Moderné vzdelávanie pre vedomostnú spoločnosť / Projekt je spolufinancovaný zo zdrojov EÚ

Ing. Michal Kompan

Analýza a návrh algoritmov z matematiky

Osvedčená pedagogická skúsenosť edukačnej praxe

Banská Bystrica

2013

Vydavateľ:	Metodicko-pedagogické centrum, Ševčenkova 11, 850 01 Bratislava
Autor OPS/OSO:	Ing. Michal Kompan
Kontakt na autora:	Gymnázium Antona Bernoláka Námestovo, Mieru 307/23, 029 01 Námestovo michal.kompan@stonline.sk
Názov OPS/OSO:	Analýza a tvorba algoritmov
Rok vytvorenia OPS/OSO:	2013
Odborné stanovisko vypracoval:	PaedDr. Katarína Poláčiková

Za obsah a pôvodnosť rukopisu zodpovedá autor. Text neprešiel jazykovou úpravou.

Táto osvedčená pedagogická skúsenosť edukačnej praxe/osvedčená skúsenosť odbornej praxe bola vytvorená z prostriedkov národného projektu Profesijný a kariérový rast pedagogických zamestnancov.

Projekt je financovaný zo zdrojov Európskej únie.

Kľúčové slová

Algoritmus, vývojový diagram, program, algoritmické myslenie, programovací jazyk.

Anotácia

Cieľom práce bolo poskytnúť učiteľom študijný materiál – banku úloh s návodom na také riešenia, ktoré budú formovať algoritmické myslenie žiakov. Slúži ako pomôcka pri analýze a návrhu algoritmov a pri tvorbe programov v ľubovoľnom programovacom jazyku. Osvedčená pedagogická skúsenosť je teda určená hlavne učiteľom informatiky na stredných školách, kde žiaci maturujú z predmetu informatika. Súčasne platná legislatíva určuje, aby v každom maturitnom zadaní z informatiky bola zaradená jedna úloha z algoritmov a programovania. Je teda kladený veľký dôraz práve na tento tematický celok v rámci osnov predmetu informatika, ktorý je špecifický tým, že na jeho osvojenie je potrebné aj špecifické – algoritmické myslenie.

Do práce boli vyberané úlohy s ktorými sa žiaci stretávajú aj na iných vyučovacích predmetoch, najmä matematike.

Práca pozostáva z dvoch častí. Prvá kapitola obsahuje teoretický základ k tvorbe algoritmov. Popisuje princípy, postupy a požiadavky na návrh správnych algoritmov. Druhá časť práce obsahuje zadania konkrétnych úloh s ich riešením. Riešenie každej úlohy obsahuje jej podrobnú analýzu, bez ktorej nie je možné navrhnúť správny algoritmus. Ďalej obsahuje popis vstupných údajov a požiadavky na ne a zápis algoritmu vo veľmi zrozumiteľnej a univerzálnej forme, ktorou je vývojový diagram.

OBSAH

ÚVOD

Dlhoročné pedagogické skúsenosti z vyučovaním predmetu informatika na strednej škole nás priviedli k presvedčeniu, že pre žiakov je najviac problematický (najťažší) tematický celok „Postupy, riešenie problémov, algoritmické myslenie“. S týmto tematickým celkom sa stretávajú v najväčšom rozsahu žiaci, ktorí maturujú z informatiky. Medzi prvé a veľmi dôležité kroky v procese tvorby programov pre PC patrí práve rozbor problému a návrh jeho riešenia pomocou jednoznačného postupu krokov – algoritmu. Zvládnuť programovací jazyk (pascal, C resp. iný) po syntaktickej stránke nie je náročné. Najťažšie pri tvorbe programov je návrh správneho algoritmu. Aby to žiaci dokázali, musia mať analytické myslenie. Musia dokázať previesť dostatočný rozbor riešeného problému a následne jeho riešenie sformulovať do jednoznačnej postupnosti krokov – algoritmu. Sú žiaci, ktorí majú takéto schopnosti ako talent. Algoritmické myslenie sa samozrejme dá formovať – rozvíjať a je veľmi dôležité pre akúkoľvek činnosť žiaka, nielen priamo pre jeho štúdium informatiky. Je vhodné ak žiaci riešia úlohy s ktorými sa stretávajú na iných vyučovacích predmetoch, pričom to nemusia byť iba prírodovedné predmety. Príprava učiteľa na hodinu informatiky na ktorej žiaci tvoria programy je časovo náročná. V prípade, ak má učiteľ k dispozícii banku úloh so zadaním úlohy, jej rozborom a vytvoreným algoritmom, je pomerne jednoduché tento algoritmus prepísať do konkrétneho programovacieho jazyka podľa toho, ktorý sa na škole vyučuje. Pritom úlohy vychádzajú z tematických okruhov s ktorými sa žiaci už stretli na iných vyučovacích predmetoch, najmä matematike. Učitelia tak majú v rukách nástroj pomocou ktorého by žiaci mali lepšie chápať princípy, postupy riešenia konkrétnych úloh.

