Základní algoritmické konstrukce ve výuce programování

Bc. Marek Drábek

Diplomová práce
2018

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2017/2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Marek Drábek**
Osobní číslo:	**A15413**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Učitelství informatiky pro střední školy**
Forma studia:	**prezenční**
Téma práce:	**Základní algoritmické konstrukce ve výuce programování**
Téma anglicky:	**Basic Algorithmic Construction in the Tuition of Programming**

Zásady pro vypracování:

1. **Proveďte literární rešerši oblasti výuky algoritmů na SŠ.**
2. **Vyberte vhodné druhy algoritmů pro výuku.**
3. **Připravte vývojové diagramy a tyto zpracujte ve vybraných jazycích.**
4. **Ověřené algoritmy zpracujte formou pracovních listů, ve kterých představíte možnosti jejich praktického použití.**
5. **Ověřte použitelnost pracovních listů ve výuce.**
6. **Vyhodnoťte využitelnost zvoleného řešení v praxi.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **CIENCIALA, Luděk a Lucie CIENCIALOVÁ. Teorie grafů a grafové algoritmy. Opava: Ústav informatiky, Výzkumný ústav Centra excelence IT4Innovations, Filozoficko-přírodovědecká fakulta v Opavě, Slezská univerzita v Opavě, 2014. ISBN 978-80-7510-060-3.**
2. **NECKÁŘ, Jan. ALGORITMY.NET [online]. 2016 [cit. 2017-11-17]. Dostupné z: http://www.algoritmy.net/.**
3. **PŠENČÍKOVÁ, Jana. Algoritmizace. Vyd. 2. Kralice na Hané: Computer Media, c2009, 128 s. ISBN 978-80-7402-034-6.**
4. **TÖPFER, Pavel. Algoritmy a programovací techniky. Praha: Prometheus, 2010, 300 s. ISBN 978-80-7196-350-9.**
5. **VANÍČEK, Jiří. Theory of algorithm. Praha: Kernberg, 2007, 431 s. ISBN 978-80-903962-4-1.**
6. **WRÓBLEWSKI, Piotr. Algoritmy. Brno: Computer Press, 2015. ISBN 978-80-251-4126-7.**

Vedoucí diplomové práce: **prof. Mgr. Roman Jašek, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **1. prosince 2017**

Termín odevzdání diplomové práce: **16. května 2018**

Ve Zlíně dne 11. prosince 2017

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
garant oboru

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 16. 5. 2018

.....
podpis diplomanta

ABSTRAKT

Cílem práce je vytvořit a odučit metodické listy pro výuku programování na středních školách. Tyto mají obsahovat základní, ale důležité programovací konstrukce, které mohou být rozvity v dalším studiu. V závěru je zhodnocena využitelnost těchto materiálů

Klíčová slova: Programování, Pascal, C++, pole, extrémy

ABSTRACT

The main purpose of this master thesis is create and demonstrate methosical sheets for tuition programming on high schools. These sheets cointain basic, but important algorithmic constructions which will become base for next learning of programming on university. Eventually, it discusses if these sheets can be used on high schools.

Keywords: Programming, Pascal, C++, array, maximum values

Tímto bych chtěl poděkovat panu magistrovi Pavlovi Špettovi z Gymnázia F.V ve Skalici, který mi umožnil využít mé metodické listy ve výuce.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Výuka Informatiky na středních školách se chystá projít důležitou reformou. Cílem této reformy je přiblížit formu výuky západním zemím, kde jsou počítače používané ve velké části předmětů. Všeobecně, u nás se na hodinách Informatiky probírá primárně kancelářský balík a hardware počítače. Až ve školách technického charakteru se pracuje s různými aplikacemi. V západních zemích se například tabulkové procesory se probírají na hodinách matematiky, práce s textovým editorem na hodině domácího jazyka a výuka Informatiky obsahuje mnohem více programování. Tato reforma samozřejmě neproběhne ze dne na den, ale postupně. Samotné vytvoření materiálů k tomu ale nestačí, je potřeba připravit i pedagogy. V požadavku globalizace výuky programování jsem se právě rozhodl vytvořit metodické listy, které jsou obsahem této práce. [4]

Mým cílem bylo vytvořit metodické listy pro výuku programování, které budou použitelné na různých typech škol, nejen technických. Tyto listy nepokrývají příliš velký obsah učiva, ale snaží se především rozvíjet logické a algoritmické myšlení žáků, které je u žáků středních škol na diametrálně rozličných úrovních. Jednotlivé dílčí části jsem se snažil demonstrovat krok za krokem s důrazem na logickou posloupnost, aby byly principy pochopitelné i pro méně technicky zdatné žáky, kteří by často nepřišli na středních školách do žádného kontaktu s programováním (výuka programování je buď nepovinná, nebo na volitelné bázi na většině středních škol, které nejsou technického charakteru). Nicméně výuka programování je tlačena stále do popředí a přizpůsobit náplň hodiny žákům, kteří nemají vztah k technickým aspektům, vyžaduje zvláštní přípravu. Materiálem, ze kterého jsem při vytváření svých metodických listů nejvíce čerpal, byla kniha „Zá-klady informačních technologií“ od pana doc. Botka, která mimo jiné obsahuje přehlednou formou zpracované příklady z oblasti algoritmizace.

Všechna zadání jsem zpracoval primárně v jazyku Pascal, který je dle mého názoru velmi názorný a ideální pro výklad, prezentování a pochopení jednodušších struktur. Zadání jsou dále zpracována i v jazyku C++, který je dnes více rozšířený a vhodnější k tvorbě uživatelských aplikací.

Metodické listy jsou rozděleny do 3 na sebe navazujících kapitol:

1. 1 rozměrné pole
2. Extrémy v 1 rozměrném poli
3. 2 rozměrné pole

Už v 1. kapitole navazuji teorií na základní znalost jazyka Pascal a k tomu odpovídající začátečnickou úroveň programování, kterou se snažím dále rozvíjet. Každá kapitola reprezentuje 1 obsahově rozvitý metodický list.

Struktura metodických listů

Každý list obsahuje několik vyřešených příkladů, které jsou prolínány potřebnou teorií. Každý příklad obsahuje nejprve zadání, slovně popsané řešení, grafické řešení ve formě diagramů a programové řešení ve formě kódu v jazyku Pascal. V příloze této práce se nachází kód všech programů v jazyku C++.

Metodické listy jsem navrhl pro využití ve třídách, kde je po seznámení se zadáním příkladu promítnuto řešení ve formě grafického diagramu a učitel slovným výkladem prezentuje dané postupy řešení. Po prezentaci žáci buď napíší celý zdrojový kód do počítačů (použitelná možnost pro jednodušší kódy), nebo doplní do kostry programového kódu, který se nachází na síťovém disku (šablony s kostrou kódu nejsou součástí práce) anebo upraví programový kód některého z algoritmů z předcházejících hodin, který učitel pro tento záměr zpřístupní na síťovém disku. V případě potřeby je možno celý kód, nebo jeho části, promítat dataprojektorem na plátno.

Všechny metodické listy jsem osobně odučil na Gymnáziu F.V. Sasinka ve Skalici na Slovensku, pod vedením Mgr. Pavla Špettu. Výuka probíhala se žáky 3. ročníku ve volitelném předmětu Informatika. V příloze se nachází potvrzený výkaz z praxe.

1 VÝUKA INFORMATIKY NA STŘEDNÍCH ŠKOLÁCH V ČR

Jak už bylo řečeno, kromě technických škol, primární náplní hodin Informatiky v ČR je kancelářský balík a hardware počítače. Na školách technického charakteru bývá programování povinné, na gymnáziích může být výuka programování povinná nebo na dobrovolné bázi, ale na školách s turistickým, humanitním nebo zdravotních zaměřením často není vůbec. Je to logické, často se sem hlásí žáci, kteří se snaží vyhnout technickým předmětům. Dnes používané rámcové vzdělávací programy byly vytvořeny více před 10 lety. Logicky neobsahují výuku samotného programování, jen rozvíjení logického a algoritmického myšlení. Národní ústav pro vzdělávání oznámil, že pracuje na nových rámcových vzdělávacích programech pro základní i střední školy, které budou v nějaké formě obsahovat přímou výukou programování. U základních škol se spekuluje u různých typů lehce programovatelných robotů, jakým je např. včelka, které se pomocí tlačítek na hřbetu určí směr, jakým se má pohybovat. Dosažení globální kvalitní výuky IT ale vůbec nebude lehký úkol. Jaroslav Fidrmuc, náměstek ministra školství, k problematice dodal: „V mnoha předmětech si žáci osvojují obecně známé postupy řešení. V informatice se zaměří na schopnost řešení vymyslet.“ Tato fráze se hezky poslouchá, ale její provedení není vůbec jednoduché. Prezentované postupy řešení ve škole se samozřejmě skládají z jednotlivých dílčích úkonů, které když žák pochopí a zároveň je schopen dostatečného konstruktivního myšlení, tak je schopen syntézou vytvořit řešení. Z mé zkušenosti je toho schopna ale jen ta nejtalentovanější část žáků, která má navíc daný předmět jako koníček. Schopnost vymyslet řešení musíme jako učitelé rozvíjet, nicméně je to jen část celku. Pokud máme za cíl výuku programování globálně rozšířit na většinu škol (základních i středních s různým zaměřením), musíme se soustředit na podporu adekvátního RVP, přípravu učitelů, dodržení technického standardu pro digitální vzdělávání na jednotlivých školách, zajištění potřebného množství hodin informatiky a především nenásilný a velmi postupný způsob její výuky. Poté prakticky odučit dostatečné množství postupů řešení a až následně se zaměřit ve větší míře na schopnosti vymyslet individuální možná řešení. Pokud nebudou dodrženy některé ze zmíněných bodů, úsilí pedagogů bude zbytečné a žáci buď nebudou schopni produktivního myšlení, nebo toho bude schopna jen zmíněna zájmová část a jsme tam, kde jsme byli. [2]

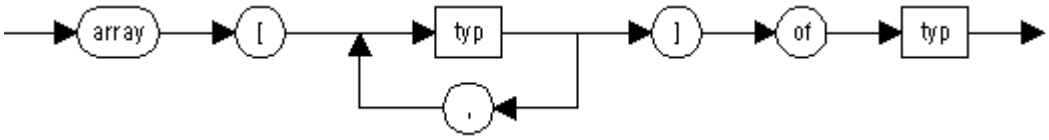
2 1 ROZMĚRNÉ POLE

Název:	1 rozměrné pole
Cíl:	Žáci se seznámí s principy datové struktury pole a naučí se základní postupy pro zpracovávání jednotlivých prvků pole.
Motivace:	Žáci navážou na svoje poznatky v oblasti programování a uvědomí si, jak je daná látka potřebná při zvládání náročnějších úloh.
Příprava:	Kontrola funkčnosti dataprojektoru a počítačů v učebně.
Postup:	Promítnutí teorie a diagramů na plátno s postupným vysvětlováním postupu práce. Ve výkladu je prostor pro otázky žáků, jak o věcech, které nejsou jasné, tak o možných řešeních daných problémů. Na konci je zadán domácí úkol.
Předpokládaný čas:	4 vyučovací hodiny
Cílová skupina:	3. ročník
Místo:	Počítačová učebna
Vyučovací metody a formy práce:	Výklad učitele Individuální práce Práce s informacemi Třídění a upevňování nových poznatků Společné vyhodnocení
Potřeby a materiál:	Dataprojektor a dostatečný počet počítačů připojených na školní síť.
Rozvoj KK:	KK řešení problémů KK k tvořivému myšlení KK pracovní KK k učení
Začlenění tematických okruhů PT:	Programování Jazyk Pascal
Vyhodnocení (reflexe):	Podpora myšlení žáků. Žáci pochopí, jak program zpracovává jednotlivé prvky pole. Žáci jsou schopni aplikovat prezentované postupy.

2.1 Pole

Pole je homogenní datová struktura, která obsahuje posloupnost prvků stejného typu. Jednotlivé prvky jsou rozlišovány pomocí indexu, který určuje jejich pořadí v poli. [2]

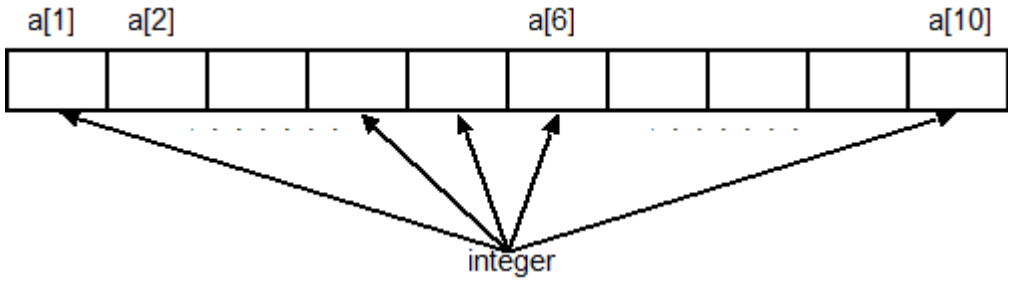
Deklarace proměnné typu pole:



Obrázek 1: Deklarace pole [2]

Např.:

```
a : array [1..10] of integer;
```



Obrázek 2: grafické znázornění pole [2]

Pořadí však nemusí být hodnota samotného indexu, můžeme použít celočíselný rozsah hodnot, nebo indexem může být i písmeno.[2]

Např.:

```
a: array [10..20] of integer;
```

V následujících programech si vždy předtím, než začneme pracovat s polem, zadáme a načteme jeho délku, která musí být rovna nebo menší rozsahu hodnot, pro které je pole deklarováno. Je to z důvodu, že pole má po deklaraci pevně vyhrazené místo v paměti počítače a při různých variacích algoritmů potřebujeme pracovat s polem různých délek a ne s polem, které má za každých okolností pevně stanovenou délku.

K procházení jednotlivých prvků celého pole používáme cyklus. Krokování cyklu zajišťuje proměnná, kterou v následujících příkladem budeme pojmenovávat písmenem „i“ a označovat jako počítadlo.

Příklad cyklu, který načte čísla (ze zdrojového souboru nebo z klávesnice) do pole s délkou „m“ a na vstupu jsou hodnoty 3,2,2,3.

```
a: array [1..10] of integer;
```

```
read(m);
For i:= 1 to m do
read a[i];
```

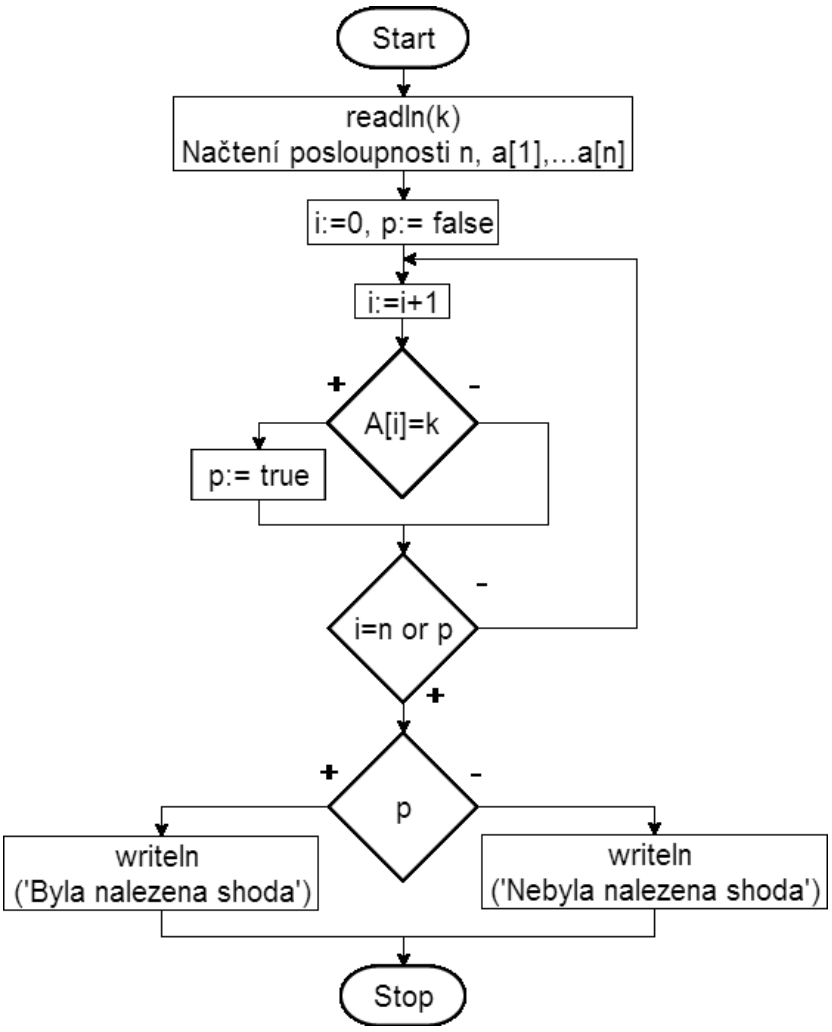
Nejprve se vytvořilo pole s názvem „Pole“ s délkou 10 prvků. Dále se načte do proměnné „m“ hodnota 3 a cyklus následně načte do pole s názvem „a“ 3 hodnoty (2,2,3).

2.2 Příklad 1

Zadání: Zjistěte, zda alespoň jeden z prvků posloupnosti má hodnotu rovnou zadané hodnotě „k“.

Cílem programu bude procházet prvky posloupnosti a porovnávat je s hodnotou „k“, kterou zadáme do programu. Pokud nastane shoda, program to oznámí a skončí. V případě, že program projde celým polem a shodu nenalezne, nastává výpis oznamující, že shoda nebyla nalezena a program opět končí.

Program si nejprve načte číslo, které bude hledat v poli, následně další číslo, které bude představovat počet prvků v poli a nakonec i samotný obsah pole. Do proměnné „p“, typu boolean, se uloží hodnota „false“. Následuje cyklus, který bude procházet pole prvek po prvku a každý tento prvek porovná s hodnotou uloženou v proměnné „k". Pokud se hodnota shoduje, program přepíše hodnotu proměnné „p“ na „true“. Cyklus následně už polem neprochází, protože obsahuje podmínku, která před každým posunutím na další prvek otestuje, zda se v proměnné „p“ nenachází hodnota „true“. Pokud se v ní tato hodnota nachází, cyklus skončí a následuje výpis, který oznámí, že byla nalezena shoda. Cyklus obsahuje i další podmínku, která v případě, že shoda nenastala, tento cyklus ukončí hned po zpracování posledního prvku v poli. V této variantě dále následuje výpis, který oznámí, že shoda nebyla nalezena. Diagram úlohy je znázorněn na obrázku 3.



