ b, h: byte;	☐ a_b: byte;	☐ suma: real;
☐ integer: integer;	☐ z_y: byte;	☐ s_1: array[-50..50] of byte;
☐ x1, x: double;	☐ z2: longinteger;	☐ s_2: array[0..100] of real;
☐ e_X= shortint;	☐ e1: array[0.5..5.5] of real;	☐ 2a: longint= 100;
☐ e_3= 2,5E1585;	☐ e4: array[];	☐ c1: char= 'průměr';
☐ sx: string= 585;	☐ s1: string= 'pondělí';	☐ c2: char= '1';
☐ e_3: integer= 2,5E1585;	☐ e5: boolean= TRUE;	☐ e6: boolean= 'TRUE';
☐ e_4: integer= -999;	☐ e_5: integer= 35000;	☐ e6: single= 1E-27;
☐ e_6: byte= 99;	☐ e_7: byte= -1;	☐ e8: single= 1E127;

Ověření správnosti vašich odpovědí si vyzkoušejte přímo v prostředí BP tak, že tyto deklarace přepíšete. Překladač vás upozorní na chybné deklarace.

4.7 Vytvoření programu

Cíl

Po prostudování oddílu 4.7 budete umět napsat program, a tak aplikovat dosud již získané znalosti o algoritmizace a programování.

Klíčová slova

Program, globální proměnná, lokální proměnná.

Průvodce studiem

V této kapitole je vysvětlena struktura programu. Teď už nic nebrání začít psát jednoduché a fungující programy. 

Čas potřebný k prostudování učiva oddílu

2 vyučovací hodiny.

Pokud jste došli až sem, nic vám nebrání začít psát programy. Chybí vám jen znalost o struktuře programu. Není to nic složitého, jen je potřeba zopakovat důležitou zásadu – nelze použít žádný objekt (proměnnou, konstantu, proceduru, funkci), pokud již nebyl deklarován!

Struktura programu je následující: jako první příkaz je klíčové slovo program, za kterým následuje název (identifikátor) programu. Potom by měla následovat klausule **`USES`** se seznamem použitých knihoven (jednotek). Překladač BP dodává standardně 2 knihovny:

CRT – práce s obrazovkou

DOS – práce se soubory

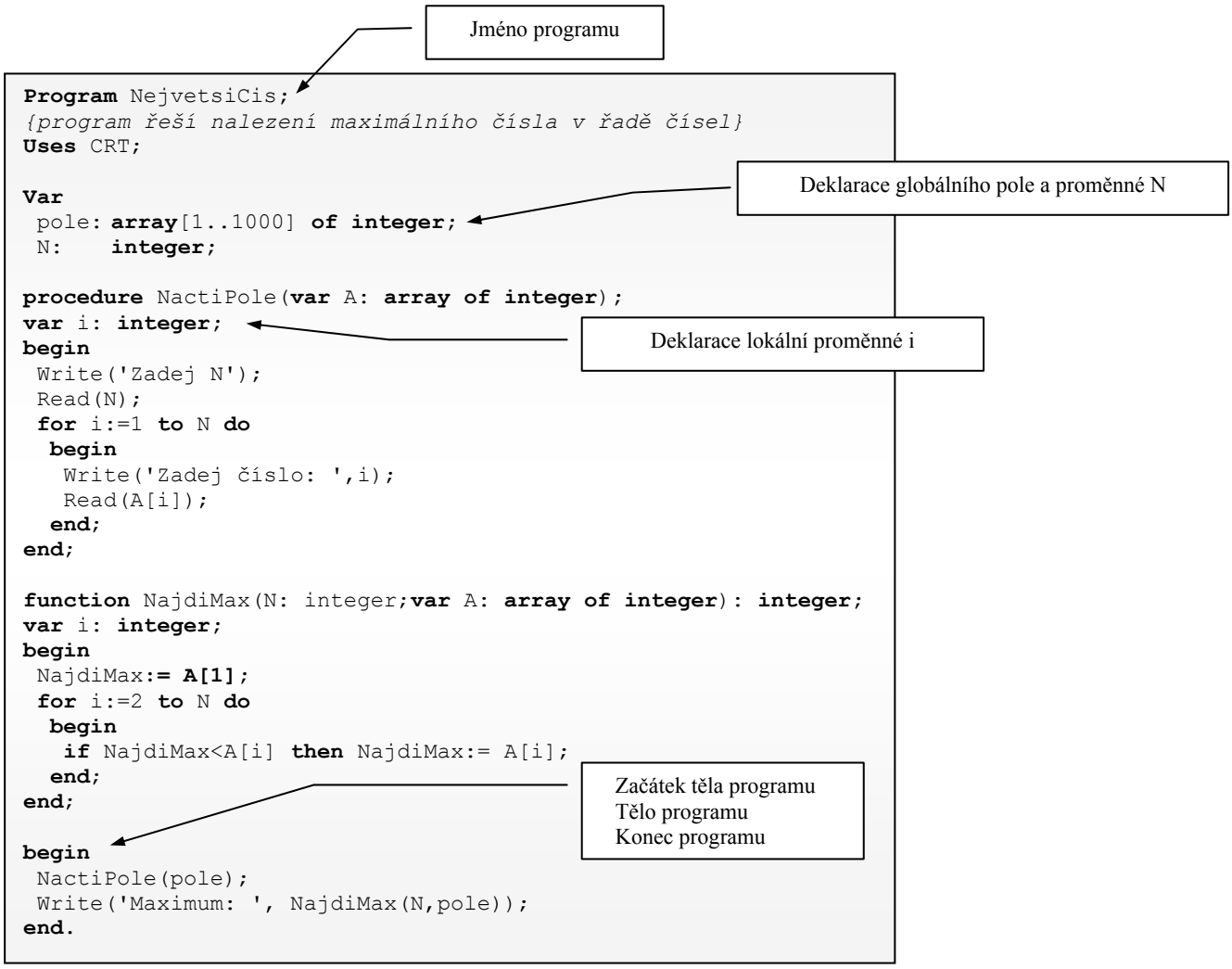
Popisy jsou zjednodušené.

Pak už budou následovat deklarace konstant a proměnných ve vhodném pořadí. Za nimi deklarace procedur a funkcí. Poslední částí je vlastní tělo programu. Je uvozeno klíčovým slovem **`begin`** a ukončeno **`end`**. Za tímto **`end`** se musí napsat tečka, která oznamuje logický konec programu. Cokoliv je napsáno za **`end`**, je bráno jako poznámka.

Zadání:

Napište program pro nalezení maximálního čísla v řadě čísel. Použijte pole a procedury.

Řešení




```
Program NejvetsiCis;
{program řeší nalezení maximálního čísla v řadě čísel}
Uses CRT;

Var
 pole: array[1..1000] of integer;
 N:    integer;

procedure NactiPole(var A: array of integer);
var i: integer;
begin
 Write('Zadej N');
 Read(N);
 for i:=1 to N do
  begin
   Write('Zadej číslo: ',i);
   Read(A[i]);
  end;
end;

function NajdiMax(N: integer;var A: array of integer): integer;
var i: integer;
begin
 NajdiMax:= A[1];
 for i:=2 to N do
  begin
   if NajdiMax<A[i] then NajdiMax:= A[i];
  end;
end;

begin
 NactiPole(pole);
 Write('Maximum: ', NajdiMax(N,pole));
end.
```

4.7.1 Poznámky v programu

Pro větší srozumitelnost je potřeba v programu používat poznámky. Popisujte si i relativně banální operace. Ušetříte si čas, když si zdrojový kód budete pročítat za delší dobu.

Poznámky lze v BP psát dvěmi způsoby:

1. pomocí složených závorek {text poznámky}
2. pomocí kulatých závorek s hvězdičkou (* text poznámky *)

Na začátku programu si popište i zadání.

4.7.2 Globální a lokální proměnné

Ve vysvětlivkách se objevily nové definice: globální a lokální proměnná.