Najväčším prínosom takto ponímaného prístupu v súlade s požiadavkami ŠVP je to, že žiaci získajú algoritmické myslenie a schopnosť vedieť uvažovať nad riešením problémov pomocou IKT. Naučia sa pri riešení problémov hľadať najefektívnejšie postupy z rôznych oblastí. OPS je určená jednak učiteľovi ale aj pre žiaka, ktorému slúži ako návod pre riešenie resp. tvorbu programu pre PC konkrétneho problému. Vybraté riešené úlohy majú slúžiť ako návod a motivovať učiteľov k určitému efektívnemu postupu pri formovaní algoritmického myslenia žiakov. Zároveň by to mal byť štandardný postup pri tvorbe programov. Niekedy, pri riešení jednoduchých úloh skúsenými programátormi, sa zdá, že tento krok v procese tvorby programov chýba. Skúsený programátor však pri riešení jednoduchších úloh prepisuje algoritmus úlohy priamo do programovacieho jazyka, pretože už má dostatočne osvojené algoritmické myslenie.

Rozvoj tvorivého myslenia, ktoré je potrebné pri návrhu, tvorbe postupov na riešenie konkrétnych úloh je veľmi dôležité pre všetkých žiakov, nielen pre tých ktorí budú tvoriť programy pre PC. Je všeobecne známy fakt, že naši žiaci majú dobré encyklopedické vedomosti, často majú zvládnuté postupy riešenia úloh, ale nedokážu pri riešení dostatočne tvorivo myslieť. Prístup k riešeniu úloh cez podrobný rozbor problému a tvorbu elementárnych krokov na jeho riešenie umožní v žiakoch rozvíjať práve tieto schopnosti. Vzhľadom k rozsahu využívania IKT v živote dnešného človeka, je priam nevyhnutné aby žiaci získavali zručnosti, ktoré im umožnia navrhovať riešenia akýchkoľvek úloh s možnosťou využitia IKT. Mali by to byť zároveň aj najefektívnejšie spôsoby riešenia.

1 ALGORITMUS VO VYUČOVACOM PROCESE

V rámci povinného predmetu informatika si na gymnáziu všetci žiaci majú osvojiť základy tvorby algoritmov. Vzhľadom k tomu, že analytické, tvorivé myslenie je potrebné pre akúkoľvek činnosť človeka, je tento tematický celok prierezový. Základy algoritmického myslenia, získané na hodinách informatiky by žiaci mali uplatňovať na všetkých ostatných vyučovacích predmetoch.
Naučiť žiakov tvoriť programy v nejakom programovacom jazyku, bez predchádzajúceho zvládnutia základov tvorby algoritmov nie je možné. Rozsah i obsah učiva z informatiky je vzhľadom k rýchlemu rozvoju IKT však taký, že učiteľ musí filtrovať obsah učiva pre konkrétnu cieľovú skupinu žiakov. Algoritmom a programovaniu v konkrétnom programovacom jazyku sa tak často v dostatočnej miere venujú iba žiaci, ktorí maturujú z informatiky. V snahe vytvoriť vhodné podmienky pre formovanie algoritmického myslenia, nielen u žiakov, ktorí maturujú z informatiky, sme vytvorili a ďalej dopĺňame zbierku úloh s návodom na riešenie konkrétnych problémov formou tvorby ich algoritmov. Naše riešené úlohy slúžia ako návod učiteľom pri osvojovaní si určitých postupov žiakmi a sú použiteľné nielen na hodinách informatiky. Pri návrhu riešení úloh počítačovým programom je potrebný špecifický prístup. Niektoré kroky v postupe riešenia pri tzv. „ručnom“ riešení sú tak samozrejmé, že ich môžeme vynechať. Najčastejšie sú to kroky súvisiace s definovaním vstupných údajov pre ich spracovanie v programe. Tu musíme predpokladať že pri nesprávnom zadaní vstupných údajov túto skutočnosť musí zistiť algoritmus – program, inak by taký program havaroval. Pri návrhu algoritmov, ktoré sa budú realizovať pomocou IKT musí byť algoritmus spracovaný do omnoho podrobnejších krokov.