Obrázek 3: Diagram pro hledání zadané hodnoty v poli [3]

Kód:

```
var
a: array [1..10] of integer;
i,m,k: integer;
p: boolean;
begin
read(k);
read(m);
p:=false;
for i := 1 to m do
    read(a[i]);
i:=0;
```

```
repeat
    begin
        i:=i+1;
        if a[i]=k then p:=true;
    end;
until ( (i=m) or (p=true) );
if p=true then writeln ('Prvek ',k,' se nachazi v posloupnosti')
          else writeln ('Prvek ',k,' se nenachazi v posloupnosti')
;
end.[3]
```

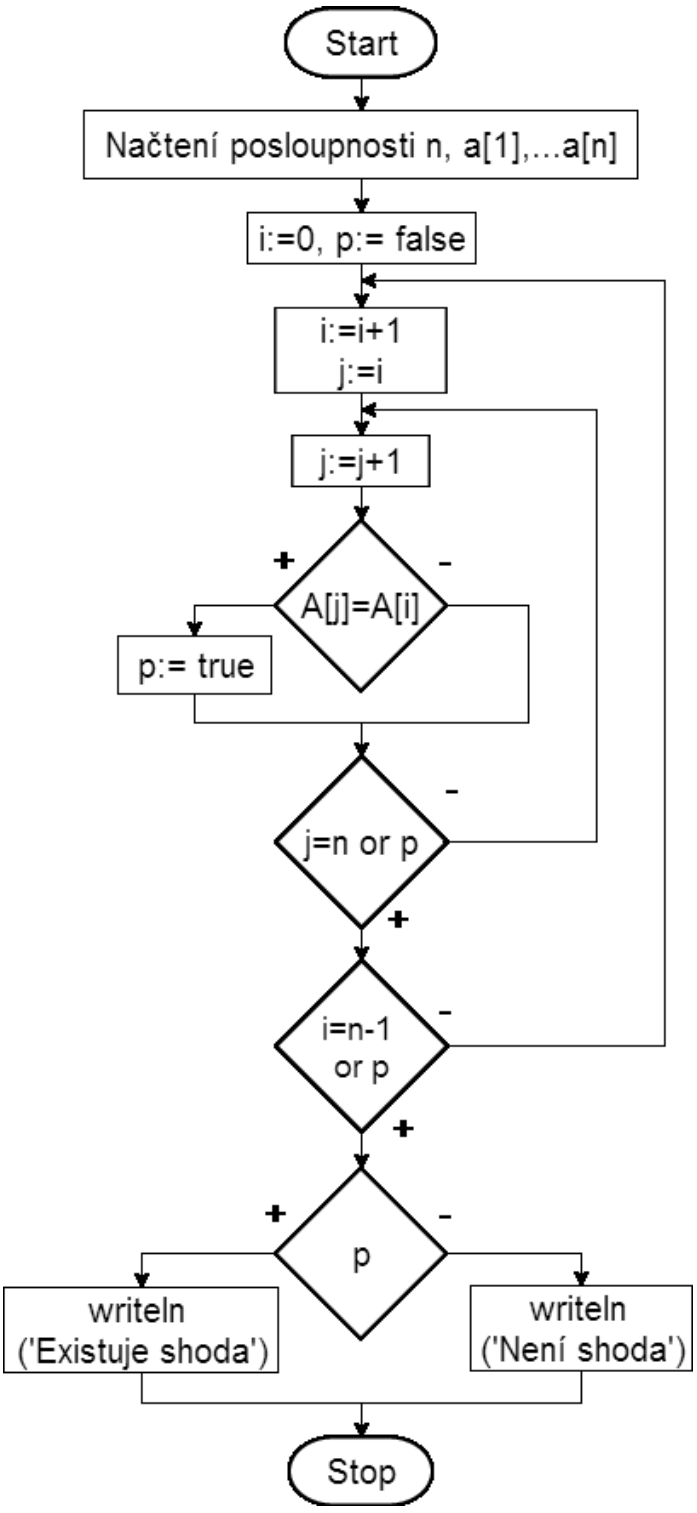
2.3 Příklad 2:

Zadání: Zjistěte, zda se v posloupnosti nachází alespoň 2 stejné prvky.

Princip tohoto programu spočívá v tom, že je potřeba porovnat prvky pole stylem každý s každým. Pokud máme například pole s prvky a[1], a[2], a[3] a a[4], tak musíme docílit toho, aby se prvek a[1] porovnal nejprve s prvkem a[2], pak s a[3] a s a[4]. Následně se prvek a[2] porovná s prvkem a[3] a a[4]. Nakonec se prvek a[3] porovná s a[4]. Poslední prvek a[4] se už se sebou porovnávat nemusí a proto cyklus končí. Tento cyklus tedy porovnává zpracovávaný prvek s prvky, které následují v posloupnosti za ním (prvky, které se nachází v posloupnosti před tímto prvkem, už není třeba procházet, protože už byly se zpracovávaným prvkem jednou porovnány).

Podobně jako v minulém příkladu, program si načte číslo, označující počet prvků, které bude obsahovat pole a následně obsah samotného pole. Poté proběhne cyklus, který bude procházet pole prvek po prvku. Tento cyklus bude opět pracovat s počítadlem „i“, které bude zvyšováno o hodnotu 1. Do tohoto cyklu vnoříme ještě další cyklus, který zabezpečí, že zpracovávaný prvek z předcházejícího cyklu se porovná s prvky následujícími. V tomto vnořeném cyklu použijeme nové počítadlo „j“, jehož počáteční hodnotu musíme nastavit na aktuální hodnotu počítadla „i“ a při každém svém průchodu se opět zvětší o 1. Tím jsme zabezpečili, aby se procházely jen ty prvky, které následují v posloupnosti až za zpracovávaným prvkem. Tento vnořený cyklus bude procházet prvky celého pole, proto jeho konečná hodnota bude poslední prvek pole (cyklus bude probíhat, dokud nenastane situace j=n). Vzhledem k faktu, že 1. průchod vnořeného cyklu pracuje hned s následujícím prvkem, který „dodá“ cyklus vnější, poslední zpracovávaný prvek vnějšího cyklu bude předposlední prvek v poli (jinak

řečeno, cyklus bude probíhat, dokud nenastane situace i=n-1). Toto je nutné, aby se zamezilo překročení hranic pole. Postup při samotném nalezení shody je podobný jako v předchozím případě, ještě před prvním cyklem se proměnná „p“, typu boolean, nastaví na „false“ a pokud je nalezena shoda, program hodnotu této proměnné přepíše na „true“. Každý ze 2 cyklů opět testuje, ještě před každým svým průchodem (kdy se zvětší počítadlo o 1), zda se v proměnné „p“ nenachází hodnota „true“. Pokud se v „p“ nachází, tak oba cykly končí a následuje výpis oznamující nalezení shody. Pokud program projde celým polem a shoda nenastane, zobrazí se výpis oznamující, že shoda nebyla nalezena. Diagram úlohy je na obrázku 4.



Obrázek 4: Diagram pro hledání duplicitních hodnot

Kód:

```
var
a: array [1..10] of integer;
i,j,n: integer;
p: boolean;
```

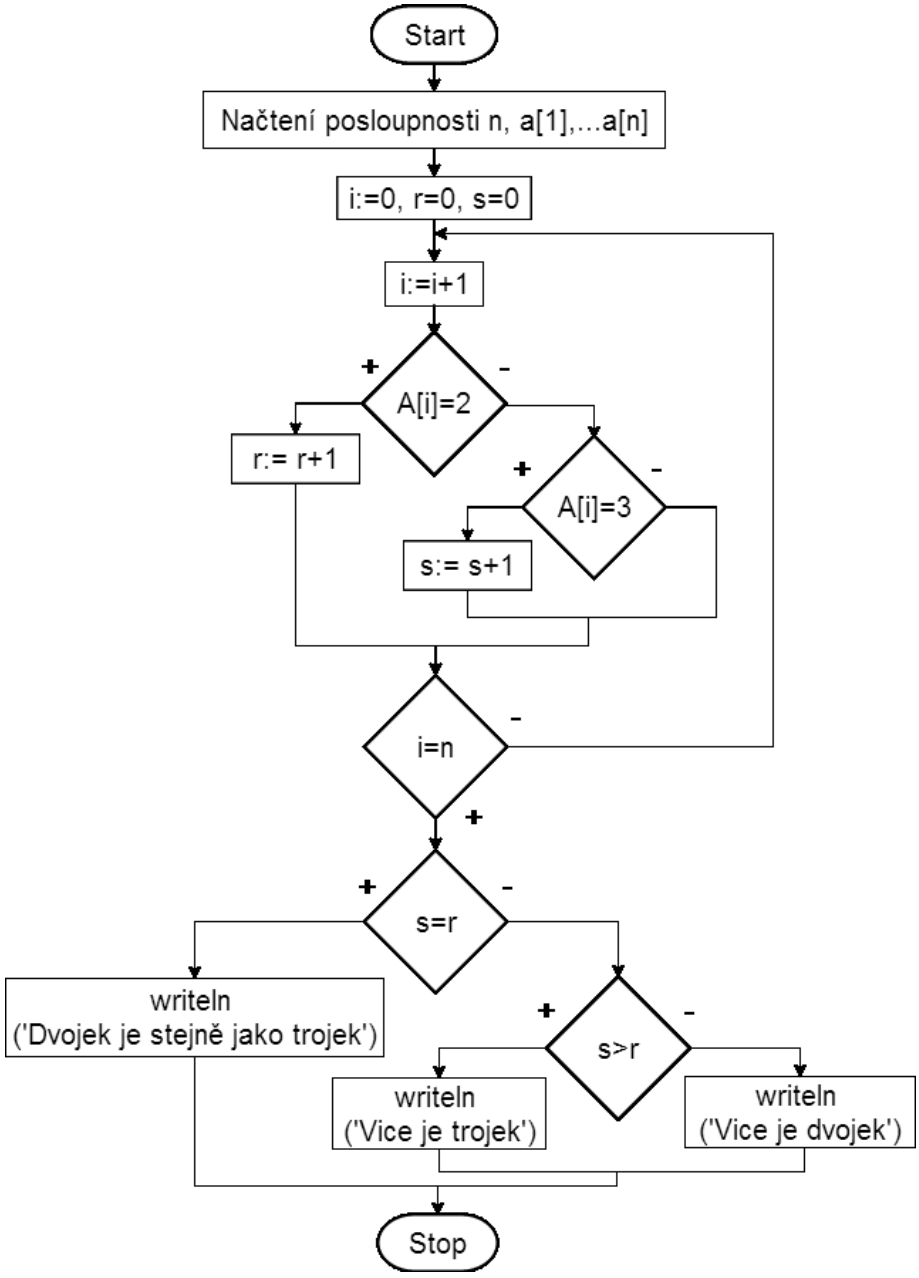
```
begin
read(n);
p:=true;
i:=0;
for i := 1 to n do
    read(a[i]);
i:=0;
repeat
    begin
        i:=i+1;
        j:=i;
            repeat
                begin
                    j:= j+1;
                    if a[i]=a[j] then p:=false;
                end;
            until ( (j=n) or (p=false) );
    end;
until ( (i=n-1) or (p=false) );
if p=false then writeln ('shoda') ;
if p=true then writeln ('neni shoda') ;
end.
```

2.4 Příklad 3:

Zadání: Zjistěte, zda se nachází v posloupnosti více dvojek nebo více trojek.

Cílem programu je projít celé pole a zaznamenat počet výskytů čísla 2 a 3. Program si nejprve načte počet prvků v poli a následně obsah samotného pole. Počet výskytů čísla 2 se bude ukládat do proměnné „r“ a počet výskytů čísla 3 do proměnné „s“. Proto do obou proměnných přiřadíme hodnotu

0. Program bude procházet prvky celého pole a každý prvek pole nejprve porovná s hodnotou 2. Pokud nastane shoda, proměnná „r“ se zvětší o hodnotu 1. Tímto je tato iterace cyklu u konce. Pokud se ale prvek neshodoval s hodnotou 2, následuje další porovnání, tentokrát s hodnotou 3. Pokud nastane shoda, proměnná „s“ se zvětší o hodnotu 1 a průchod cyklu opět končí. Pokud nenastala shoda ani v 1 případě, průchod cyklu končí bez toho, aby se zvětšila hodnota kterékoli proměnné. Po zpracování všech prvků v poli se porovnají hodnoty proměnných „r“ a „s“ a je proveden 1 z 3 podmíněných příkazů, který oznámí, že se v posloupnosti vyskytuje buď více dvojek, nebo více trojek, anebo že je jejich počet stejný. Diagram úlohy je na obrázku 5.



Obrázek 5: Diagram s výskytem dvojek a trojek

Kód:

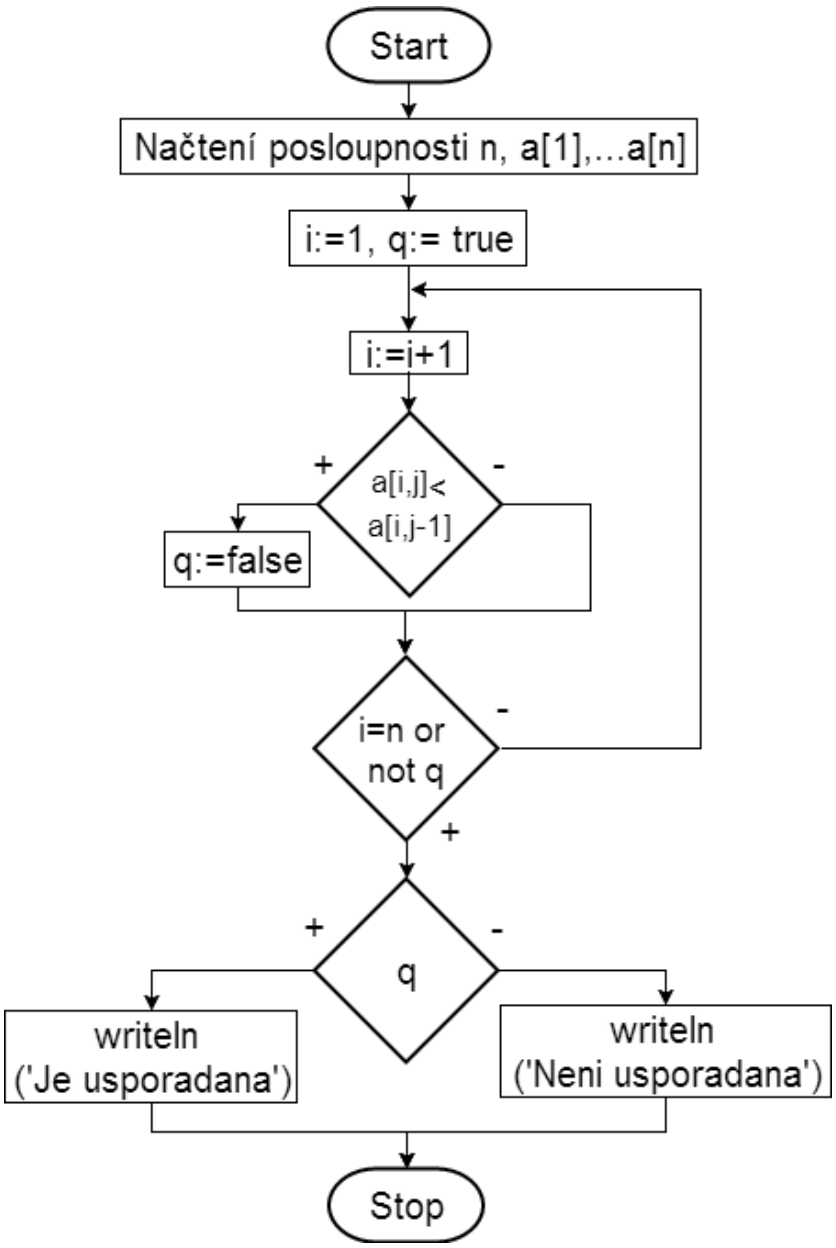
```
var
a: array [1..10] of integer;
i,n,r,s: integer;
begin
read(n);
r:=0;
s:=0;
for i := 1 to n do
    read(a[i]);
For i:=1 to n do
    begin
        if a[i]=2 then r:=r+1;
        if a[i]=3 then s:=s+1;
    end;
if r=s then writeln ('Dvojek je stejne jak trojek');
if r>s then writeln ('Dvojek je vice nez trojek');
if s>r then writeln ('Trojek je vice nez dvojek');
end.
```

2.5 Příklad 4:

Zadání: Zjistěte, zda posloupnost n načtených prvků je uspořádána vzestupně.

Výsledný program bude procházet dvojice sousedních prvků v posloupnosti a bude při tom zjišťovat, zda jsou uspořádány vzestupně (prvek na pozici i musí mít menší hodnotu, než prvek na pozici i+1). Pokud tato podmínka není u některé dvojice splněna, tak následuje výpis oznamující, že posloupnost není uspořádaná (i jedna neuspořádaná dvojice způsobí, že celá posloupnost není uspořádaná) a program končí. Pokud se programu podaří zpracovat celou posloupnost, tak to znamená, že je uspořádána vzestupně a následuje výpis, který tuto skutečnost oznámí.

Program si nejprve načte počet prvků, které bude posloupnost obsahovat, obsah samotné posloupnosti a do proměnné „q“, typu boolean, se uloží hodnota „true“. Následuje cyklus, který porovnává dvojice prvků. Tento cyklus pracuje vždy s prvkem, který je dán počítadlem a prvkem který mu předchází. Proto tento cyklus začíná v posloupnosti prvkem na 2. pozici (počítadlo „i“ obsahuje hodnotu 2), který je porovnáván s prvkem na 1. pozici (nemůžeme začít s prvkem na 1. pozici, protože program by ho chtěl porovnat s předchozím prvkem, který neexistuje a následovala by chybová hláška). Pokud je prvek na 1. pozici větší než prvek na 2. pozici (dvojice není uspořádaná), do proměnné „q“, typu boolean, se uloží hodnota „false“. Před každým dalším průchodem cyklu (zvýšením počítadla) program kontroluje hodnotu proměnné „q“ a pokud je „false“, tak následuje výpis oznamující, že posloupnost není uspořádaná a program končí. Pokud by byla tato dvojice uspořádána (1. prvek by byl menší než 2. prvek), počítadlo by se zvýšilo a program by pokračoval zpracováním následující dvojice. Jak už bylo řečeno, program pokračuje do nalezení neuspořádané dvojice, po které následuje výpis a konec programu, nebo do zpracování posledního prvku v posloupnosti (i=n), který značí, že posloupnost je uspořádána vzestupně, což oznámí výpis a program opět končí. Diagram úlohy je na obrázku 6.



Obrázek 6: Diagram s testem, zda je uspořádané pole

Kód:

```
var
a: array [1..10] of integer;
i,n: integer;
p: boolean;
begin
read(n);
p:=false;
for i := 1 to n do
```

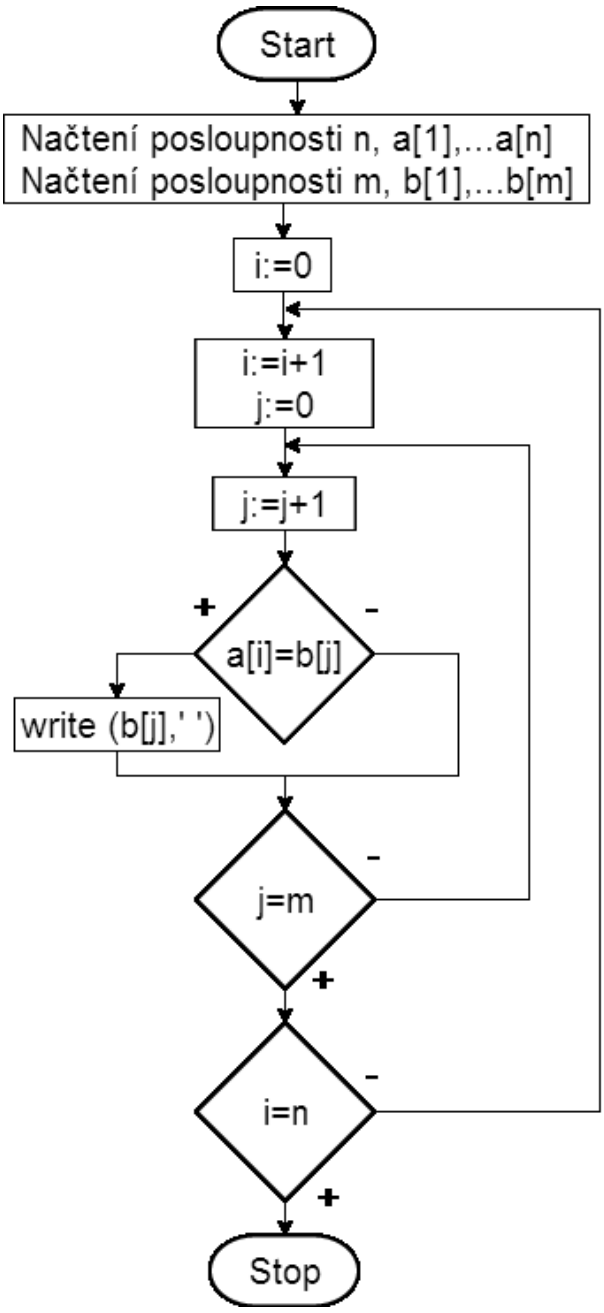
```
    read(a[i]);
i:=1;
repeat
    begin
        i:=i+1;
        if a[i-1]>a[i] then p:=true;
    end;
until ( (i=n) or (p=true) );
if p=true then writeln ('Posloupnost neni usporadana vzestupne')
          else writeln ('Posloupnost je usporadana vzestupne') ;
end.
```

2.6 Příklad 5:

Zadání: Vypište prvky, které se nacházejí v průniku 2 polí.