Globální proměnná proměnná, která má platnost v celém programu. Deklaruje se na počátku programu a lze s nimi pracovat i v procedurách nebo funkcích.

Lokální proměnná deklarovaná proměnná v rámci procedury nebo funkce. Její platnost je lokální, pouze v dané proceduře a funkci. Lokální proměnné se vytvoří v okamžiku volání procedury nebo funkce a po ukončení činnosti procedury nebo funkce jsou opět zrušeny.

Příklady k řešení:

Než začneme s dalšími zadáními, seznámíme se s dalšími potřebnými funkcemi

Matematické operandy:

`Div` celočíselné dělení. Smí se použít jen pro celočíselné proměnné a výsledkem je opět celočíselná proměnná

`Mod` zbytek po celočíselném dělení. Opět se používá pro celočíselné proměnné a výsledkem je celé číslo. Příklad použití pro zjištění zda je číslo dělitelné pěti:

```
if (N mod 5)=0 then …
```

podmínka je pravdivá pro N dělitelné pěti

Zadání:

- Napište program, který zjistí počet kladných čísel z pole N čísel.
- Napište program, který zjistí počet sudých čísel z pole N čísel.
- Napište program, který zjistí počet lichých čísel z pole N čísel.

Výsledné programy zašlete E-mailem vyučujícímu..

5 Závěr

Ukončili jste úvodní část algoritmizace a programování. Ke skutečnému programování však máte ještě dost daleko. Nyní je vhodné se seznámit s možnostmi vývojového prostředí BP. Snažte se aplikovat algoritmy, které znáte, do nových programů. Jednou z potřebných činností je naučit se analyzovat předložené algoritmy. Pokud budete vědět, jak algoritmus funguje, budete jej umět i přizpůsobit také svým požadavkům.

6 Rejstřík

U

V

W

Z

7 Literatura

KOSTOLÁNYOVÁ, K. *Algoritmizace a řešení problémů.* Ostrava: Ostravská universita v Ostravě, Pedagogická fakulta, 2002, ISBN 80-7042-227-0

MIKULA, P. *Pascal 7.0 od příkladů k příkazům.* Praha: Grada, 1993, ISBN 80-85623-91-9

PRŮCHA, J. *Jak psát učební texty pro distanční studium.* Ostrava, Praha, Vysoká škola báňská - TU, Centrum pro studium vysokého školství: 2003, ISBN 80-248-0281-3

ŠTEFAN, R. *Autorské systémy – vývojové prostředí Delphi.* Ostrava: Ostravská universita v Ostravě, Pedagogická fakulta, 2002, ISBN 80-7042-253-X

ŠTEFAN, R. *Programování.* Ostrava: Ostravská universita v Ostravě, Pedagogická fakulta, 2002, ISBN 80-7042-254-8

TEXIERA, S., PACHECO, X. *Mistrovství v Delphi 6.* Praha: Computer Press, 2002, ISBN 80-7226-627-6

URL: < http://home.tiscali.cz/~ca080987/FEI/_private/IVT/algoritmy.htm> [cit. 2006-03-19] - algoritmy

URL: < http://virtualni.osu.cz/e-learning_pro_skoly/Kveton-Uloha_e-learningu_na_skolach.pdf > [cit. 2005-11-30] - úloha e-learnigu na školách

URL: <http://www.cdiv.upol.cz/www/autori_prirucka.htm> [cit. 2005-11-30] - příručka pro psaní distančních textů

URL: < http://www.elearn.cz/ > [cit. 2005-11-30] – e-learning

URL: < http://www.elearn.cz/soubory/e-learning_trends_ROI.pdf > [cit. 2005-11-30] - efektivita e-learningu

URL: < http://www.enviweb.cz/?secpart=ems_archiv_dgach_en_> [cit. 2005-11-30] – e-learning

URL: < http://www.net-university.cz/elearning/download/pruvodceelearning.pdf > [cit. 2005-11-30] - průvodce e-learningem