Z uvedeného je zrejmý aj jeden z dôležitých cieľov vyučovania informatiky[1]:

- žiaci získajú algoritmické myslenie a schopnosť uvažovať nad riešením problémov pomocou IKT.
- žiaci vedia uvažovať nad rôznymi parametrami efektívnosti rôznych riešení problémov.
- žiaci vedia uplatňovať rôzne postupy a mechanizmy pri riešení úloh z rôznych oblastí.

1.1 Kontext a rámec

- **Typ školy:** stredná – gymnázium – vyššie sekundárne vzdelávanie
- **Východiská:** pre zvládnutie tejto OPS sa od učiteľov očakáva dobré analytické myslenie, určitý nadhľad pri riešení úloh s možnosťou ich riešenia na PC a poznať základný všeobecný princíp činnosti PC.
- **Vzdelávacia oblasť:** matematika a práca s informáciami.
- **Škola, ročník:** stredná – gymnázium, prvý – štvrtý.
- **Predmet:** informatika.
- **Tematický celok:** postupy, riešenie problémov, algoritmické myslenie.

1.2 Hlavné ciele

V súlade s cieľmi vyučovania informatiky na gymnáziu je hlavným cieľom tejto OPS:

- zefektívniť prácu učiteľa tým, že má v dispozícii študijný materiál - banku úloh s návodom na tvorbu algoritmov
- mať učebnú pomôcku – návod ako formovať a rozvíjať algoritmické myslenie žiakov
- motivovať aj učiteľov iných predmetov ako je informatika, aby pri návrhoch riešení rôznych úloh používali postupy formujúce algoritmické myslenie žiakov, postupy, ktoré predpokladajú riešenie úloh s použitím IKT.

1.3 Algoritmus

Návrh algoritmu prestavuje vlastne jednu z etáp tvorby programov pre PC. Pri riešení jednoduchých úloh často niektoré kroky chýbajú. Je však veľmi dôležité, aby žiaci, ktorí začínajú tvoriť programy získali správne návyky, uvedomovali si všetky kroky tvorby programu.

Etapy tvorby programov:

- *Zadanie úlohy, a rozbor problému* (čo ideme riešiť, presne sformulujeme zadanie problému a samozrejme aj požiadavky kladené na program). Výsledkom rozboru je popis vstupných a výstupných informácií a vzťahov medzi nimi. Pre správny návrh algoritmu je nevyhnutné, aby sme dokonale poznali problematiku – teóriu riešenej úlohy.
- *Návrh krokov pre spracovanie definovaných údajov /návrh algoritmu/*: postupnosť krokov musí mať určité vlastnosti, z ktorých žiadna nesmie chýbať. Výsledkom návrhu riešenia je zápis jednotlivých krokov (príkazov) algoritmu a to buď v algoritmickom jazyku alebo graficky pomocou vývojového diagramu alebo štruktúrogramu. Ak sa vyskytnú viaceré možnosti riešenia, snažíme sa zistiť, ktorý je najefektívnejší.
- *Realizácia:* prepis algoritmu do programovacieho jazyka, ladenie a testovanie – ladenie programu (oprava chýb syntaktických a logických, novšie verzie, prispôsobenie softvéru požiadavkám používateľa)
- *Dokumentácia a údržba*: Vhodná je aj dokumentácia, ktorú môžu využiť iní programátori, kde je popis, riešenia jednotlivých problémov.