Cílem programu je porovnat každý prvek z 1. posloupnosti s každým prvkem z 2. posloupnosti a vypisovat při tom prvky, které se shodují.

Nejprve se načte počet prvků 1. posloupnosti, pak samotná 1. posloupnost, počet prvků 2. posloupnosti a 2. posloupnost. Program bude procházet pole pomocí 2 cyklů, vnitřního a vnějšího. Vnější cyklus bude používat počítadlo s proměnnou „i“ a bude procházet prvky 1. posloupnosti. Porovnání aktuálního prvku z této posloupnosti s každým prvkem posloupnosti druhé zabezpečí vnitřní cyklus, který bude používat pro počítadlo proměnnou „j“. Pokud se prvky shodují, jsou ještě v tomto průchodu cyklu vypsány na obrazovku. Po zpracování posledního prvku z 1. posloupnosti (tím pádem i 2. posloupnosti) program končí. Diagram úlohy je na obrázku 7.



Obrázek 7: Diagram s průnikem polí

Kód:

```
var
a: array [1..10] of integer;
b: array [1..10] of integer;
i,j,m,n: integer;
begin
read(m);
read(n);
```

```
writeln ('Prvky nachazejici se v pruniku poli ');
for i := 1 to n do
    read(a[i]);
for i := 1 to m do
    read(b[i]);
For i:= 1 to n do
    begin
        For j:= 1 to m do
            begin
                if a[i]=b[j] then writeln (b[j],' ');
            end;
    end;
end.
```

Domácí úkol: Napište program, který spočítá, kolikrát se v posloupnosti vyskytuje číslo 4.

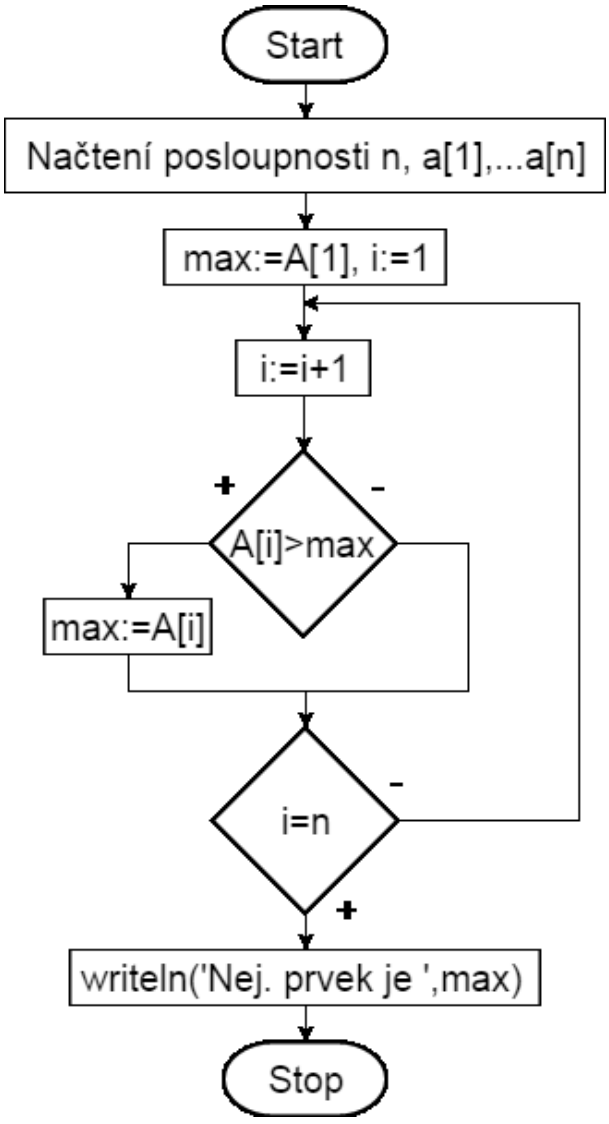
3 EXTRÉMY V 1 ROZMĚRNÉM POLI

Název:	Extrémy v 1 rozměrném poli
Cíl:	Žáci se seznámí s postupy, které lze aplikovat na hledání různých extrémů a jejich pozic v poli.
Motivace:	Žáci si zdokonalí svoje poznatky a uvědomí si, jak je daná tematika potřebná při zvládání náročnějších praktických struktur.
Příprava:	Kontrola funkčnosti dataprojektoru a počítačů v učebně.
Postup:	Promítnutí teorie a diagramů na plátno s postupným vysvětlováním postupu práce. Ve výkladu je prostor pro otázky žáků, jak o věcech, které nejsou jasné, tak o možných řešeních daných problémů. Na závěr následuje diskuze o možném budoucím využití prezentovaných programů. Zadání domácího úkolu.
Předpokládaný čas:	6 vyučovacích hodin
Cílová skupina:	3. ročník
Místo:	Počítačová učebna
Vyučovací metody a formy práce:	Výklad učitele Individuální práce Práce s informacemi Třídění a upevňování nových poznatků Společné vyhodnocení
Potřeby a materiál:	Dataprojektor a dostatečný počet počítačů připojených na školní síť.
Rozvoj KK:	KK řešení problémů KK k tvořivému myšlení KK pracovní KK k učení
Začlenění tematických okruhů PT:	Programování Jazyk Pascal
Vyhodnocení (reflexe):	Podpora myšlení žáků na různých úrovních. Žáci pochopí zpracovávání prvků pole ve složitějších strukturách. Žáci jsou schopni na základě daného problému vybrat vhodný postup k řešení. Žáci jsou schopni aplikovat prezentované postupy.

3.1 Příklad 1:

Zadání: Nalezněte a vytiskněte největší prvek z dané posloupnosti.

Program si do proměnné pro maximum – „max“ nejprve načte 1. prvek z posloupnosti. Následně se každý z prvků posloupnosti porovná s aktuální hodnotou v proměnné „max“. Pokud je větší, uloží se do této proměnné. Po zpracování všech prvků posloupnosti následuje výpis samotného maxima. Diagram úlohy je na obrázku 8.



Obrázek 8: Diagram s hledáním maxima

Kód:

```
var    n, i, max: Integer;
A: array [1..20] of Integer;
begin
writeln ('Zadejte pocet cisel v posloupnosti.');
```

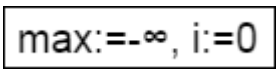
```
readln (n);
i:=1;
for i:= 1 to n do
   begin
         writeln ('Zadejte cislo.');
         readln(A[i]);
   end;
max:=A[1];
for i:= 2 to n do
    if A[i]>max then max:= A[i];
write('Nejvetsi cislo posloupnosti je ',max);
end.
```

Vzneseme polemiku nad účelem samotného algoritmu, zda by se nedal použít na nalezení minima, případně jak. Poslechneme si možné nápady a vyvodíme jednoduchý postup úpravy programu, po které bude program hledat minimum z dané posloupnosti. Stačí, abychom v příkazu typu „If“ změnili znaménko u porovnání i-tého prvku a aktuálního maxima z „>“ na „<“. Následně už jen provedeme formální úpravu výpisu a názvu proměnné pro minimum.

Úprava načtení pole

Program si do proměnné pro maximum načetl nejprve 1. prvek z posloupnosti a další prvky porovnával s hodnotou této proměnné. Tento postup se dá upravit do tvaru, kdy se do zmíněné proměnné pro maximum nejprve přiřadí nejnižší hodnot jejího datového typu – integeru. V jazyku Pascal tuto operaci provedeme příkazem „max := -MaxInt“, který do této proměnné uloží hodnotu -32 768. Při následném hledání maxima v posloupnosti porovnáváme po jednom všechny její prvky s hodnotou proměnné pro maximum, jako doposud, jen s tím rozdílem, že začínáme už s 1. prvkem v posloupnosti (v dosavadních příkladech jsme začínali až 2. prvkem). Upravený postup se může zdát z hlediska dosavadních programů krokem navíc, nicméně tato činnost výrazně zpřehlední budoucí rozvětvené algoritmy. V následujících vývojových diagramech bude tato spodní hraniční hodnota znázorněna symbolickým zápisem „-∞“.

V případě diagramu předcházejícího programu nastane jen malá změna v těle programu, kde se kromě proměnné „max“ upraví i počítadlo „i“, do kterého se uloží nula. Tato operace značí, že ještě nebyl načten žádný prvek z posloupnosti. Upravená část těla programu bude mít tento tvar:



Obrázek 9: Upravená část programu

Co se týká předcházejícího kódu, tak se kromě výše zmíněného příkazu pro přiřazení hraniční hodnoty do proměnné „max“ upraví jen druhý cyklus typu „For“, který porovnával vstupní hodnoty s maximem. Ten nebude začínat od 2. prvku pole, ale od 1. Výstup programu se tímto krokem nijak nezmění.

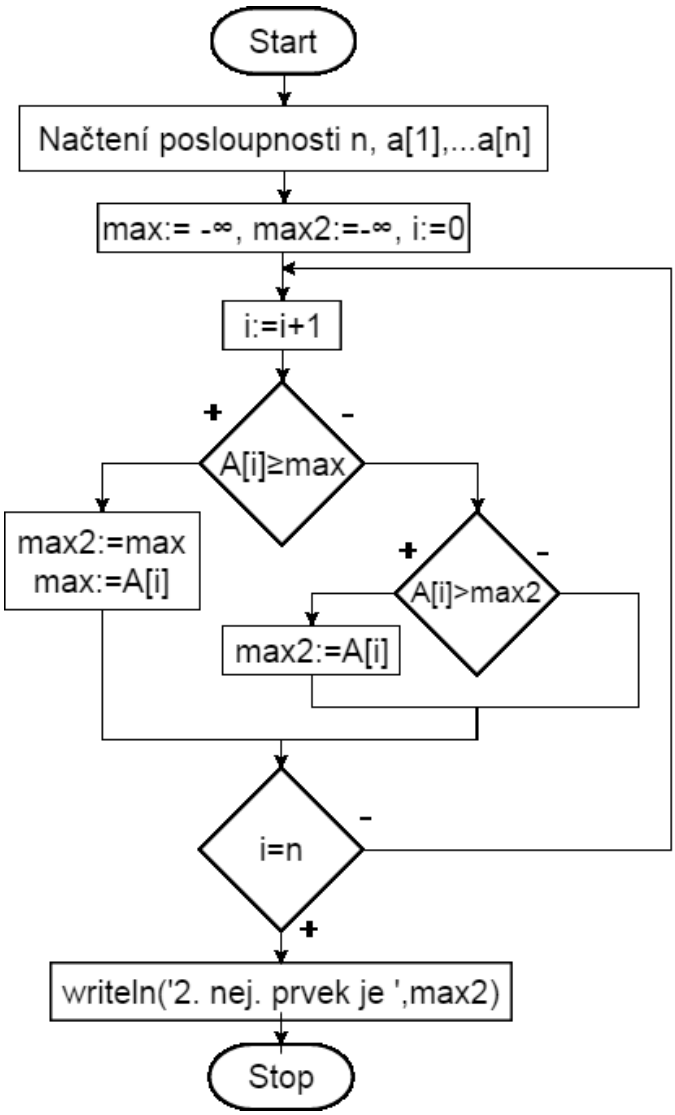
3.2 Příklad 2:

Zadání: Nalezněte a tiskněte 2. největší prvek z posloupnosti.

Budeme vycházet z předchozího příkladu, který si rozšíříme o proměnnou pro 2. nejvyšší prvek. Tu přidáme do operačního bloku, do kterého se dostaneme po nalezení nového maxima. Program vezme i-tý prvek z posloupnosti, porovná se s aktuální hodnotou v proměnné „max“ a nastává jedna z možností:

1. pokud je větší nebo roven, tak se do nové proměnné „max2“ nejprve uloží předchozí maximum a samotná proměnná „max“ se přepíše na nové maximum.
2. Pokud vstupní prvek není větší nebo roven předchozímu maximu, dostáváme se větvením do dalšího rozhodovacího bloku, kde proběhne obdobná operace, vstupní prvek se porovná s proměnnou, která obsahuje 2. nejvyšší prvek – „max2“. Pokud je prvek větší, uloží se do této proměnné.

Tímto procesem prochází opět všechny prvky posloupnosti a následuje výpis 2. největších prvků.



Obrázek 10: Diagram pro hledání 2. nej. prvku

Kód:

```
var n, i, max, max2 : Integer;
       A: array [1..20] of Integer;
begin
max:=-MaxInt;
max2:=-MaxInt;
writeln ('Zadejte pocet cisel v posloupnosti.');
   readln (n);
for i:= 1 to n do
   begin
         writeln ('Zadejte cislo.');
```

```
          readln(A[i]);
   end;
for i:= 1 to n do
    begin
          if A[i]>=max then
              begin
                  max2:=max;
                  max:= A[i];
              end
                                  else
                  if A[i]>max2 then max2:=A[i];
    end;
write('2. Nejvetsi cislo posloupnosti je ', max2);
end.
```

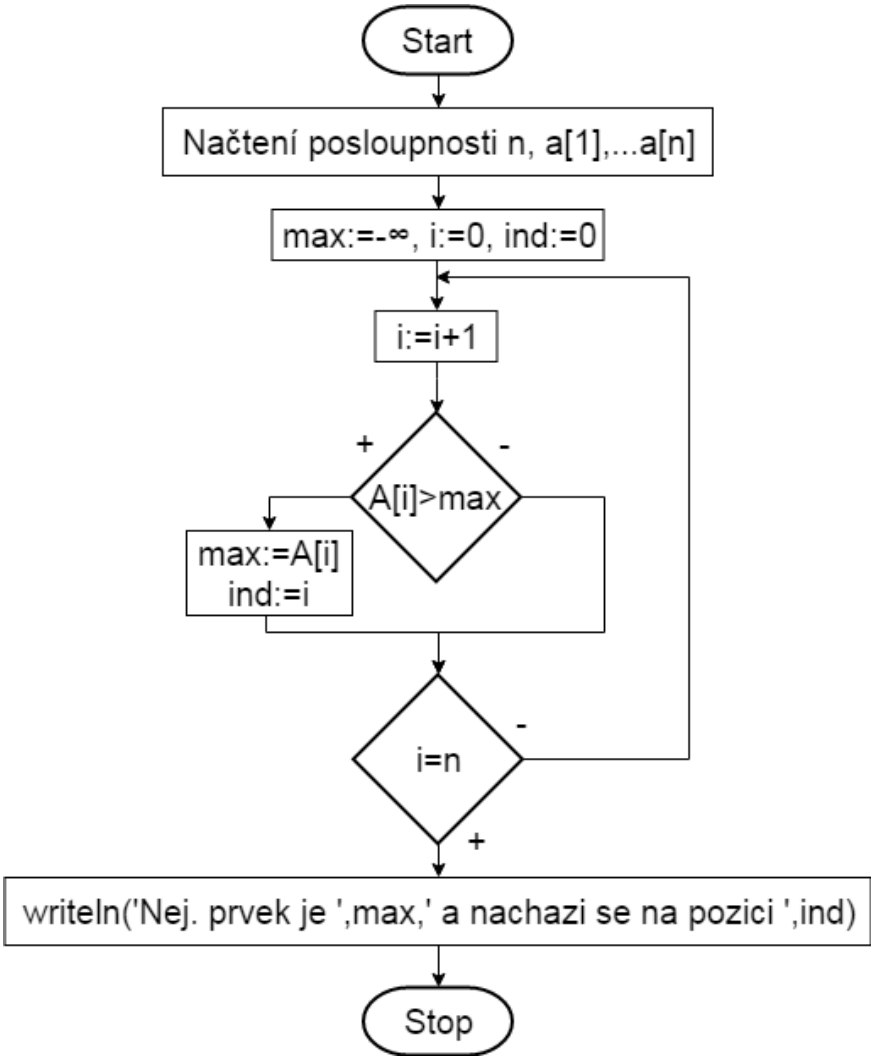
Strukturu uvedeného algoritmu lze teoreticky použít i pro hledání 3., 4., až i-tého extrému v posloupnosti. Je ale třeba vždy vycházet z algoritmu pro hledání extrému o jednu úroveň menšího, než aktuálně potřebujeme. Vždy je potřeba vytvořit novou proměnnou pro nový i-tý extrém, kterou přidáme do bloku, kde se „posouvají“ dosavadní extrémy při nalezení extrému nového a přidat další rozhodovací blok do větvící části pro případ, kdy je prvek na vstupu menší (větší) než aktuální extrém. Nicméně, tento postup se stává při hledání vyšších i-tých extrémů stále více neefektivní a proto je v těchto případech lepší zvolit metodu, kdy se nejprve posloupnost setřídí některým z třídících algoritmů a z výsledné posloupnosti se vypíše požadovaný i-tý extrémní prvek.

3.3 Příklad 3:

Zadání: Nalezněte a vytiskněte největší prvek i s jeho pořadím v rámci dané posloupnosti.

Vzneseme se žáky diskuzi na téma, jak by se dalo zjistit pořadí vybraného prvku. Po jednoduché úvaze předneseme řešení, kdy si do nové vytvořené proměnné číselného typu uložíme aktuální hodnotu proměnné „i“ u vybraného prvku, která slouží jako počítadlo.

Budeme vycházet z 1. příkladu, který si rozšíříme o proměnnou pro pořadí maxima. Tu přidáme do operačního bloku, do kterého se dostaneme po nalezení nového maxima. Tady se, jako v předchozím případě, uloží do proměnné „max“ hodnota tohoto maxima, a dále se do proměnné pro jeho pořadí – „ind“ uloží aktuální hodnota proměnné „i“ (počítadla). To je jediná praktická změna oproti 1. příkladu, po zpracování všech prvků posloupnosti následuje výpis maxima a jeho pořadí v posloupnosti.



Obrázek 11: Diagram pro hledání maxima i s pozicí

Kód:

```
var n, i, max, ind : Integer;
       A: array [1..20] of Integer;
begin
  max:=-MaxInt;
  writeln ('Zadejte pocet cisel v posloupnosti.');
   readln (n);
```

```
for i:= 1 to n do
   begin
         writeln ('Zadejte cislo.');
         readln(A[i]);
   end;
for i:= 1 to n do
    begin
         if A[i]>max then
             begin
                 max:= A[i];
                 ind:=i;
             end;
    end;
 write('Nejvetsi cislo posloupnosti je ', max, ' nachazejici se na
pozici ',ind);
end.
```

Vzneseme diskuzi nad předchozím programem a zaměříme se na otázku, zda je dostatečně univerzální pro různé vstupy. Po úvaze zjistíme, že v případě, kdy je ve vstupní posloupnosti více prvků obsahujících stejné maximum, je program schopen stále uchovat pouze pozici prvního maxima. Vzniká tedy otázka, jak tento problém vyřešit.

Při použití doteď probraných struktur by se pravděpodobně dala použít jen varianta, kde by se pro každý prvek posloupnosti vytvořilo vlastní pole, kde by se ukládala pozice všech prvků, obsahujících tuto hodnotu. Ve výsledku by se vypsal největší z těchto prvků spolu se svým polem (obsahujícím jeho pozice v posloupnosti, tzv. „indexy“). Tento postup by byl sice funkční, ale také by byl zbytečně náročný a především, z programátorského hlediska, by byl absolutně neefektivní, protože pro každý prvek posloupnosti by muselo být vytvořeno vlastní pole, s rozsahem odpovídajícím počtu prvků na vstupu programu pro případ, kdy by všechny prvky pole obsahovaly stejnou hodnotu. Všechny tyto pole by zabírala v paměti počítače zbytečně mnoho místa, ze kterého by stejně většina nebyla v praxi využita. Pro efektivní zvládnutí tohoto typu úlohy je vhodné použít specifickou variantu datového typu pole, tzv. dynamické pole.

3.3.1 Dynamické pole

Tento druh pole je charakteristický svou délkou, která není pevně daná, jak v případě dosavadních polí, ale může se libovolně měnit během vykonávání programu.

Deklarace dynamického pole probíhá deklaraci proměnných zápisem: „Var „název pole“ : array of integer“. Tímto vznikne dynamické pole bez určené délky a přidělené paměti. Dosud používaná pole, kde byla nejprve v deklaraci zadána délka, budeme odteď nazývat pole statická. Paměť je dynamickému poli přidělená (tzv. alokovaná) voláním funkce „SetLength(„název pole“, „počet prvků“)“. Pole je vždy indexováno od 0 (na rozdíl od statického pole, kde jsme si v deklaraci proměnných určili jeho rozsah, spolu s počátečním a konečným prvkem). Pokud použijeme příkaz „SetLength(Pole, 3)“, tak nově vzniknuté pole bude mít 3 prvky označeny jako:

1. Pole[0],
2. Pole[1],
3. Pole[2].

K prvkům pole se postupuje stejně, jako v případě statického pole, indexováním. Doteď jsme pro vypsání prvků pole používali cyklus „For“, který probíhal od prvku 1 po požadovaný počet prvků n, což je v docela přesné analogii s lidským myšlením. Pokud chceme tento cyklus použít pro dynamické pole, musíme snížit hodnoty počítadla „i“ o 1, takže zápis bude mít následující tvar: „For i := 0 to n-1 do“. Kromě této změny funguje cyklus jako doposud, jen je potřeba brát ohled na dané počítadlo. V praxi ale často neznáme aktuální délku daného dynamického pole, a proto je předchozí formulace příkazu prakticky nepoužitelná. Pro tuto situaci slouží příkaz „Length(„název pole“)“, který vrátí aktuální délku daného pole. Příklad: Máme existující pole vytvořeno příkazem „SetLength(Pole, 8)“, následný příkaz „writeln (Length(Pole))“ vypíše na obrazovku hodnotu 8. Tento příkaz budeme používat zejména při výpisu prvků pole, kterého neznáme aktuální délku. Zápis bude mít tvar: „For i := 0 to Length(„Název pole“)-1 do write(„Název pole“[i], ' ')“, a ve výsledku vypíše všechny prvky daného dynamického pole. [4]

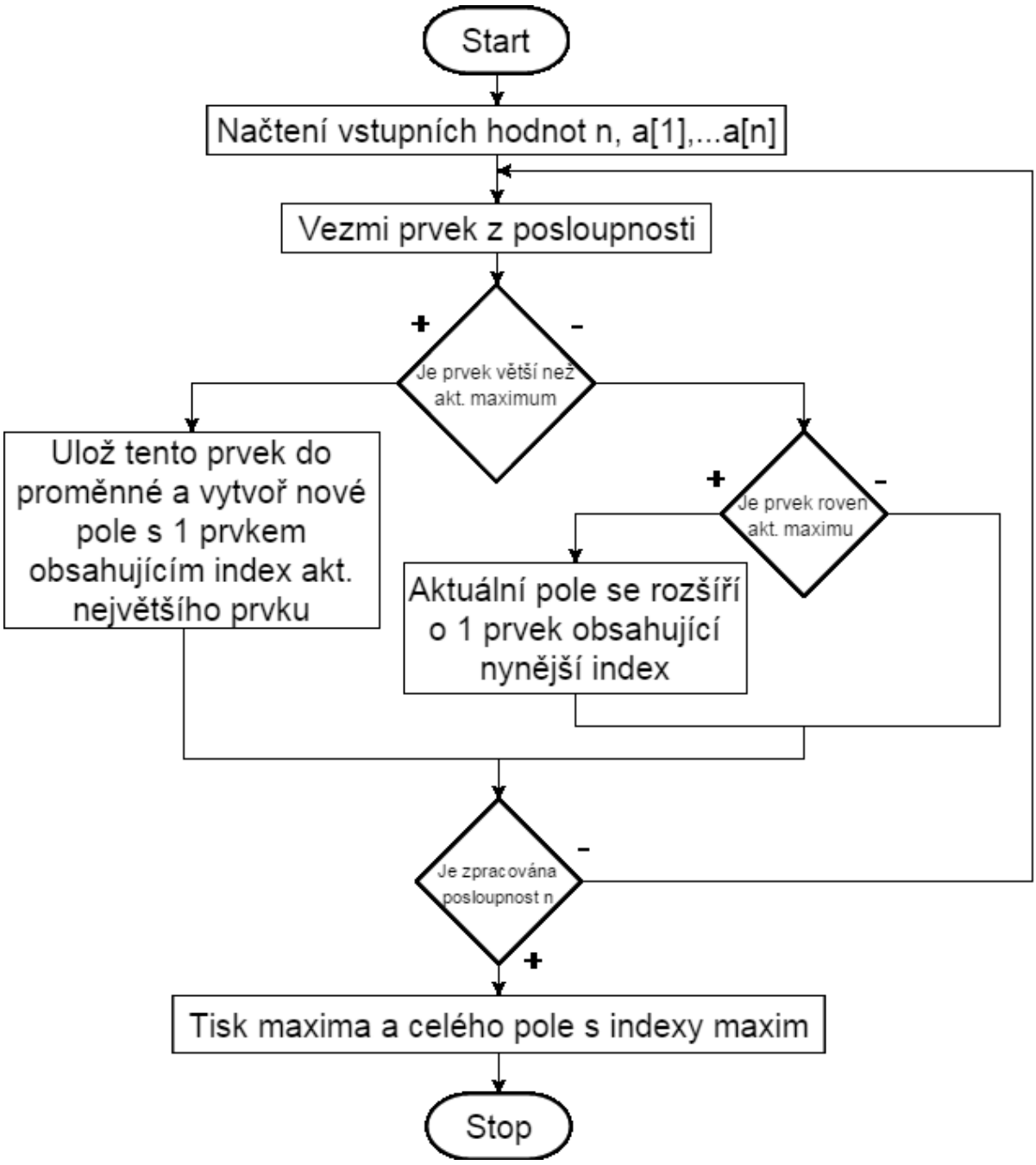
Zvláštním atributem tohoto příkazu je možnost upravit délku už existujícího dynamického pole na určitý počet prvků. Pokud máme existující dynamické pole s názvem „Pole“ obsahující 3 konkrétní prvky, tak použitím příkazu „SetLength(Pole, 7)“ dostáváme pole se stejným názvem, ale s novou délkou 7 prvků (původní tři prvky přitom zůstaly nezměněny na svém místě). Proces funguje i obráceně, pokud na existující pole o délce 7 prvků aplikujeme příkaz „SetLegth(Pole,2)“, tak ve výsledku dostáváme toto pole o délce 2 prvků (obsahující 2 původní prvky, stále na svých místech).

Teď se vrátíme k předešlé úloze, kde máme za úkol vypsat maximum i s pořadím všech prvků, ve kterých se nachází. Program je potřebné vytvořit tak, aby si v případě nalezení nového maxima toto

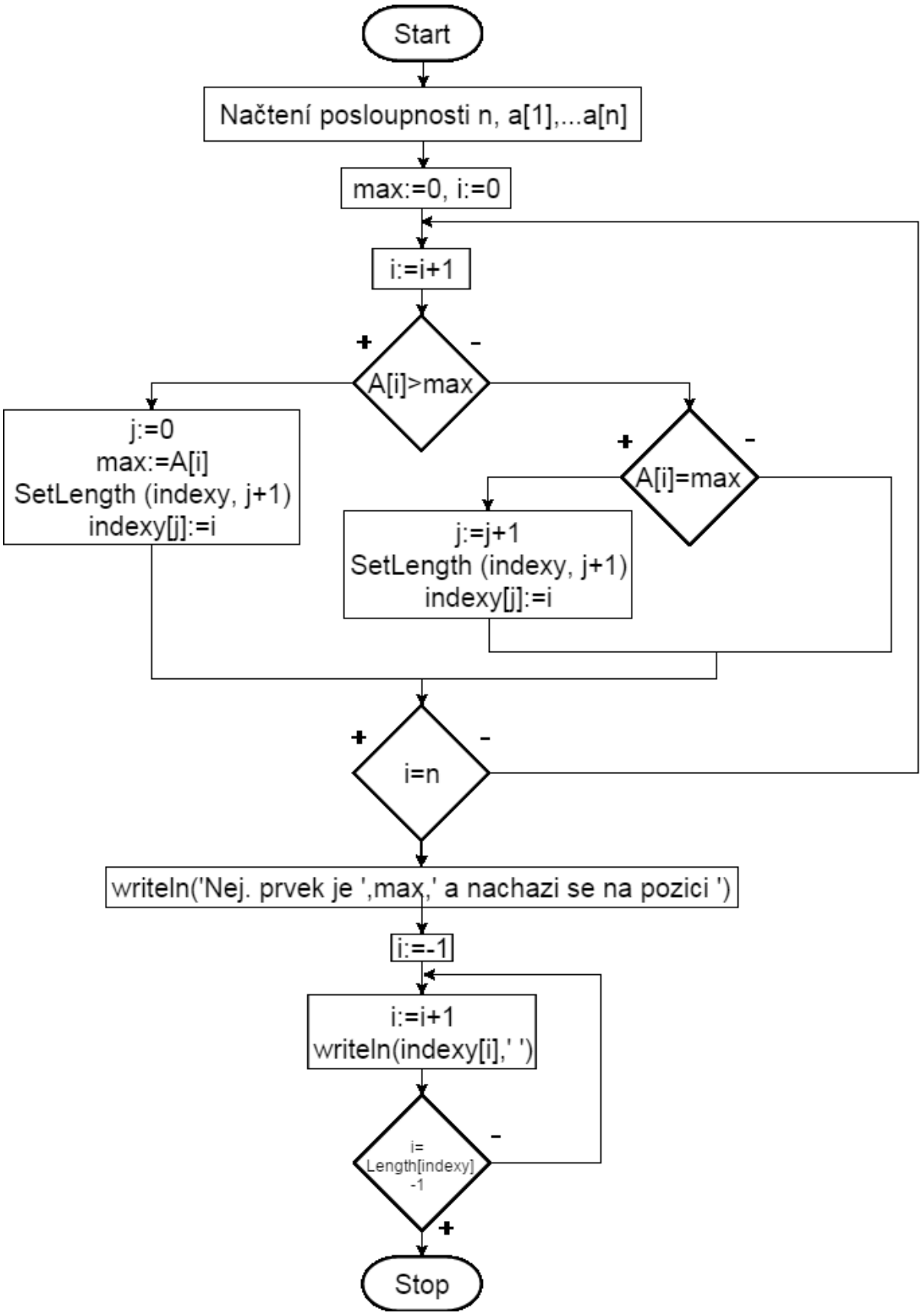
maximum uložil a vytvořil dynamické pole, do kterého uloží aktuální pozici tohoto prvku v posloupnosti. První číslo na vstupu (bereme v úvahu číslo z rozsahu integeru, které je větší než spodní hranice tohoto datového typu) si tedy program uloží do proměnné „max“. Proměnná „j“ se nastaví na 0 a vzápětí se vytvoří pole o velikosti j+1, takže pro 1 prvek. Tento prvek má index „j“ (tedy „0“) a uloží se do něj aktuální hodnota proměnné „i“ (pořadí). Při vstupu jakéhokoliv dalšího prvku (z množiny integeru) nastává vždy 1 z 3 situací:

1. Vstupní prvek je větší než maximum uložené v proměnné „max“: Tady se opakuje se předchozí proces – nové maximum se zaznamená a předcházející pole je přepsáno novým o délce 1 prvku s indexem prvku s novým maximem.
2. Vstupní prvek má stejnou hodnotu jako aktuální maximum: Aktuální dynamické pole je nutné rozšířit o prvek, do kterého je uložena pozice zpracovávaného prvku. Nejprve se tedy proměnná „j“, potřebná především pro index maxima, zvětší o hodnotu 1 a u aktuálního dynamického pole s indexy nejvyšších prvků je nastavena nová velikost na j+1 (tím je rozšířeno o 1 prvek). Následným zápisem „indexy[j]:=i“ se do tohoto pole uloží index aktuálního maxima. Tímto krokem se tedy jen zaznamenal index prvku s maximem, které už máme v programu uloženo.
3. Pokud vstupní prvek není větší ani roven aktuálnímu maximu, tak se nijak nezpracuje a pokračuje se do dalšího kroku, kde se načte další prvek z posloupnosti.

Po zpracování všech prvků posloupnosti následuje výpis maximálního prvku a pole obsahující indexy prvků, ve kterých se dané maximum nachází. Strukturovaný vývojový diagram na obrázku 12 nám bez dořešení některých dílčích problémů zachycuje zvolený postup.



Obrázek 12: Struktura algoritmu pro maximum s duplicitní pozicí



Obrázek 13: Diagram pro hledání maxima s duplicitní pozicí

Kód:

```
var n, max, i, j : Integer;
```

```
       A : array [1..40] of Integer;
       indexy: array of Integer;
begin
max := -MaxInt;
 writeln ('Zadejte pocet cisel v posloupnosti.' );
   readln (n);
for i:= 1 to n do
   begin
       writeln ('Zadejte cislo.');
       readln(A[i]);
    end;
for i:= 1 to n do
        begin
                    if A[i] > max then
                       begin
                            j := 0;
                            max := A[i];
                            SetLength(indexy, j+1);
                            indexy[j] := i;
                       end
                else if A[i] = max then
                       begin
                            j := j + 1;
                            SetLength(indexy, j+1);
                            indexy[j] := i;
                       end;
        end;
```

```
 writeln('Nejvetsi cislo posloupnosti je ', max, ' nachazejici se
na pozici(ch) ');
       for i := 0 to Length(indexy)-1 do
          write(indexy[i], ' ');
end.
```

Proces, kdy do programu vstupuje nové maximum, staré pole s indexy předchozího maxima se smaže a je vytvořeno pole nové s indexem nového maxima, zaručuje, že vždy pracujeme právě s jedním dynamickým polem, které obsahuje jen aktuální hodnoty indexů, a tím pádem maximálně šetříme paměť počítače.

3.4 Přiklad 5:

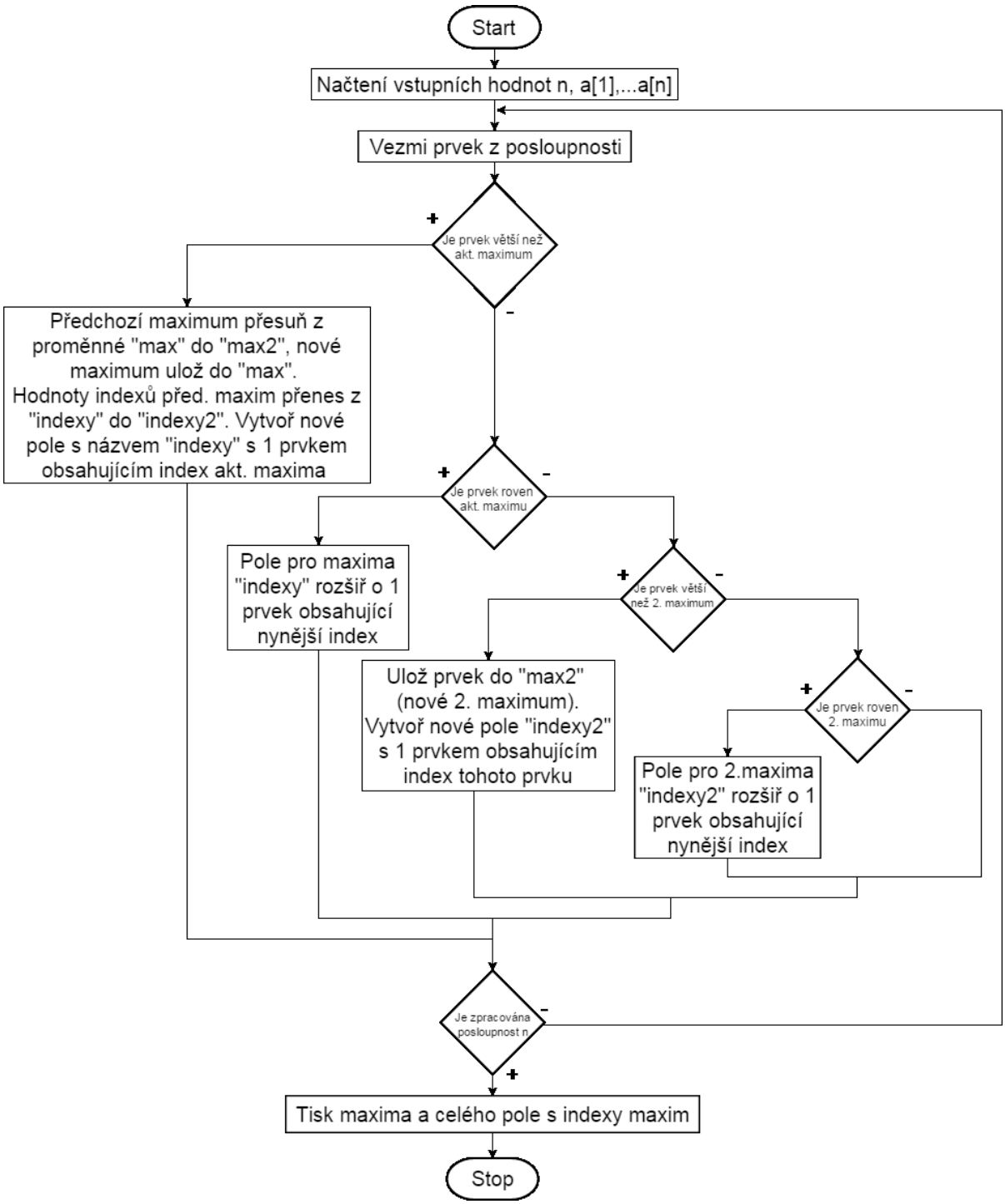
Zadání: Nalezněte a tiskněte 2. největší prvek a jeho pozici v dané posloupnosti.