K tomu aby sme boli schopní správne navrhnúť algoritmus, je potrebné dokonale rozumieť samotnému pojmu algoritmus. V odbornej literatúre nájdeme mnoho v podstate veľmi podobných definícií tohto pojmu. Pre našu potrebu by sme mohli algoritmus chápať ako postupnosť krokov pomocou ktorých sa spracovávajú – transformujú vstupné údaje na požadovaný výstup. Z uvedenej charakteristiky vyplýva, že návrh algoritmu predstavuje vlastne dve prvé etapy tvorby programov.

Návrh riešenia určitého problému vytvorením krokov algoritmu však nemusí vždy súvisieť iba s riešením úlohy na PC, teda prepísaním algoritmu do programovacieho jazyka. Vedieť formulovať úlohu, previesť jej analýzu a návrh postupu riešenia je jedným z vyučovacích cieľov pri tematickom celku postupy, riešenie problémov, algoritmické myslenie v rámci predmetu informatika. Žiaci väčšinou poznajú princípy,

riešenia určitých úloh ale nevedia sformulovať postupnosť krokov, návod podľa ktorého by niekto iný (napr. aj PC na základe obsiahnutého algoritmu v programe) dokázal úlohu riešiť. Podľa správne navrhnutého algoritmu by mal vyriešiť úlohu aj ten, kto nerozumie princípu, podstate riešeného problému.

Vlastnosti algoritmov

- *jednoznačnosť /determinovanosť/:*v každom kroku algoritmu musí byť jednoznačne dané, ktorý nasledujúci krok sa bude realizovať. K nejednoznačnosti algoritmu dochádza najčastejšie pri vetvení algoritmu.
- *konečnosť /rezultatívnosť/:*po konečnom počte krokov musí algoritmus skončiť. Viacnásobné opakovanie určitých krokov algoritmu nazývame cyklus. Pri nesprávne formulovanej výstupnej podmienke dochádza k takémuto zacykleniu, príkazy sa opakujú v nekonečnom cykle.
- *Hromadnosť:* algoritmus je možné použiť na riešenie problémov toho istého typu.
- *Efektívnosť:* správny algoritmus má byť aj efektívny z hľadiska času /časová efektívnosť/ a z hľadiska nárokov na pamäť pre uchovávanie údajov /pamäťová efektívnosť/.

Zápis algoritmu

Zápis algoritmu môžeme previesť viacerými spôsobmi. Záleží od toho, pre aký účel je algoritmus navrhnutý.

- *slovný zápis*
- *zápis pomocou vývojového diagramu, štruktúrogramu*
- *rozhodovacie tabuľky*
- *programovacie jazyky*

Pre účely ďalšieho prepisovania algoritmu do programovacieho jazyka je vhodný zápis vo forme vývojového diagramu.

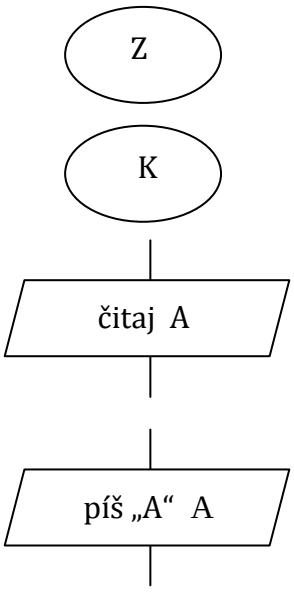
Značky vývojového diagramu použité v práci:

Začiatok algoritmu

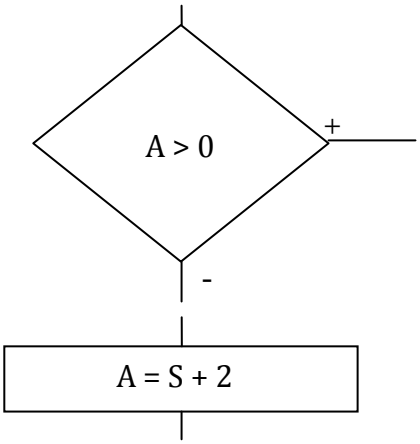
Koniec Algoritmu

Blok pre načítanie vstupných údajov

Blok pre výstup – výpis údajov



Rozdovací blok – vetvenie algoritmu



Blok priradenia

V práci vo vývojových diagramoch v riešených úlohách používame relačné, logické, aritmetické operátory a príkazy priradenia ako v programovacom jazyku C.