Funkce předchozích algoritmů můžeme spojit dohromady při tvorbě algoritmu nového, který je schopen vypsat pořadí 2. nejvyššího prvku. Jednotlivé dílčí části z předchozích algoritmů se napojují na sebe poměrně analogicky, nicméně je potřeba přidat několik dodatečných kroků. Nejprve program rozšíříme o proměnnou pro 2. nejvyšší prvek, dynamické pole pro jeho indexy a nové „počítadlo“ k tomuto poli. Všeobecně je program potřebné přizpůsobit na 5 různých situací, které můžou nastat v závislosti od hodnoty vstupního prvku (možné situace jsou řazeny v pořadí, ve kterém musí v programu probíhat rozhodnutí o vykonání každé z nich):

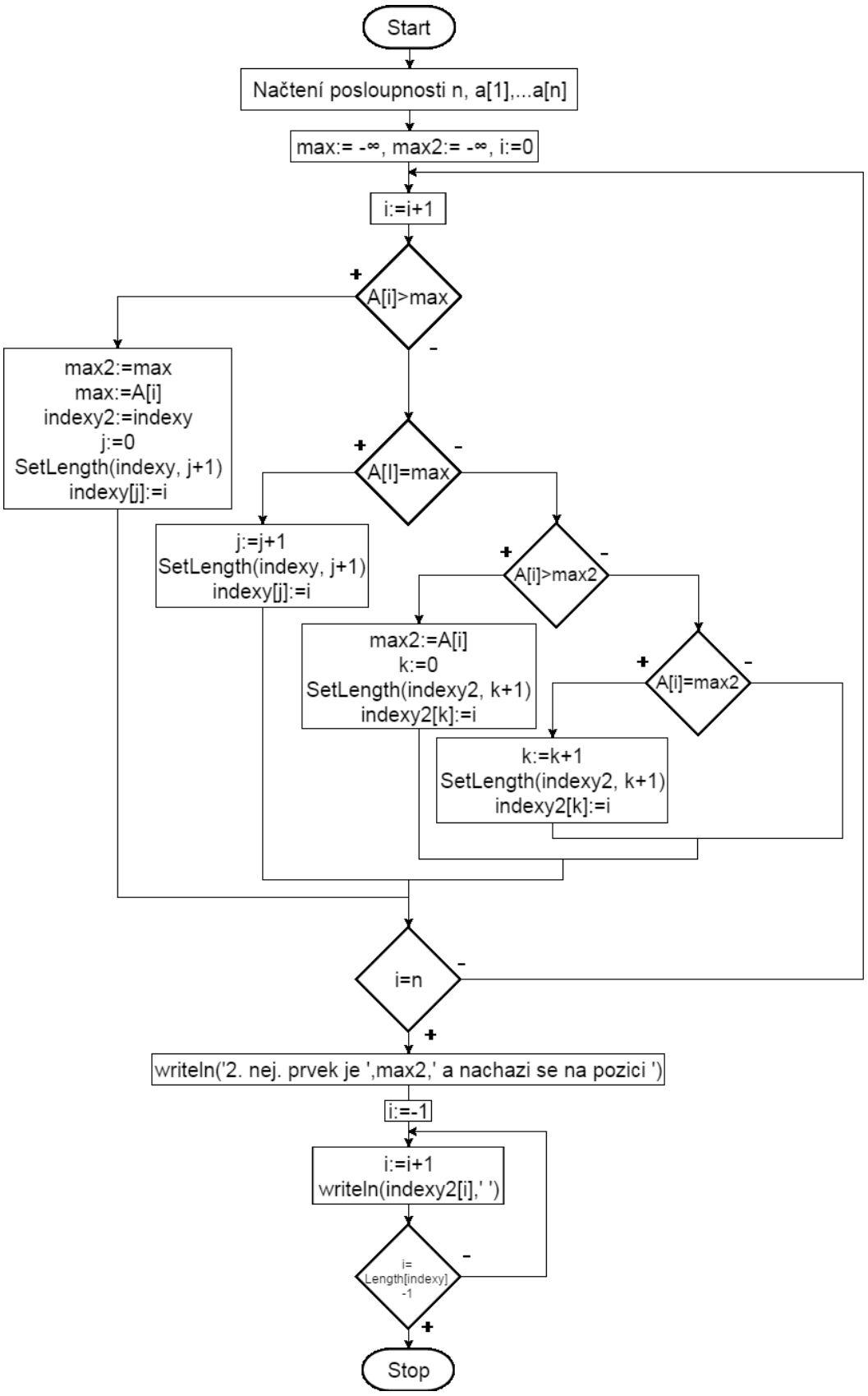
1. Vstupní prvek je větší než aktuální maximum: Dosavadní maximum se „posune“ do proměnné pro 2. maximum, nové maximum se uloží do původní proměnné pro maximum, aktuální pole pro indexy maxim se zkopíruje do nového pole a vytvoří se nové pole pro indexy maxim o délce 1 prvku, které obsahuje index aktuálního prvku. Po provedení tohoto kroku máme uloženo nové i předcházející maximum s jejich pozicemi (v polích).
2. Vstupní prvek se rovná aktuálnímu maximu: Aktuální pole pro maxima se rozšíří o prvek s indexem aktuálního prvku. Tímto krokem si jen zaznamenáme index prvku s maximem, které už máme v programu uloženo.
3. Vstupní prvek je větší než aktuální 2. maximum: Tento prvek se uloží do původní proměnné pro 2. maximum, aktuální pole s jeho indexy se přepíše polem o délce jednoho prvku, které obsahuje index aktuálního prvku. Tímto krokem si uložíme nové 2. největší maximum i s jeho pozicí.

4. Vstupní prvek se rovná aktuálnímu maximu: Aktuální pole pro 2. nejvyšší maxima se rozšíří o 1 prvek s indexem aktuálního prvku. Tímto jsme si zaznamenali index prvku s 2. nej. maximem, které už máme v programu uloženo.
5. Ani jedna z možností výše: žádná operace s aktuálním prvkem se nevykoná a pokračuje se k dalšímu kroku, kde se načte další prvek z posloupnosti.

Hrubý tvar vývojového diagramu na obrázku 14 popisuje výše uvedený rozbor úlohy.



Obrázek 14: Struktura algoritmu pro hledání 2. nej. prvku s duplicitní pozicí



Obrázek 15: Diagram algoritmu pro hledání 2. nej prvku s duplicitní pozicí

Kód:

```
var n, max, max2, i, j,k : Integer;
       A : array [1..40] of Integer;
       indexy, indexy2: array of Integer;
begin
max := -MaxInt;
max2 := -MaxInt;
 writeln ('Zadejte pocet cisel v posloupnosti.' );
   readln (n);
for i:= 1 to n do
   begin
       writeln ('Zadejte cislo.');
       readln(A[i]);
    end;
for i:= 1 to n do
        begin
                    if A[i] > max then
                       begin
                            max2:=max;
                            max := A[i];
                            indexy2:= indexy;
                            j := 0;
                            SetLength(indexy, j+1);
                            indexy[j] := i;
                       end
               else if A[i] = max then
                       begin
                            j := j + 1;
```

```
                    SetLength(indexy, j+1);
                    indexy[j] := i;
                end
            else if A[i] > max2 then
                begin
                    max2:=A[i];
                    k := 0;
                    SetLength(indexy2, k+1);
                    indexy2[k] := i;
                end
            else if A[i] = max2 then
                begin
                    k := k + 1;
                    SetLength(indexy2, k+1);
                    indexy2[k] := i;
                end
        end;
 writeln('2. nejvetsi cislo posloupnosti je ', max2, ' nachazejici se na pozici(ch) ');
       for i := 0 to Length(indexy2)-1 do
          write(indexy2[i], ' ');
end.
```

Domácí úkol: Napište program, který zjistí, zda se ve vstupní posloupnosti nachází prvek (y) s hodnotou 5, pokud ano, vypište jeho pozice.

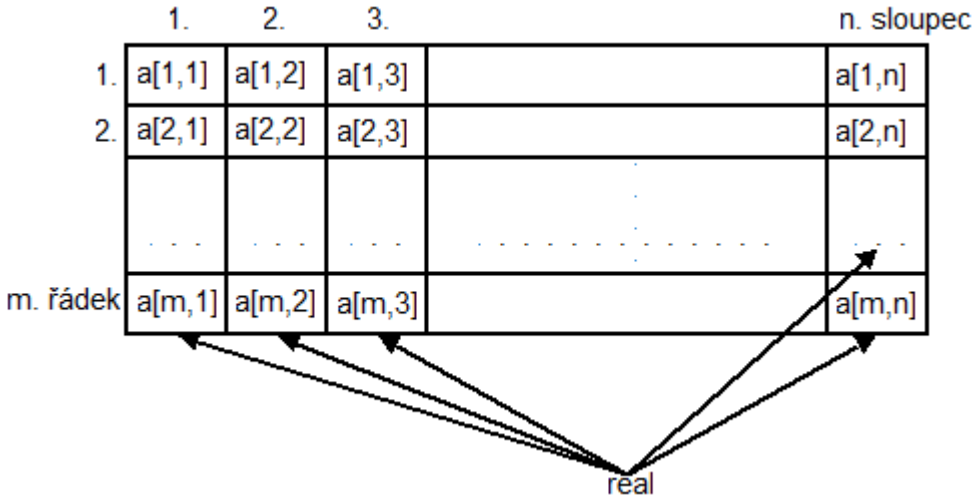
4 2 ROZMĚRNÉ POLE

Název:	2 rozměrné pole
Cíl:	Žáci se seznámí s 2 - rozměrným typem polem, s postupy, které lze aplikovat lokálně na jednotlivé řádky a sloupce.
Motivace:	Žáci si zdokonalí svoje poznatky a uvědomí si, jak je daná tematika potřebná při zvládání náročnějších praktických struktur.
Příprava:	Kontrola funkčnosti dataprojektoru a počítačů v učebně
Postup:	Promítnutí teorie a diagramů na plátno s postupným vysvětlováním postupu práce. Ve výkladu je prostor pro otázky žáků, jak o věcech, které nejsou jasné, tak o možných řešeních daných problémů. Zadání domácího úkolu.
Předpokládaný čas:	6 vyučovacích hodin
Cílová skupina:	3. ročník
Místo:	Počítačová učebna
Vyučovací metody a formy práce:	Výklad učitele Individuální práce Práce s informacemi Třídění a upevňování nových poznatků Společné vyhodnocení
Potřeby a materiál:	Dataprojektor a dostatečný počet počítačů připojených na školní síť.
Rozvoj KK:	KK řešení problémů KK k tvořivému myšlení KK pracovní KK k učení
Začlenění tematických okruhů PT:	Programování Jazyk Pascal
Vyhodnocení (reflexe):	Podpora myšlení žáků na různých úrovních. Žáci pochopí, jak program zpracovává prvky pole podle různých kritérií. Žáci jsou schopni na základě daného problému vybrat vhodný postup k řešení. Žáci jsou schopni aplikovat prezentované postupy.

4.1 2 rozměrné pole

2 rozměrné pole je datová struktura, kterou opisuje definice od doc. Botka: „Ve 2 rozměrném poli je každý prvek určen dvojicí indexů, z nichž první určuje řádek a druhý sloupec prvku. Zápis $a[i,j]$ určuje tedy prvek matice a na i-tém řádku a j-tém sloupci.“ [3, s. 46]

Obrázek 16 zachycuje matice „a“, obsahující počet řádků „m“, počet sloupců „n“ typ složky – real.



Obrázek 16: 2 rozměrné pole [3]

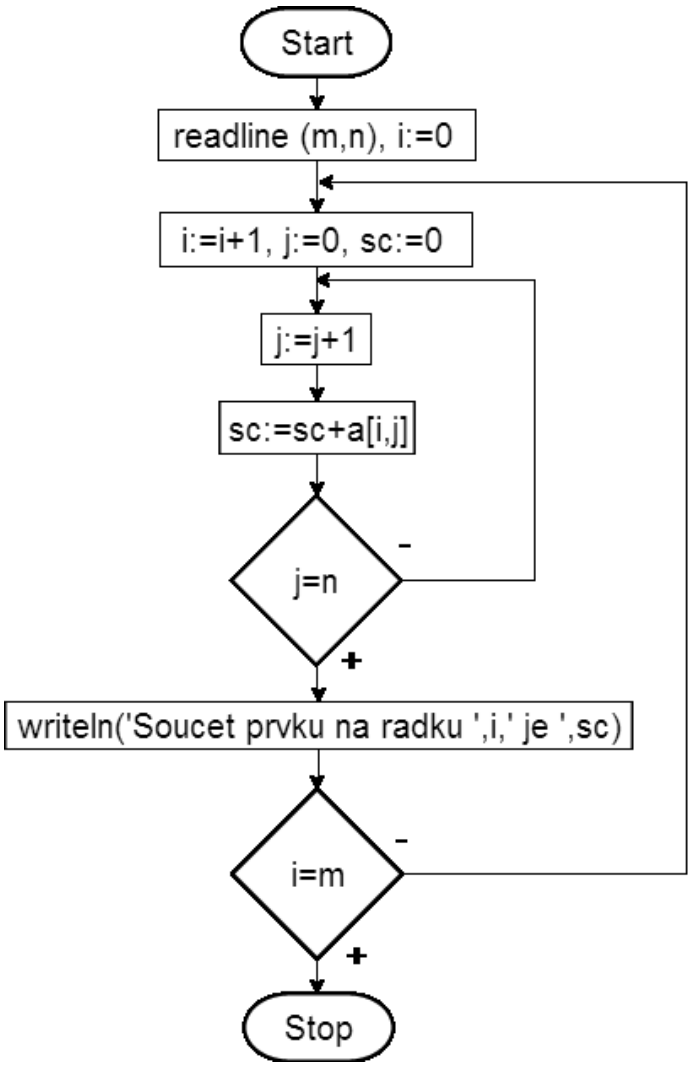
Příklad deklarace 2 rozměrného pole:

```
a : array [1..m,1..n] of real;
```

4.2 Příklad 1:

Zadání: Vypište součty prvků v jednotlivých řádcích

Zamyslíme se s žáky nad daným zadáním a po úvaze předneseme řešení, kterým je program obsahující vnitřní cyklus (pro zpracování jednotlivých prvků v řádku) a vnější cyklus (pro zpracování samostatných řádků). Ve vnitřním cyklu program prochází celým řádkem prvek po prvku (zvyšuje se počítadlo „i“) a jejich hodnotu ukládá do jedné proměnné. Po zpracování celého řádku cyklus skončí a zobrazí výpis čísla řádku s hodnotou proměnné pro součet prvků. Vnější cyklus zabezpečí proběhnutí předchozího cyklu i s výpisem pro každý řádek v poli. Nejprve připraví pro vnitřní cyklus podmínky- zvýší se počítadlo pro jednotlivé řádky „j“, do počítadla „i“ se uloží 0 – protože budeme procházet nový řádek od začátku a do proměnné pro součet prvků se také uloží 0. Poté proběhne samotný vnitřní cyklus (zpracují se všechny prvky v řádku) a následuje výpis. Vnější cyklus se následně opět posune na další řádek. Po zpracování posledního řádku v poli vnější cyklus, i celý program, skončí.



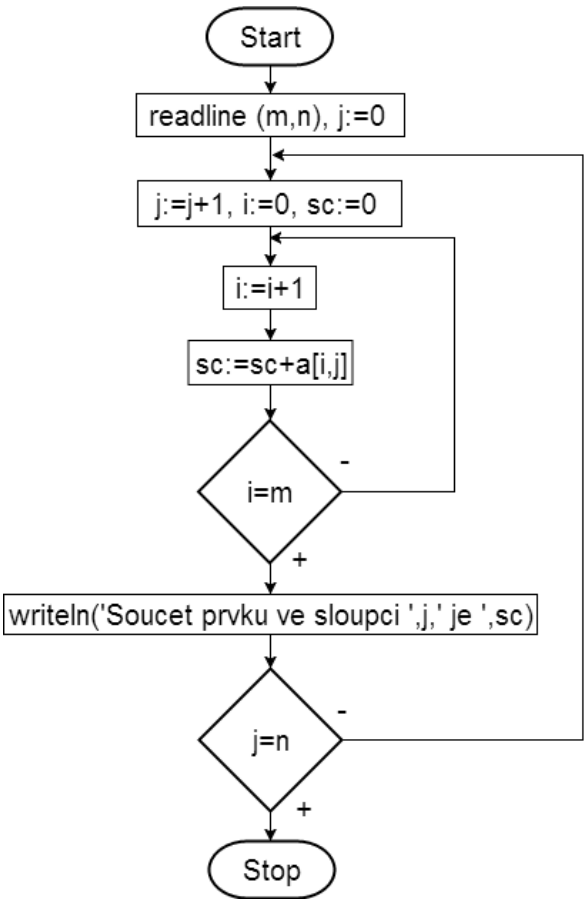
Obrázek 17: Diagram pro součty na řádcích

Kód:

```
var
a: array [0..20,0..20] of integer;
i,j,m,n,sc: integer;
begin
read(m);
read(n);
for i := 1 to m do
    for j := 1 to n do
        read(a[i][j]);
for i := 1 to m do
    begin
```

```
        sc:=0;
        for j := 1 to n do
            begin
                sc:=sc+a[i][j];
            end;
            writeln('Soucet na radku ',i,' je ',sc);
    end;
end.
```

Tento program lze upravit, aby vypisoval součty prvků v jednotlivých sloupcích. Cílem této úpravy je, aby program pracoval místo v řádcích, ve sloupcích. To znamená, že se ve vnitřním cyklu budou zpracovávat prvky z jednotlivých sloupců a ve vnějším cyklu budou zpracovávány sloupce. V 1. kroku je třeba upravit vnitřní cyklus - místo řádkového počítadla „i“ bude sloupcové počítadlo „j“. Díky tomu program bude zpracovávat prvky v daném sloupci. Výpis formálně upravíme a proměnnou „i“ nahradíme proměnnou „j“. Ve 2. kroku upravíme vnější cyklus, který opět zabezpečuje, aby předchozí cyklus i s výpisem proběhl pro každý řádek v poli. Takže místo zvýšení počítadla „j“ zvýšíme počítadlo „i“ (tímto se program posune do dalšího sloupce), a dále namísto „i“ vynulujeme počítadlo „j“ (každý nový sloupec zpracováváme od začátku). Poté proběhne samotný vnitřní cyklus (zpracují se všechny prvky v řádku) a následuje výpis. Nový program tedy obsahuje vnitřní cyklus, kde se se zpracují prvky v jednotlivém sloupci, nastane výpis a vnější cyklus se posune na další sloupec. Po zpracování posledního sloupce v poli vnější cyklus, a tím pádem i celý program, skončí.



Obrázek 18: Diagram pro součty ve sloupcích

4.3 Příklad 2:

Zadání: Součty prvků na diagonálách

Sečíst prvky na diagonálách můžeme jen u těch typů 2 rozměrných polí, které mají prvky rozmístěny rovnoměrně podél diagonál. To znamená, že pro tento účel jsou vhodné jen tzv. čtvercová pole. Čtvercové pole se vyznačuje tím, že má stejný počet řádků a sloupců. Toto pole má 2 diagonály, tzv. hlavní a vedlejší. Hlavní diagonála je tvořena prvky, které leží na přímce z levého horního rohu do pravého spodního rohu pole (počáteční prvek má souřadnice [1,1] a konečný prvek [m,m]). Prvky vedlejší diagonály leží na přímce z pravého horního rohu do levého spodního rohu (souřadnice počátečního prvku jsou [1,m] a konečného [m,1]). Potřebujeme tedy program, který v 1. části uloží součet prvků z hlavní diagonály do proměnné, ve 2. části uloží součet prvků z vedlejší diagonály do další proměnné, obě proměnné následně vypíše a zbytek pole ho nebude zajímat.

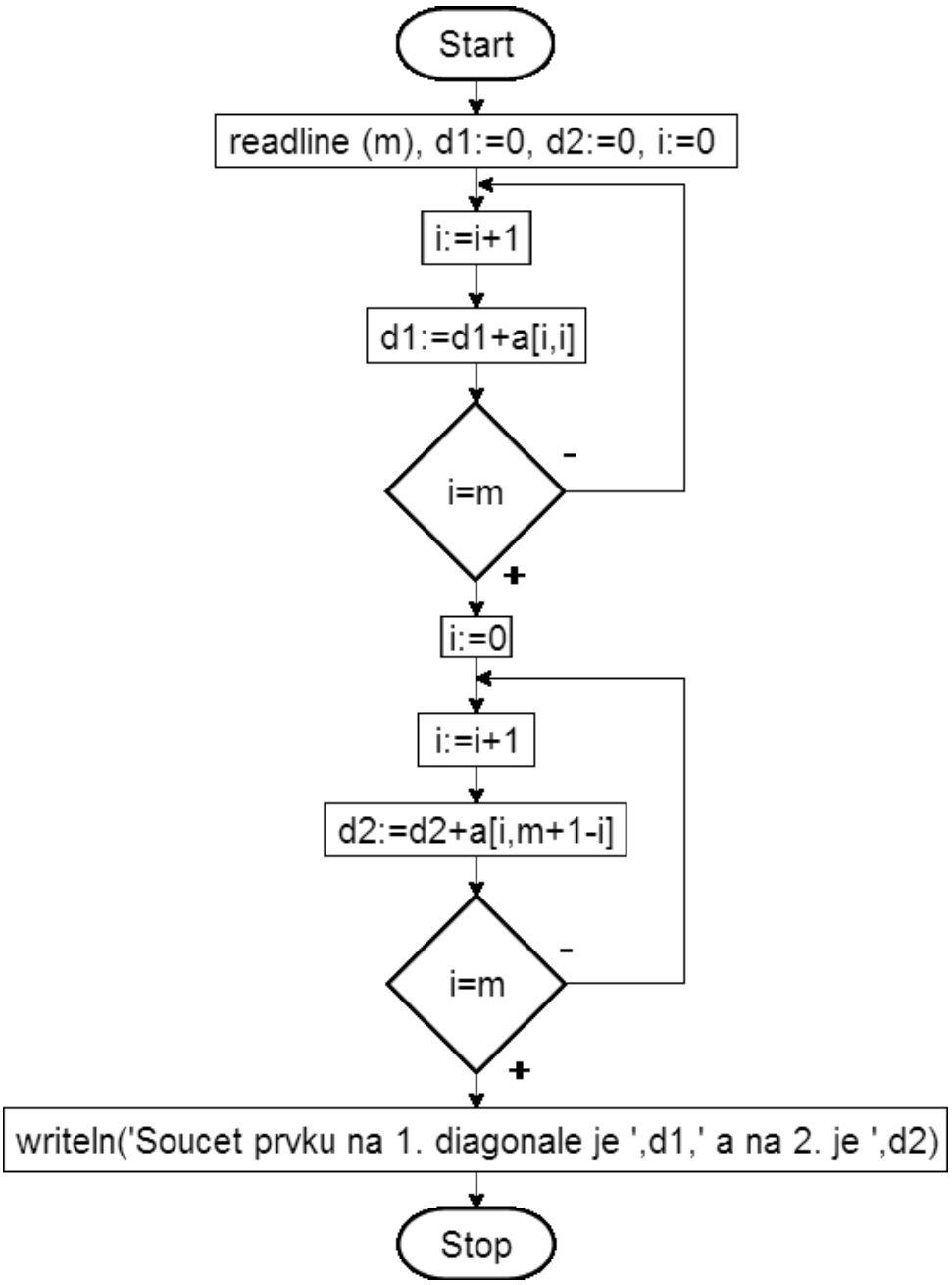
V první části si nejprve vytvoříme proměnnou typu integer pro součet. Všechny prvky na hlavní diagonále mají stejnou vlastnost, kterou je, že jejich X- ová souřadnice je stejná, jako Y- ová. Potřebujeme tedy sečíst prvek se souřadnicemi [1,1] s prvkem se souřadnicemi [2,2], [3,3] až [m,m]. Vytvoříme proto cyklus, který bude ukládat do proměnné pro součet prvky se souřadnicemi podle vzorce

$$d1 = d1 + a[i, i]$$

„d1" je proměnná pro součet hodnot, „i“ je počítadlo pro souřadnice, které bude nabývat hodnoty v rozsahu od 1 až po „m“ („m“ je délka pole) a „a“ je identifikátor označující pole, ve kterém pracujeme. Výsledný cyklus tedy sečte hodnoty prvků podél této diagonály. S vedlejší diagonálou to ale bude složitější. Zkusíme se žáky polemizovat, podle kterých vlastností by mohl program identifikovat prvky z vedlejší diagonály. Následně předneseme řešení, kterého podstata je cyklus, který bude ukládat do proměnné hodnoty prvků podle souřadnicového vzorce

$$d2 = d2 + a[i, m + 1 - i]$$

„d2“ označuje opět proměnnou pro součet prvků, „i“ je počítadlo, které bude nabývat hodnoty v rozsahu od 1 až po „m“ („m“ označuje délku pole) a „a“ jako identifikátor pole, ve kterém se pohybujeme. Ve výsledku získáme součet prvků podél vedlejší diagonály, která byla zpracována ve směru od spodního levého rohu po pravý horní roh. Diagram úlohy je na obrázku 19.



Obrázek 19: Diagram pro součty na diagonálách

Kód:

```
var
a: array [0..20,0..20] of integer;
i,j,m,n,d1,d2: integer;
begin
read(m);
d1:=0;
d2:=0;
```

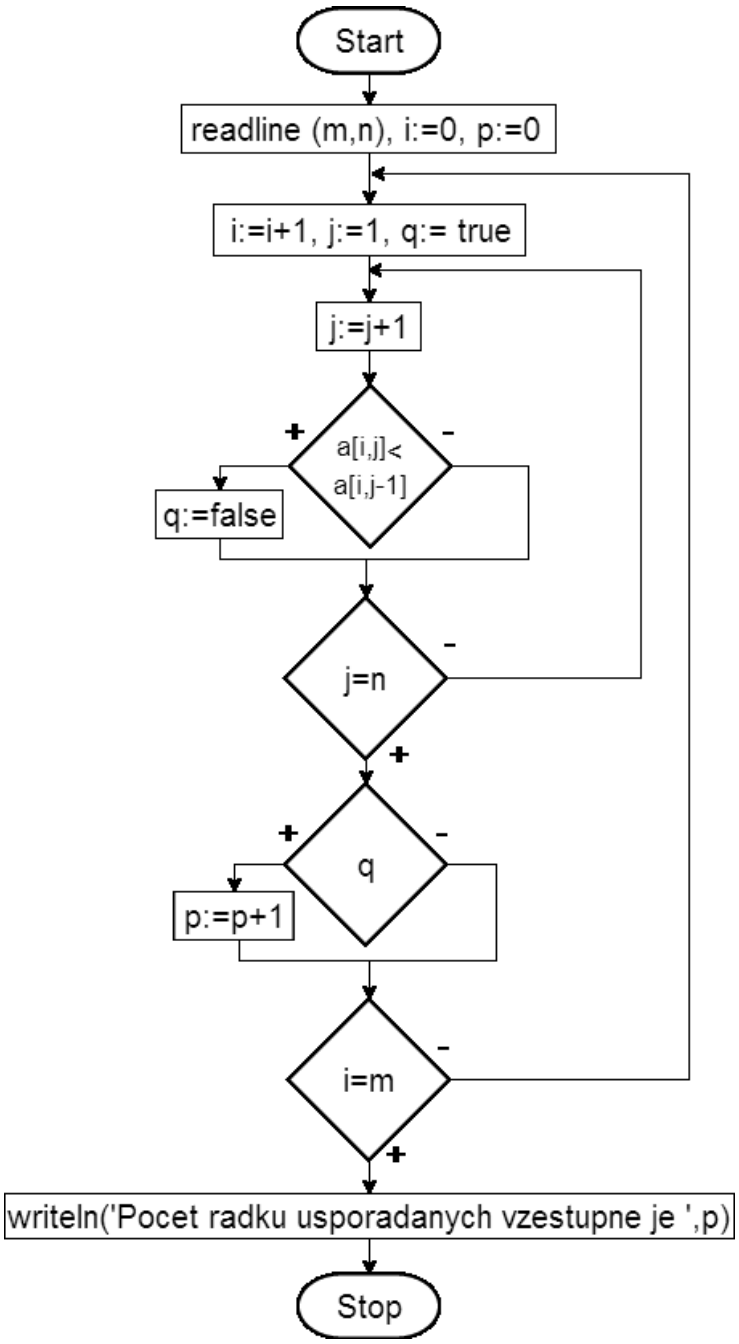
```
for i := 1 to m do
    for j := 1 to m do
        read(a[i][j]);
for i := 1 to m do
    d1:= d1+a[i][i];
for i := 1 to m do
    d2:= d2+(a[i][m+1-i]);
writeln('Soucet prvni diagonaly je ',d1,' a druhe je ',d2,'.');
end.
```

4.4 Příklad 3:

Zadání: Výpis počtu řádků, které jsou uspořádány vzestupně.

Řešením tohoto příkladu bude opět algoritmus, jehož základem budou vnější cyklus pro zpracování řádků a vnitřní pro zpracování jednotlivých prvků v tomto řádku. Program bude porovnávat dvojice sousedících prvků v každém řádku a podle toho zjistí, kolik řádků je uspořádáno vzestupně.

Před zpracováním jednoho řádku se do proměnné „q“, typu boolean, přiřadí hodnota „true“. Následuje cyklus, který porovnává dvojici prvků. Tento cyklus pracuje vždy s prvkem, který je dán počítadlem a prvkem který mu předchází. Proto tento cyklus začíná v daném řádku prvkem prvek na 2. pozici (počítadlo „j“ obsahuje hodnotu 2), který je porovnáván s prvkem na 1. pozici (nemůžeme začít s prvkem na 1. pozici, protože program by ho chtěl porovnat s předchozím prvkem, který neexistuje a následovala by chybová hláška). Pokud je prvek na 1. pozici větší než prvek na 2. pozici (dvojice není uspořádaná), do proměnné „q“, typu boolean, se uloží hodnota „false“. Pokud je ale 2. prvek větší než 1. prvek, (dvojice je uspořádána vzestupně), v proměnné zůstává hodnota „true“. Po zpracování každého řádku program zjišťuje hodnotu této proměnné. Pokud je tato hodnota „true“ (indikuje, že řádek je uspořádán vzestupně), tak je hodnota proměnné „p", která počítá seřazené řádky, navýšena o 1. Pokud je ale v proměnné „q“ hodnota „false“ (která indikuje, že alespoň jedna dvojice v právě zpracovaném řádku není seřazena vzestupně, a tím pádem celý tento řádek nemůže být seřazen vzestupně), tak nenastane žádná „reakce“. Vnější cyklus takto zpracuje všechny řádky pole. Následuje výpis, kde je vypsána hodnota proměnné „p“, která označuje celkový počet řádků, které jsou seřazeny vzestupně.



Obrázek 20: Diagram pro počty řádků uspořádaných vzestupně

Kód:

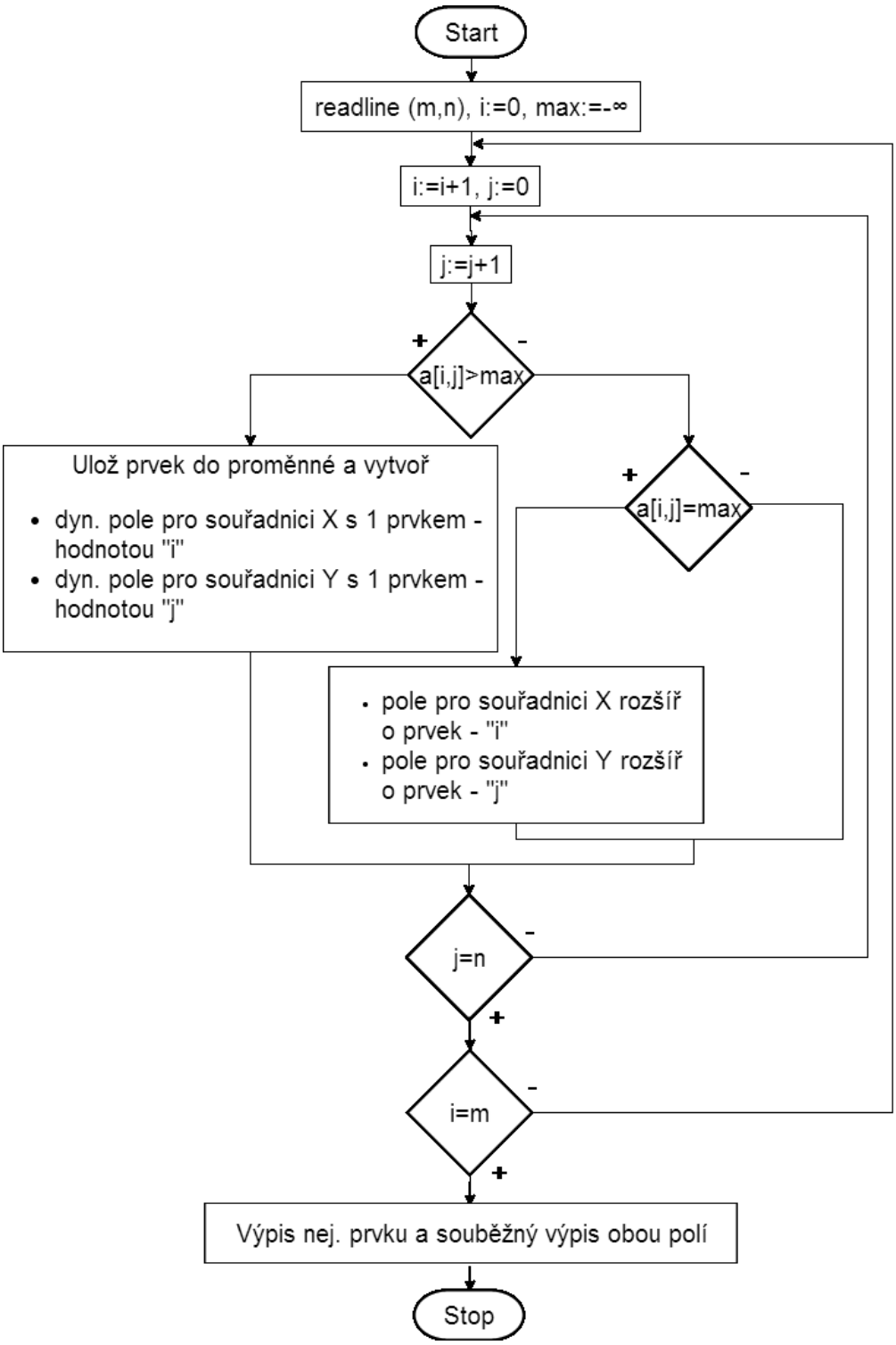
```
var
a: array [0..20,0..20] of integer;
i,j,m,n,p,max: integer;
q: boolean;
begin
read(m);
```

```
read(n);
p:=0;
for i := 1 to m do
    for j := 1 to n do
        read(a[i][j]);
for i := 1 to m do
    begin
        q:=true;
        for j := 2 to n do
            begin
                if a[i][j]<a[i][j-1] then q:=false;
            end;
            if q=true then p:=p+1
    end;
writeln('Pocet radku usporadanych vzestupne je ',p,'.');
end.
```

4.5 Příklad 5:

Zadání: Výpis největšího prvku v poli i s jeho souřadnicemi.

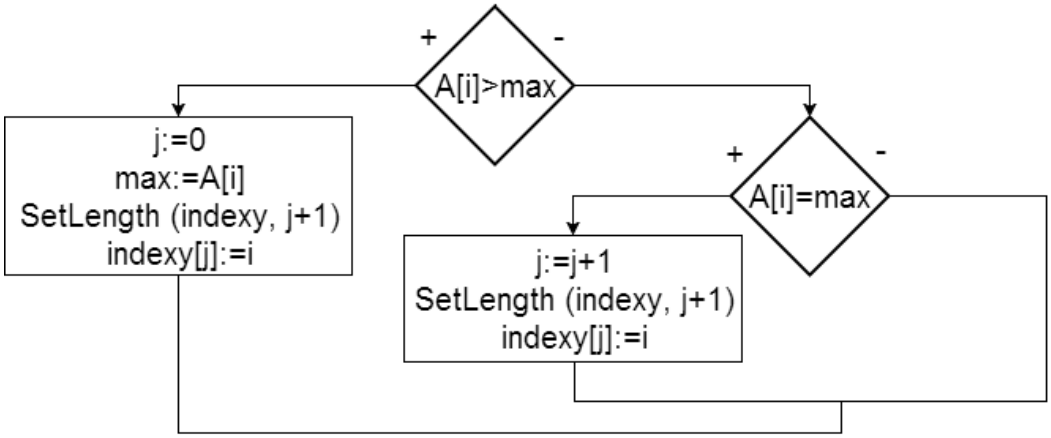
Vzneseme diskuzi nad možným řešením. Pro řešení tohoto zadání použijeme doteď získané poznatky, které je ale musíme upravit do „nového celku“. Připomeneme si jeden z předchozích příkladů, kde program hledal maximum v 1 rozměrném poli. Program si tehdy zaznamenal hodnotu maxima do proměnné typu integer a jeho umístění do dynamického pole. Náš nový program bude pracovat s 2 rozměrným polem a kromě zapamatování maxima si uloží i jeho „souřadnice“, kterými budou řádek a sloupec. Hrubý tvar algoritmu znázorňuje obrázek 21.



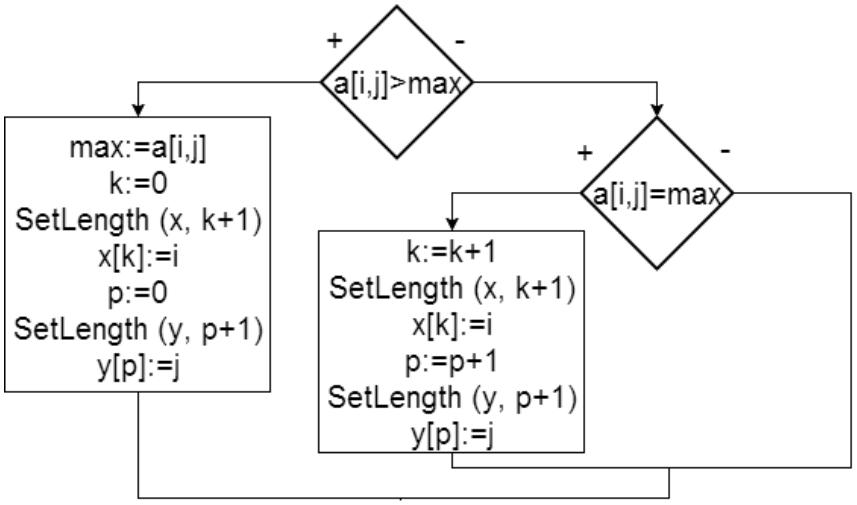
Obrázek 21: Struktura diagramu pro hledání maxima se souřadnicemi

Nový algoritmus bude procházet 2 rozměrným polem opět pomocí 2 cyklů, vnitřní cyklus bude procházet prvek po prvku a vnější cyklus řádek po řádku. Když program narazí na prvek, který je větší než dosavadní maximum, uloží si jeho hodnotu do proměnné typu integer. Pro každou jeho souřadnici (X-ová = počítadlo „i“, Y-ová = počítadlo „j“) bude k dispozici dynamické pole, které bude v tomto případě nejprve vymazáno a následně bude do něj uložena aktuální hodnota počítadla. V případě opakovaného nalezení největšího prvku budou obě pole rozšířena o aktuální hodnotu počítadla. Porovnání klíčové části cyklu jednoho z minulých programů, kde se zaznamenávalo počítadlo „i“ s

nynější obdobnou částí programu, kde je nutné zaznamenat počítadla „i“ a „j“, je znázorněno na obrázcích 22 a 23.