1.4 Navrhované postupy a odporúčania

- Po zadaní úlohy učiteľ v diskusii so žiakmi zistí ich úroveň osvojenia si vedomostí z riešeného problému. Ak napr. navrhujú algoritmus pre výpočet koreňov kvadratickej rovnice, musia poznať potrebné vzorce (z matematiky) na ich výpočet. Je to jeden z nevyhnutných predpokladov úspešného návrhu algoritmu. Učiteľ by mal zaraďovať úlohy v súlade z požiadavkou rozvíjania medzipredmetových vzťahov.
- Žiaci rozdelení do skupín analyzujú požiadavky na výstup z algoritmu. Definujú teda cieľový – požadovaný stav. Tento často vyplýva už zo zadania úlohy.
- Žiaci rozdelení do skupín analyzujú požiadavky na vstupné hodnoty, ktoré bude algoritmus spracovávať. Učiteľ kladie dôraz na to aby žiaci uvažovali nad problémom komplexne, s možnosťou riešenia úlohy prostriedkami IKT. Práve táto skutočnosť si vyžaduje špecifický prístup k analýze úlohy.
- Žiaci prezentujú svoje návrhy, diskutujú o nich. Učiteľ usmerňuje diskusiu v tom smere, aby návrhy žiakov zohľadňovali všetky možné požiadavky na správne vstupné a výstupné spracovávané údaje. Tu je potrebné, aby učiteľ upozorňoval na možné chyby pri zadávaní údajov z dôvodu nepozornosti, omylu užívateľa. Zaradením vhodných vstupných testov do algoritmu je možné výrazne znížiť pravdepodobnosť chybných vstupných údajov.
- Učiteľ nakoniec (môže aj pomocou interaktívnej tabule) odhalí správne vstupné a výstupné požiadavky.
- Žiaci rozdelení do skupín navrhnú postupnosť krokov - algoritmus riešenia úlohy. Tento spoločne prezentujú a diskutujú o jednotlivých krokoch a ich správnosti. Je dôležité aby učiteľ analyzoval v diskusii rôzne návrhy. Viaceré navrhované postupy žiakov môžu byť správne, ale nemusia byť rovnako efektívne.
- Jednotlivé kroky zapisujú do vývojového diagramu.
- Učiteľ odhalí správny algoritmus.

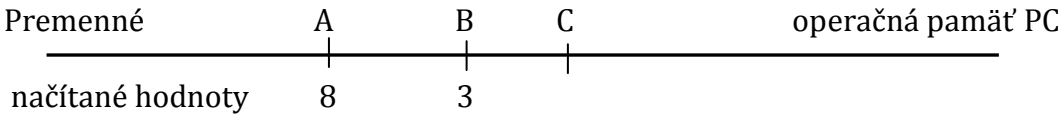
Časová dotácia pre jednotlivé kroky je veľmi individuálna a závisí od zložitosti riešenej úlohy. Učiteľ musí pri jednotlivých krokoch upozorňovať najmä na tie, ktoré sú nevyhnutné v prípade ak očakávame, že úloha – algoritmus sa bude realizovať na PC. Žiaci si musia stále uvedomovať, že v navrhovanom algoritme musia byť aj kroky, ktoré sú pre riešiteľa – človeka často úplne samozrejmé a v postupe by ich inak neuvádzali.

2 NÁVRHY ALGORITMOV

Úloha 1

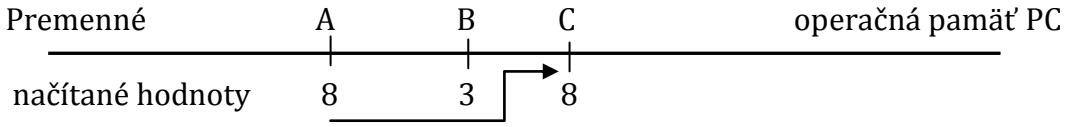
Návrh algoritmu pre výmenu hodnôt dvoch premenných A,B s použitím pomocnej, tretej premennej.