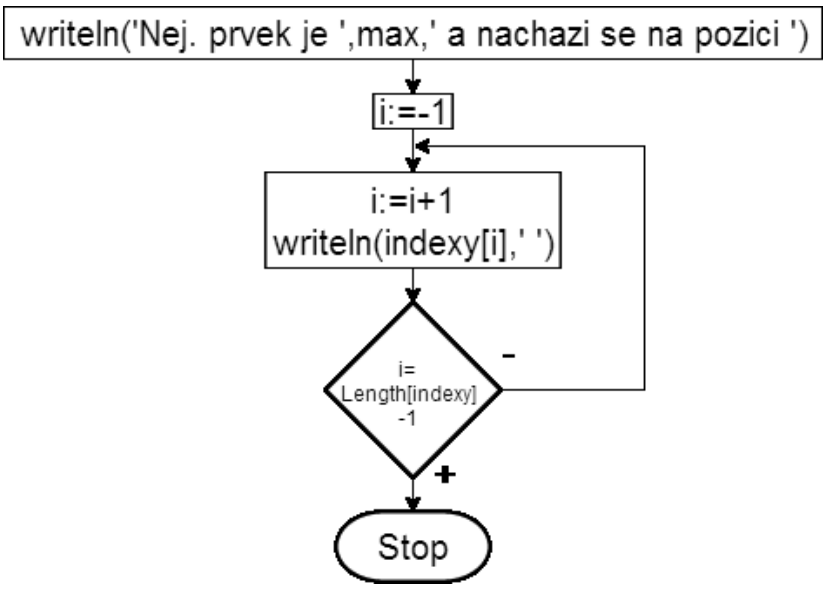


Obrázek 22: Část cyklu ze staršího algoritmu

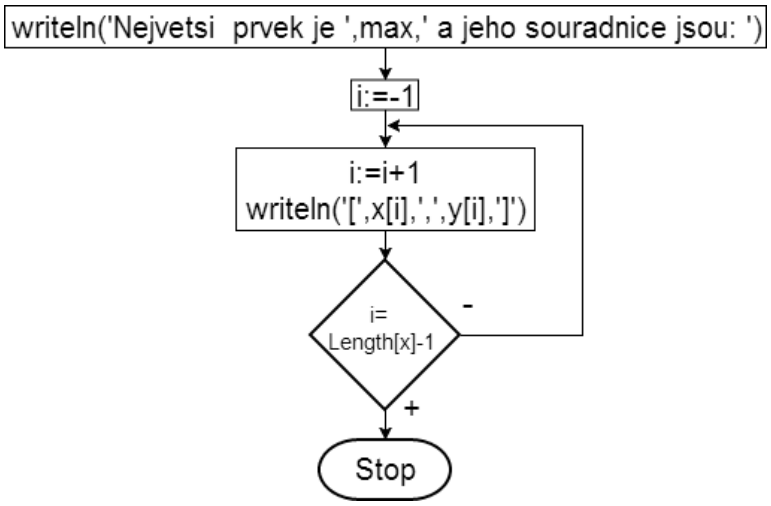


Obrázek 23: Nová část cyklu

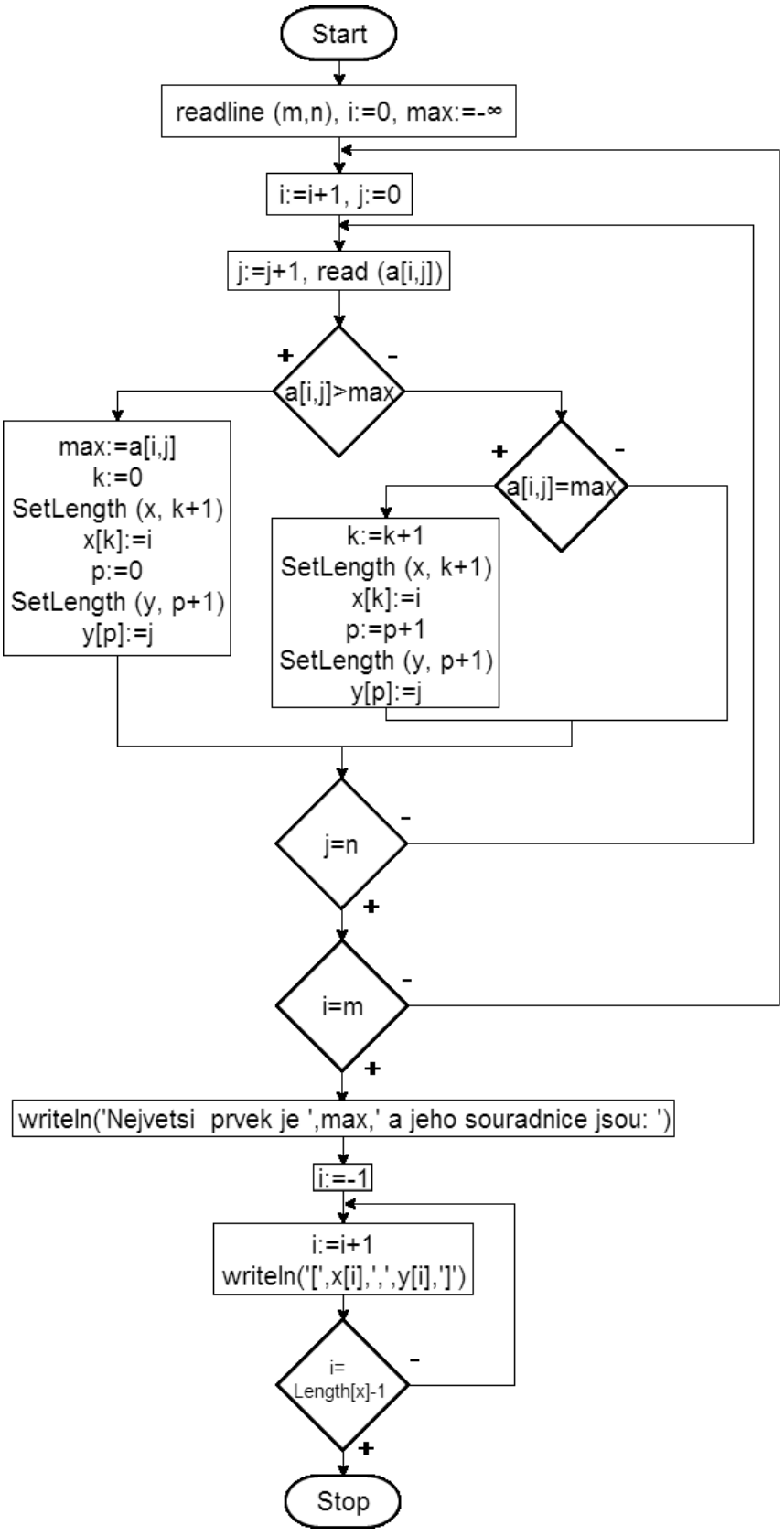
Konečný výpis programu je nutné upravit tak, aby s každou souřadnicí X byla vypsána i korespondující souřadnice Y (obě souřadnice k jednomu zpracovávanému prvku mají stejné umístnění ve svém poli). Porovnání výpisu zmíněného minulého programu, kde se s maximem zaznamenávalo jedno počítadlo s nynějším výpisem, kde se s maximem zaznamenávaly počítadla dvě, je znázorněno na obrázcích 24 a 25. V aktuálně řešeném programu jsme použili u výpisu obou souřadnicových polí stejné počítadlo „i“, protože ke každé souřadnici v 1. poli je korespondující souřadnice v druhém poli na stejném místě.



Obrázek 24: Výpis staršího algoritmu



Obrázek 25: Aktuální výpis algoritmu



Obrázek 26: Diagram pro hledání maxima se souřadnicemi

Kód:

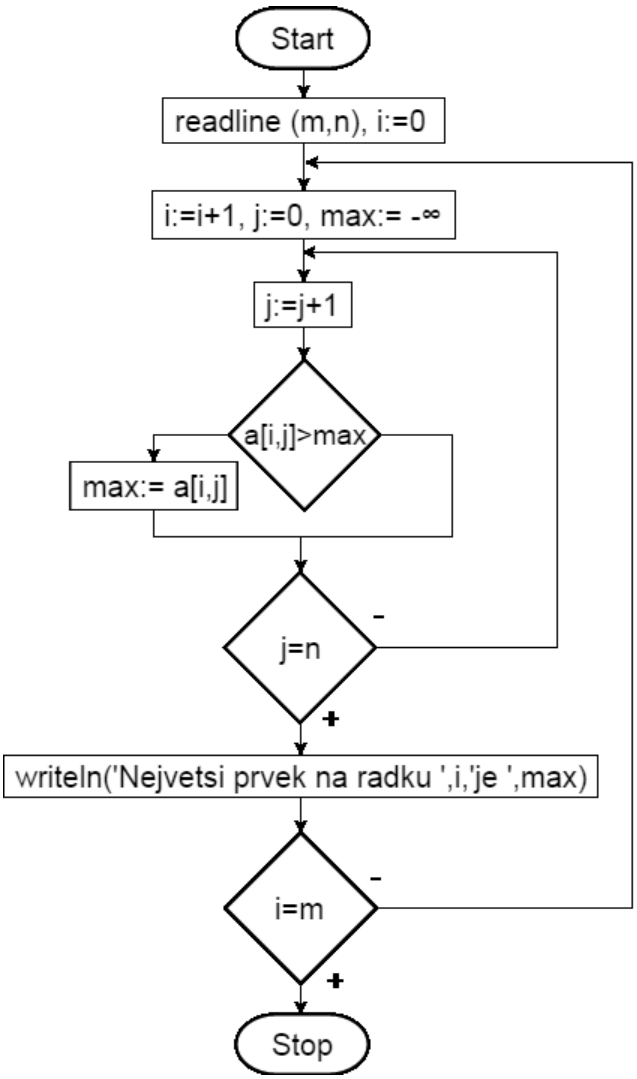
```
var
     a: array [0..10,0..10] of integer;
     x: array of integer;
     y: array of integer;
m,n,i,j,k,p,max: integer;
begin
read(m);
read(n);
max := -MaxInt;
for i := 0 to m-1 do
    for j := 0 to n-1 do
        read(a[i][j]);
for i := 0 to m-1 do
    begin
    for j := 0 to n-1 do
        begin
            if a[i][j]>max then
            begin
                max:=a[i][j];
                k := 0;
                SetLength(x, k+1);
                x[k] := i;
                p := 0;
                SetLength(y, p+1);
                y[p] := j;
            end
            else if a[i][j]=max then
            begin
                k := k + 1;
                SetLength(x, k+1);
                x[k] := i;
                p := p + 1;
                SetLength(y, p+1);
                y[p] := j;
            end;
```

```
        end;
    end;
write('Nejvetsi prvek je ', max, ' a jeho souradnice jsou: ');
    for i := 0 to Length(x)-1  do
         write('[',x[i],',',y[i],']');
end.
```

4.6 Příklad 6:

Zadání: Výpis největších prvků v jednotlivých řádcích.

Opět si pokusíme se žáky nejprve připomenout jeden z předchozích příkladů, jehož cílem bylo tisknout součet prvků na každém řádku. Tento program lze lehce upravit, aby místo počítání součtu hledal na každém řádku maximum. Program bude přecházet jednotlivé prvky v řádku (počítadlo „i“) a když narazí na prvek větší než aktuální maximum, uloží ho do proměnné. Po tom, co program narazí na konec řádku, vypíše toto maximum. Následně se posune na další řádek (počítadlo „j"). Tady je nutné, aby se nejdříve vynulovaly proměnná „i“ a proměnná pro maximum. Opět se procházejí prvky celého řádku, maximum je uloženo a po zpracování řádku vypsáno. Postup se opakuje do doby, než je zpracován poslední řádek.



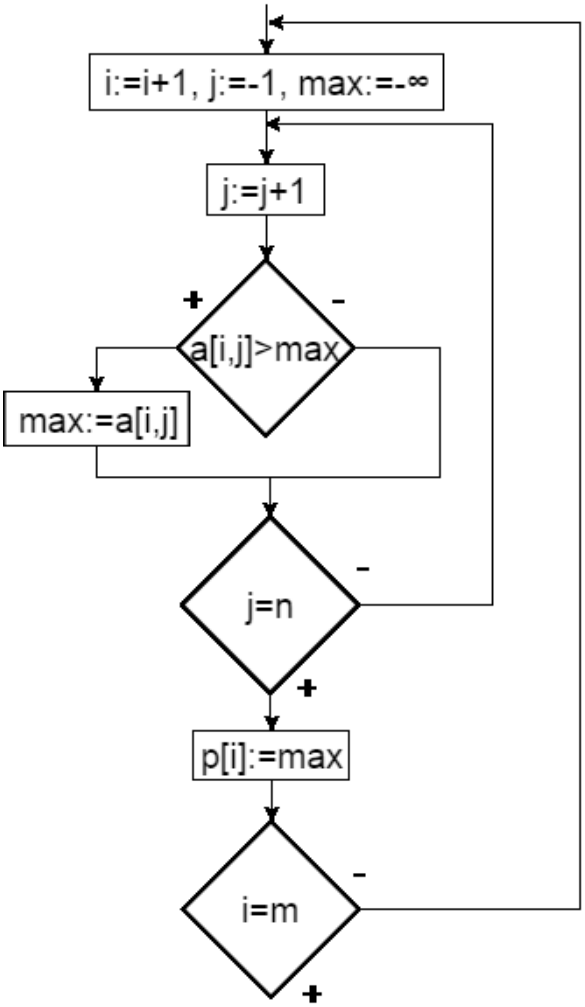
Obrázek 27: Diagram pro hledání maxima

Kód:

```
var
a: array [0..20,0..20] of integer;
i,j,m,n,max: integer;
begin
read(m);
read(n);
for i := 1 to m do
    for j := 1 to n do
        read(a[i][j]);
for i := 1 to m do
```

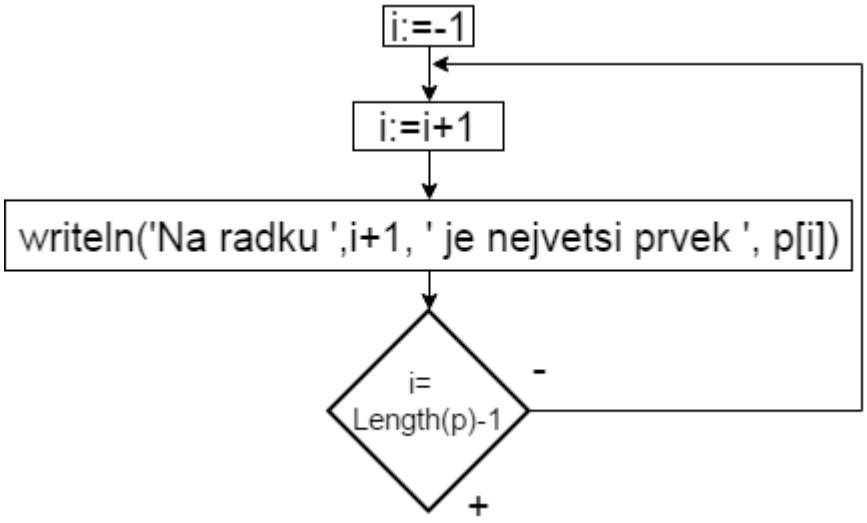
```
    begin
        max := -MaxInt;
        for j := 1 to n do
            begin
                if a[i][j]>max then max:=a[i][j];
            end;
        writeln('Nejvetsi prvek na radku ',i,' je ',max);
    end;
end.
```

Nyní se zeptáme žáků, jak by postupovali, pokud bychom potřebovali z jednotlivých maxim vypočítat třeba průměr. Když se zamyslíme nad fungováním programu, tak zamyslíme, že to není možné, protože proměnná pro maximum obsahuje vždy jen hodnotu posledního maxima (předchozí hodnoty byly během programu přepsány). Při řešení problému zhodnotíme případné nápady žáků a předneseme řešení, kdy se v programu vytvoří dynamické pole, do kterého budou ukládány maxima z jednotlivých řádků. Indexy tohoto pole budou přislouchající k jednotlivým maximům a budou reprezentovat řádky původního 2 rozměrného pole. Nalezení a ukládání maxim do nového pole je znázorněno na obrázku 28.



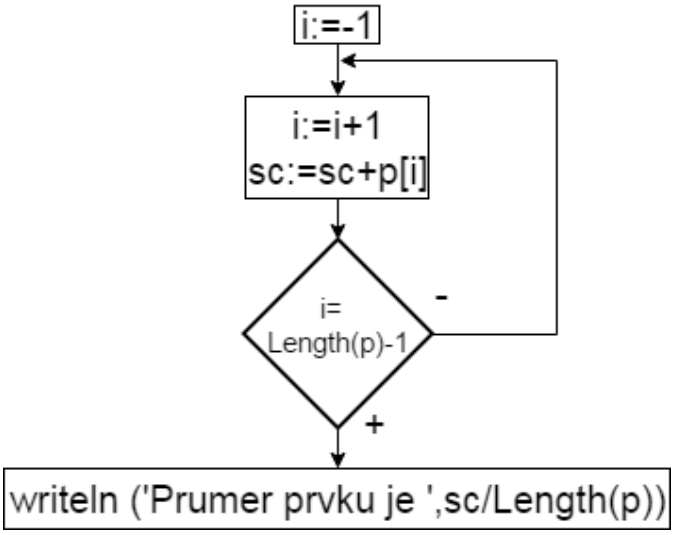
Obrázek 28: Cyklus vytvářející nové pole

Výpis tohoto pole bude mít tvar znázorněný na obrázku 29.



Obrázek 29: Výpis nového pole

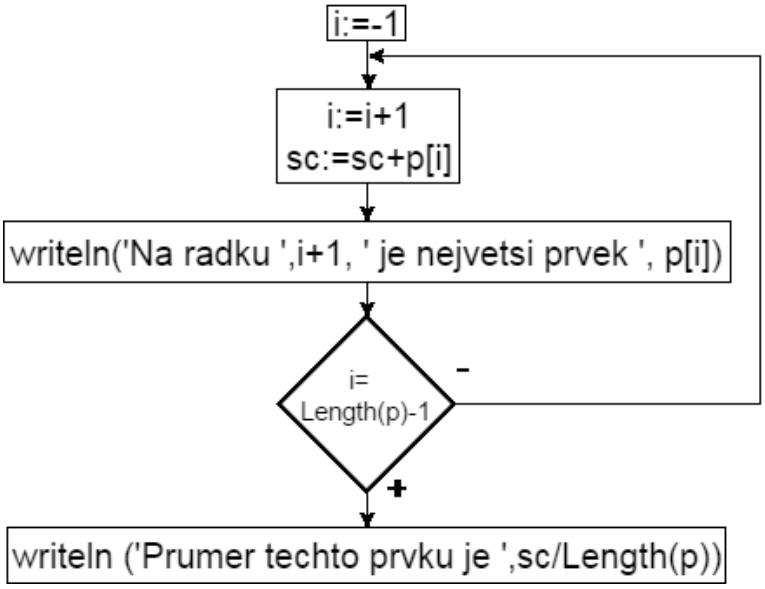
V následujícím kroku vypočítáme průměr ze všech jeho prvků. Postup je zachycen na obrázku 30.