Rozbor problému:
Ak načítame hodnoty dvoch premenných v pamäti PC sa na príslušnú adresu tieto hodnoty zapíšu. Príkaz priradenia napr. v jazyku C *A=B*, spôsobí, že obsah premennej *A* sa prepíše obsahom premennej *B*. Ak by sme následne zapísali príkaz *B=A* v oboch premenných by sme mali rovnakú hodnotu *B*. Preto najjednoduhší spôsob zámeny je použitie pomocnej premennej napr. *C.* Celý priebeh výmeny môžeme znázorniť na schéme operačnej pamäti s obsahom jednotlivých premenných v jednotlivých krokoch algoritmu. Obrázok č. 1 znázorňuje situáciu po načítaní vstupných hodnôt do premenných *A, B.*



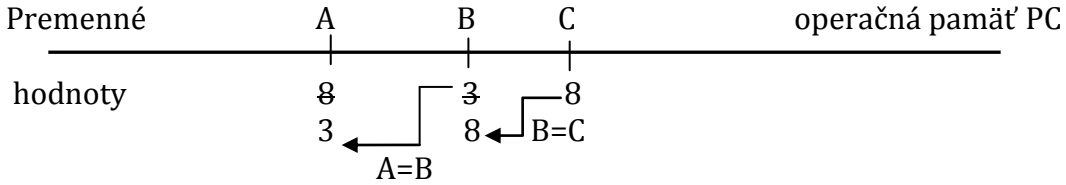
Obr. 1: Obsah premenných po načitaní vstupných hodnôt (vlastný zdroj).

Ak chceme do premennej *A* priradiť hodnotu premennej *B* musíme si najskôr obsah premennej *A* „odložiť“ , zapamätať v inej premennej napr. pomocnej premennej *C*. Teda použijeme príkaz *C=A*.



Obr. 2: Obsah premenných po príkaze *C=A* (vlastný zdroj).

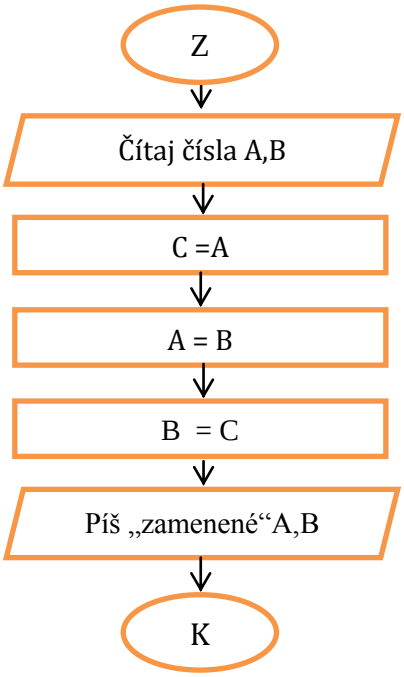
V nasledujúcom kroku už môžeme prepísať obsah premennej *A* premennou *B* a do premennj *B* uložiť obsah pomocnej premennej *C*. Výsledkom týchto krokov bude zamenený obsah premenných *A,B* ako ukazuje obr. 3.



Obr. 3: Obsah premenných po výmene (vlastný zdroj).

Vstup: *A, B* – čísla.
Výstup: *A,B* – čísla so vzájomne zamenenými hodnotami

Vývojový diagram



Úloha 2

Návrh algoritmu pre výmenu hodnôt dvoch premenných A,B bez použitia pomocnej tretej premennej.