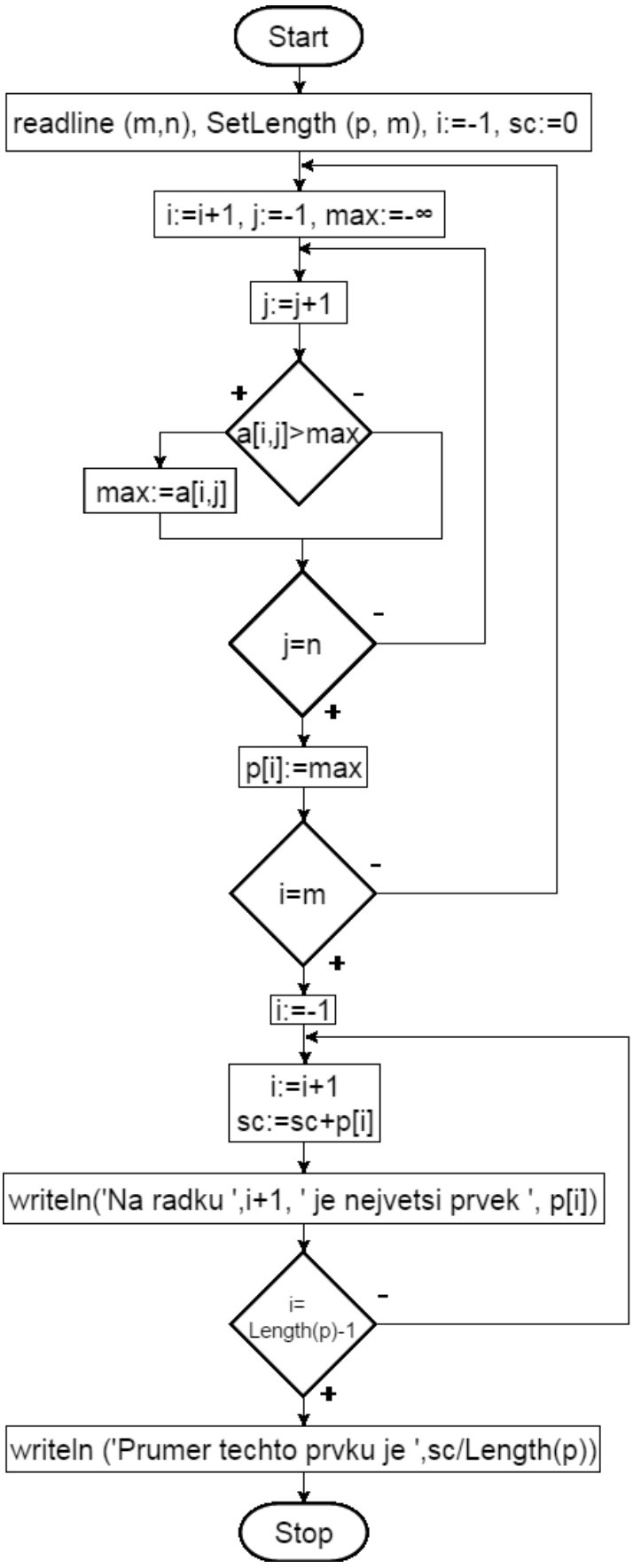
Obrázek 30: Počítání průměru z prvků pole

Toto pole se stalo novým „základem“, které je možno použít pro různé účely, nejenom počítání průměru.

Zmíněný výpis i výpočet průměru mají jednu stejnou charakteristiku – oba jsou provedeny cyklem, který prochází právě vzniklým polem, a proto můžeme obě operace sloučit do jednoho cyklu. Tento cyklus je znázorněn na obrázku 31.



Obrázek 31: Výpis



Obrázek 32: Celý diagram

Kód:

```
var
```

```
    a: array [0..20,0..20] of integer;
    p: array of integer;
i,j,m,n,max: integer;
sc: double;
begin
read(m);
read(n);
sc:=0;
SetLength(p, m);
for i := 0 to m-1 do
    for j := 0 to n-1 do
        read(a[i][j]);
for i := 0 to m-1 do
    begin
        max := -MaxInt;
        for j := 0 to n-1 do
            begin
                if a[i][j]>max then max:=a[i][j];
            end;
        p[i] := max;
    end;
for i := 0 to Length(p)-1  do
    begin
        writeln('V radku ',i+1,' je nejvetsi prvek ',p[i]);
        sc:=sc+p[i];
    end;
writeln('Prumer techto prvku je ',sc/Length(p):3:1);
end.
```

ZÁVĚR

Reformu výuky Informatiky na středních školách je nutné provádět s opatrností a především s dostatečnou připraveností. Doufám, že se v budoucnu dočkáme užitečného výsledku. Svou výuku s použitím mých metodických listů hodnotím kladně. Jejich obsah korespondoval s tematickým plánem na gymnáziu. Metodické listy jsou využitelné pro výuku na středních školách, v jaké míře záleží na konkrétním pedagogovi a tematickém plánu.

SEZNAM POUŽITÉ LITERATURY

[1] Výuka informatiky na školách se mění, zaměří se na programování. *Hospodářské noviny* [online]. [cit. 2018-05-10]. Dostupné z:https://archiv.ihned.cz/c1-65984410-vyuka-informatiky-na-skola

[2] TYPY POLE, ŘETĚZEC A ZÁZNAM. *Pj.jcu* [online]. [cit. 2018-05-10]. Dostupné z: http://home.pf.jcu.cz/~edpo/program/kap08.html

[3] BOTEK, Zdeněk. *Základy informačních technologií*. I. Zlín.

[4] Dynamické premenné. *Pascal Input* [online]. [cit. 2018-05-17]. Dostupné z: http://pascal.input.sk/index.php/22.Prednaska

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA I: POTVRZENÍ O PRAXI

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

VÝKAZ PEDAGOGICKÉ PRAXE

Škola: Gymnázium F. V. Šaška v Skalici
Cvičný učitel: Mgr. Pavol Špetta
Posluchač: Bc. Marek Drábek
Předmět: Informatika
Termín praxe: 9. 4. 2018 - 4. 5. 2018

Souhrnné vyjádření učitele: Učivo vyhovovalo tematickému plánu.
Výklad študenta bol názorný a poučný.
Hodina bola pútavá pre žiakov, splnila stanovené ciele.

GYMNAZIUM Františka Víťazoslava Sasinka Skalica

podpis studenta

razítko školy a podpis cvičného učitele

V Skalici dne: 4.5.2018

Datum	Předmět, učivo	Podpis vyučujícího
9.4.2018	1-rozmerné pole	
13.4.2018	1-rozmerné pole	
16.4.2018	Extrémy v 1-rozmernom poli	
20.4.2018	Extrémy v 1-rozmernom poli	
23.4.2018	Extrémy v 1-rozmernom poli	
27.4.2018	2-rozmerné pole	
30.4.2018	2-rozmerné pole	
4.5.2018	2-rozmerné pole	

PŘÍLOHA II: KÓDY V JAZYKU C++

KAP.2, PR. 1

```
     #INCLUDE <IOSTREAM>
USING NAMESPACE STD;

INT MAIN() {
    INT A[11];
    INT I,M,K;
    BOOL P;
    CIN >> K;
    CIN >> M;
    P = FALSE;
    FOR (I = 1; I <= M; ++I)
        CIN >> A[I];
    I = 0;
    DO
    {
        I = I+1;
        IF (A[I] == K)
            P = TRUE;
    }
    WHILE (!(I == M || P == TRUE) );
    IF (P == TRUE) COUT << "PRVEK " << K << " SE NACHAZI V PO-
SLOUPNOSTI" << ENDL;
              ELSE COUT << "PRVEK " << K << " SE NENACHAZI V PO-
SLOUPNOSTI" << ENDL;
    RETURN 0;
}
```

KAP. 2, PR. 2

```
#INCLUDE <IOSTREAM>
USING NAMESPACE STD;

INT MAIN() {
    INT A[11];
    INT I,J,N;
```

```
    BOOL P;
    CIN >> N;
    P = TRUE;
    FOR (I = 1; I <= N; ++I)
        CIN >> A[I];
    I = 0;
    DO
    {
        I = I+1;
        J = I;
        DO
        {
            J = J+1;
            IF (A[I] == A[J])
                P = FALSE;
        }
        WHILE (!(J == N || P == FALSE) );
    }
    WHILE (!(I == N-1 || P == FALSE) );
    IF (P == FALSE) COUT << "SHODA" << ENDL;
    IF (P == TRUE)  COUT << "NENI SHODA" << ENDL;
    RETURN 0;
}
```

KAP. 2, PR. 3

```
#INCLUDE <IOSTREAM>
USING NAMESPACE STD;

INT MAIN() {
    INT A[11];
    INT I,N,R,S;
    CIN >> N;
    R = 0;
    S = 0;
    FOR (I = 1; I <= N; ++I)
        CIN >> A[I];
```

```
    FOR (I = 1; I <= N; ++I)
    {
        IF (A[I] == 2) R = R+1;
        IF (A[I] == 3) S = S+1;
    }
    IF (R == S) COUT << "DVOJEK JE STEJNE JAK TROJEK" << ENDL;
    IF (R > S)  COUT << "DVOJEK JE VICE NEZ TROJEK" << ENDL;
    IF (S > R)  COUT << "TROJEK JE VICE NEZ DVOJEK" << ENDL;
    RETURN 0;
}
```

KAP. 2, PR. 4

```
#INCLUDE <IOSTREAM>
USING NAMESPACE STD;

INT MAIN() {
    INT A[11];
    INT I,N;
    BOOL P;
    CIN >> N;
    P = FALSE;
    FOR (I = 1; I <= N; ++I)
        CIN >> A[I];
    I = 1;
    DO
    {
        I = I+1;
        IF (A[I-1] > A[I]) P = TRUE;
    }
    WHILE (!(I == N || P == TRUE) );
    IF (P == TRUE) COUT << "POSLOUPNOST NENI USPORADANA VZESTUPNE"
<< ENDL;
              ELSE COUT << "POSLOUPNOST JE USPORADANA VZESTUPNE"
<< ENDL;
    RETURN 0;
}
```

KAP. 2, PR. 5

```
#INCLUDE <IOSTREAM>
USING NAMESPACE STD;

INT MAIN() {
    INT A[11];
    INT B[11];
    INT I,J,M,N;
    CIN >> M;
    CIN >> N;
    COUT << "PRVKY NACHAZEJICI SE V PRUNIKU POLI ";
    FOR (I = 1; I <= N; ++I)
        CIN >> A[I];
    FOR (I = 1; I <= M; ++I)
        CIN >> B[I];
    FOR (J = 1; J <= M; ++J)
    {
        FOR (I = 1; I <= N; ++I)
        {
            IF (A[I] == B[J])
                COUT << B[J] << ENDL;
        }
    }
    RETURN 0;
}
```

KAP. 3, PR. 1

```
#INCLUDE <IOSTREAM>
USING NAMESPACE STD;

INT MAIN() {
    INT N, I, MAX;
    INT A[21];
    COUT << "ZADEJTE POCET CISEL V POSLOUPNOSTI." << ENDL;
```

```
    CIN >> N;
    FOR (I = 1; I <= N; ++I)
    {
        COUT << "ZADEJTE CISLO." << ENDL;
        CIN >> A[I];
    }
    MAX = A[1];
    FOR (I = 2; I <= N; ++I)
        IF (A[I] > MAX)
            MAX = A[I];
    COUT << "NEJVETSI CISLO POSLOUPNOSTI JE " << MAX << ENDL;
    RETURN 0;
}
```

KAP. 3, PR. 2

```
#INCLUDE <CLIMITS>
#INCLUDE <IOSTREAM>
USING NAMESPACE STD;

INT MAIN() {
    INT N, I, MAX, MAX2 ;
    INT A[21];
    MAX = INT_MIN;
    MAX2 = INT_MIN;
    COUT << "ZADEJTE POCET CISEL V POSLOUPNOSTI." << ENDL;
    CIN >> N;
    FOR (I = 1; I <= N; ++I)
    {
        COUT << "ZADEJTE CISLO." << ENDL;
        CIN >> A[I];
    }
    FOR (I = 1; I <= N; ++I)
    {
        IF (A[I] >= MAX)
        {
            MAX2 = MAX;
```

```
            MAX =  A[I];
        }
        ELSE IF (A[I] > MAX2)
            MAX2 = A[I];
    }
    COUT << "2. NEJVETSI CISLO POSLOUPNOSTI JE " << MAX2 << ENDL;
    RETURN 0;
}
```

KAP. 3, PR. 3

```
#INCLUDE <CLIMITS>
#INCLUDE <IOSTREAM>
USING NAMESPACE STD;

INT MAIN() {
    INT N, I, MAX, IND;
    INT A[21];
    MAX = INT_MIN;
    COUT << "ZADEJTE POCET CISEL V POSLOUPNOSTI." << ENDL;
    CIN >> N;
    FOR (I = 1; I <= N; ++I)
    {
        COUT << "ZADEJTE CISLO." << ENDL;
        CIN >> A[I];
    }
    FOR (I = 1; I <= N; ++I)
    {
        IF (A[I] > MAX)
        {
            MAX =  A[I];
            IND = I;
         }
    }
    COUT << "NEJVETSI CISLO POSLOUPNOSTI JE " << MAX << " NACHAZE-
JICI SE NA POZICI " << IND << ENDL;
    RETURN 0;
```

```
}
```

KAP. 3, PR. 4

```
#INCLUDE <CLIMITS>
#INCLUDE <IOSTREAM>
#INCLUDE <VECTOR>
USING NAMESPACE STD;

INT MAIN() {
    INT N, MAX, I, J ;
    INT A[41];
    VECTOR<INT> INDEXY;
    MAX = INT_MIN;
    COUT << "ZADEJTE POCET CISEL V POSLOUPNOSTI." << ENDL;
    CIN >> N;
    FOR (I = 1; I <= N; ++I)
    {
        COUT << "ZADEJTE CISLO." << ENDL;
        CIN >> A[I];
    }
    FOR (I = 1; I <= N; ++I)
    {
        IF (A[I] > MAX)
        {
            J = 0;
            MAX = A[I];
            INDEXY.RESIZE(J+1);
            INDEXY[J] = I;
        }
        ELSE IF (A[I] == MAX)
        {
            J = J + 1;
            INDEXY.RESIZE(J+1);
            INDEXY[J] = I;
        }
    }
```

```
    COUT << "NEJVETSI CISLO POSLOUPNOSTI JE " << MAX << " NACHAZE-
JICI SE NA POZICI(CH) ";
    FOR (I = 0; I <= INT(INDEXY.SIZE()) - 1; ++I)
        COUT << INDEXY[I] << " ";
    RETURN 0;
}
```

KAP. 3, PR. 5

```
#INCLUDE <CLIMITS>
#INCLUDE <IOSTREAM>
#INCLUDE <VECTOR>
USING NAMESPACE STD;

INT MAIN() {
    INT N, MAX, MAX2, I, J, K;
    INT A[41];
    VECTOR<INT> INDEXY, INDEXY2;
    MAX = INT_MIN;
    MAX2 = INT_MIN;
    COUT << "ZADEJTE POCET CISEL V POSLOUPNOSTI." << ENDL;
    CIN >> N;
    FOR (I = 1; I <= N; ++I)
    {
        COUT << "ZADEJTE CISLO." << ENDL;
        CIN >> A[I];
    }
    FOR (I = 1; I <= N; ++I)
    {
        IF (A[I] > MAX)
        {
            MAX2 = MAX;
            MAX = A[I];
            INDEXY2 = INDEXY;
            J = 0;
            INDEXY.RESIZE(J+1);
            INDEXY[J] = I;
```

```
        }
        ELSE IF (A[I] == MAX)
        {
            J = J + 1;
            INDEXY.RESIZE(J+1);
            INDEXY[J] = I;
        }
        ELSE IF (A[I] > MAX2)
        {
            MAX2 = A[I];
            K = 0;
            INDEXY2.RESIZE(K+1);
            INDEXY2[K] = I;
        }
        ELSE IF (A[I] == MAX2)
        {
            K = K + 1;
            INDEXY2.RESIZE(K+1);
            INDEXY2[K] = I;
        }
    }
    COUT << "2. NEJVETSI CISLO POSLOUPNOSTI JE " << MAX2 << " NA-
CHAZEJICI SE NA POZICI(CH) ";
    FOR (I = 0; I <= INT(INDEXY2.SIZE())-1; ++I)
        COUT << INDEXY2[I] << " ";
    RETURN 0;
}
```

KAP. 4, PR. 1

```
#INCLUDE <IOSTREAM>
USING NAMESPACE STD;

INT MAIN() {
    INT A[21][21];
    INT I,J,M,N,SC;
    CIN >> M;
```

```
    CIN >> N;
    FOR (I = 1; I <= M; ++I)
        FOR (J=1; J <= N; ++J)
            CIN >> A[I][J];
    FOR (I = 1; I <= M; ++I)
    {
        SC = 0;
        FOR (J = 1; J <= N; ++J)
        {
            SC = SC+A[I][J];
        }
        COUT << "SOUCET NA RADKU " << I << " JE " << SC << ENDL;
    }
    RETURN 0;
}
```

KAP.4, PR. 2

```
#INCLUDE <IOSTREAM>
USING NAMESPACE STD;

INT MAIN() {
    INT A[21][21];
    INT I,J,M,N,D1,D2;
    CIN >> M;
    D1 = 0;
    D2 = 0;
    FOR (I = 1; I <= M; ++I)
        FOR (J = 1; J <= M; ++J)
            CIN >> A[I][J];
    FOR (I = 1; I <= M; ++I)
        D1 =  D1+A[I][I];
    FOR (I = 1; I <= M; ++I)
        D2 =  D2+(A[I][M+1-I]);
    COUT << "SOUCET PRVNI DIAGONALY JE " << D1 << " A DRUHE JE "
<< D2 << "." << ENDL;
    RETURN 0;
```

```
}
```

KAP. 3, PR. 3

```
#INCLUDE <IOSTREAM>
USING NAMESPACE STD;

INT MAIN() {
    INT A[21][21];
    INT I,J,M,N,P,MAX;
    BOOL Q;
    CIN >> M;
    CIN >> N;
    P = 0;
    FOR (I = 1; I <= M; ++I)
        FOR (J = 1; J <= N; ++J)
            CIN >> A[I][J];
    FOR (I = 1; I <= M; ++I)
    {
        Q = TRUE;
        FOR (J = 2; J <= N; ++J)
        {
            IF (A[I][J] < A[I][J-1])
                Q = FALSE;
        }
        IF (Q == TRUE)
            P = P+1;
    }
    COUT << "POCET RADKU USPORADANYCH VZESTUPNE JE " << P << "."
<< ENDL;
    RETURN 0;
}
```

KAP. 4, PR. 4

```
#INCLUDE <CLIMITS>
#INCLUDE <IOSTREAM>
#INCLUDE <VECTOR>
```

```
USING NAMESPACE STD;

INT MAIN() {
    INT A[11][11];
    VECTOR<INT> X, Y;
    INT M,N,I,J,K,P,MAX;
    CIN >> M >> N;
    MAX = INT_MIN;
    FOR (I = 0; I <= M-1; ++I)
        FOR (J = 0; J <= N-1; ++J)
            CIN >> A[I][J];
    FOR (I = 0; I <= M-1; ++I)
    {
        FOR (J = 0; J <= N-1; ++J)
        {
            IF (A[I][J] > MAX)
            {
                MAX = A[I][J];
                K = 0;
                X.RESIZE(K+1);
                X[K] = I;
                P = 0;
                Y.RESIZE(P+1);
                Y[P] = J;
            }
            ELSE IF (A[I][J] == MAX)
            {
                K = K + 1;
                X.RESIZE(K+1);
                X[K] = I;
                P = P + 1;
                Y.RESIZE(P+1);
                Y[P] = J;
            }
        }
    }
```

```
    COUT << "NEJVETSI PRVEK JE " << MAX << " A JEHO SOURADNICE
JSOU: ";
    FOR (I = 0; I <= INT(X.SIZE())-1; ++I)
        COUT << "["  << X[I] << "," << Y[I] << "]";
    RETURN 0;
};
```

KAP. 4, PR. 5

```
#INCLUDE <CLIMITS>
#INCLUDE <IOSTREAM>
#INCLUDE <VECTOR>
USING NAMESPACE STD;

INT MAIN() {
    INT A[21][21];
    VECTOR<INT> P;
    INT I,J,M,N,MAX;
    DOUBLE SC;
    CIN >> M;
    CIN >> N;
    SC = 0;
    P.RESIZE(M);
    FOR (I = 0; I <= M-1; ++I)
        FOR (J = 0; J <= N-1; ++J)
            CIN >> A[I][J];
    FOR (I = 0; I <= M-1; ++I)
    {
        MAX = INT_MIN;
        FOR (J = 0; J <= N-1; ++J)
        {
            IF (A[I][J] > MAX)
                MAX = A[I][J];
        }
        P[I] = MAX;
    }
    FOR (I = 0; I <= INT(P.SIZE()) - 1; ++I)
```

```
    {
        COUT << "V RADKU " << I+1 << " JE NEJVETSI PRVEK " << P[I]
<< ENDL;
        SC = SC+P[I];
    }
    COUT << "PRUMER TECHTO PRVKU JE " << (DOUBLE)SC / P.SIZE() <<
ENDL;
}
```