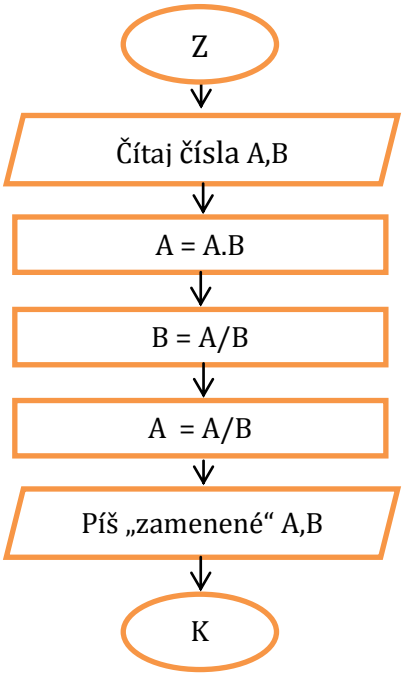
Rozbor problému:
Ak vstupné hodnoty premenných *A,B* budú čísla môžeme použiť na výmenu nejaký prepočet pomocou matematických operácií, napr. súčinu a podielu. Ak premenné vynásobíme a výsledok uložíme do premennej A. Potom stačí urobiť podiel ich súčinu – teda premennej *A* a premennej *B* a tento uložiť do premennej *B*. Tak do premennej *B* dostaneme pôvodnú hodnotu premenej*A*. Následne urobíme podiel *A* s *B* a výsledok uložíme do *A*, čo je vlastne pôvodná hodnota premennej *B*. Názorne sú jednotlivé kroky znázornené na obr. 4.

premenné		A	B
obsah v pamäti		**8**	**2**
	po príkaze *A=A*B*	16	2
	po príkaze B=A/B	16	8
	po príkaze A=A/B	**2**	**8**

Obr. 4: Zámena obsahu dvoch premenných bez pomocnej premennej (vlastný zdroj).

Vstup: *A, B* – čísla.
Výstup: *A,B* – čísla so vzájomne zamenenými hodnotami

Vývojový diagram



Úloha 3

Návrh algoritmu pre výpočet hodnoty funkcie $f(x)=A_n.X^n+A_{n-1}.X^{n-1}+.....+A_1.X^1+A_0$ v bode X.

Rozbor problému:

Vypočítať hodnotu danej funkcie znamená vypočítať súčet súčinov koeficientov A_i a mocniny X^i pre $I=0$ až N . Na výpočet mocniny môžeme vytvoriť funkciu mocnina. Úlohu je možné riešiť deklaráciou jednorozmerného poľa *A,* do ktorého načítame hodnoty, alebo po zadaní príslušného koeficienta hneď počítame súčin mocniny zadaného čísla *X* a príslušného koeficienta $A_{i..}$ Na začiatku algoritmu si definujeme pomocné premenné *I* a *F* s počiatočnou hodnotou 0. Premenná *I* (riadiaca premenná cyklu) zvyšuje hodnotu o 1 po každom prečítaní hodnoty koeficienta *A.* Do premennej *F* budeme kumulovať súčet súčinov koeficientov A_ia mocniny X^i pre $I=0$ až N . Definovaná funkcia *mocnina* vypočítava hodnotu *I*-tej mocniny čísla *X* tak, že v cykle toto číslo násobí *I*-krát. V schéme vývojového diagramu je znázornená aj náhrada formálnych parametrov skutočnými pri volaní funkcie.

Vstup:

N – nezáporné celé číslo

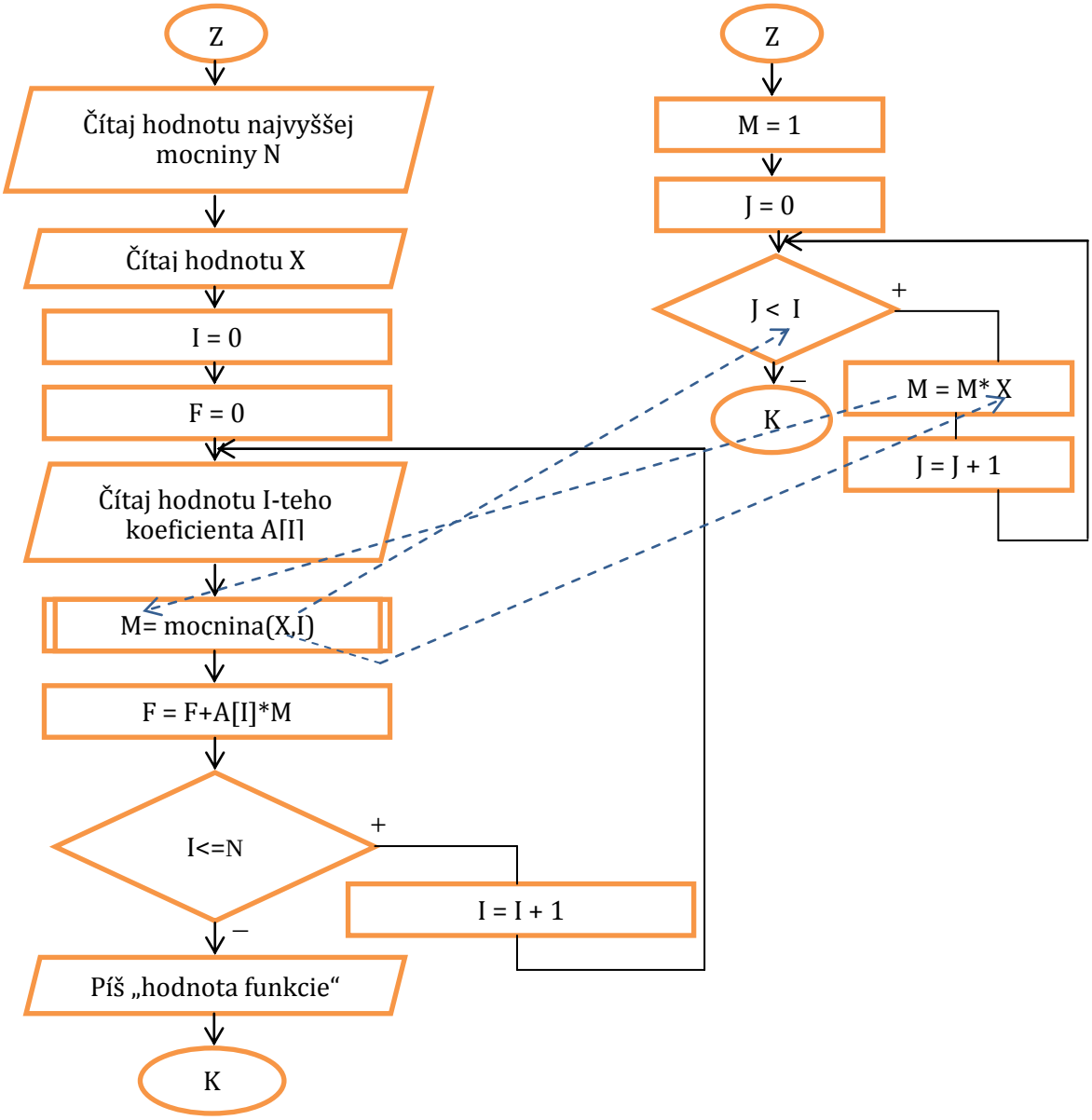
A_0 .... A_n - koeficienty , reálne čísla

X – reálne číslo v ktorom počítame hodnotu funkcie

Výstup:

F – reálne číslo , hodnota danej funkcie v bode X

Vývojový diagram funkcia *mocnina*



Náhrada formálnych parametrov skutočnými pri volaní funkcie *mocnina*

Úloha 4

Návrh algoritmu pre výpočet hodnoty lineárnej rovnice A*X + B = 0

Rozbor problému:
Pri formulovaní vstupných podmienok pre vstupné údaje pri riešení rovnice musíme vychádzať zo skutočnosti, že v matematike (a teda ani počítač) nevieme deliť nulou. V našom prípade k deleniu nulou dôjde vtedy, ak by koeficient *A* mal hodnotu 0. V prípade riešenia rovnice „ručne“ vieme hneď pri zadaní napr. rovnice v tvare *0.X + 4=0* povedať že sa nedá riešiť. Pri riešení úlohy na PC v programe napr. v jazyku C musíme

zadať príkaz *X=-B/A*. V prípade, ak by sme (napr. omylom) zadali pri spustení programu pre koeficient *A* hodnotu 0, program by nám havaroval. Preto musíme v programe „ošetriť“ pri vstupe koeficient *A* tak, aby nemohol byť 0. Najjednoduchší spôsob je použiť cyklus s podmienkou na konci, ktorý vráti proces realizácie algoritmu opäť na krok načítania koeficienta *A*.

Vstup:

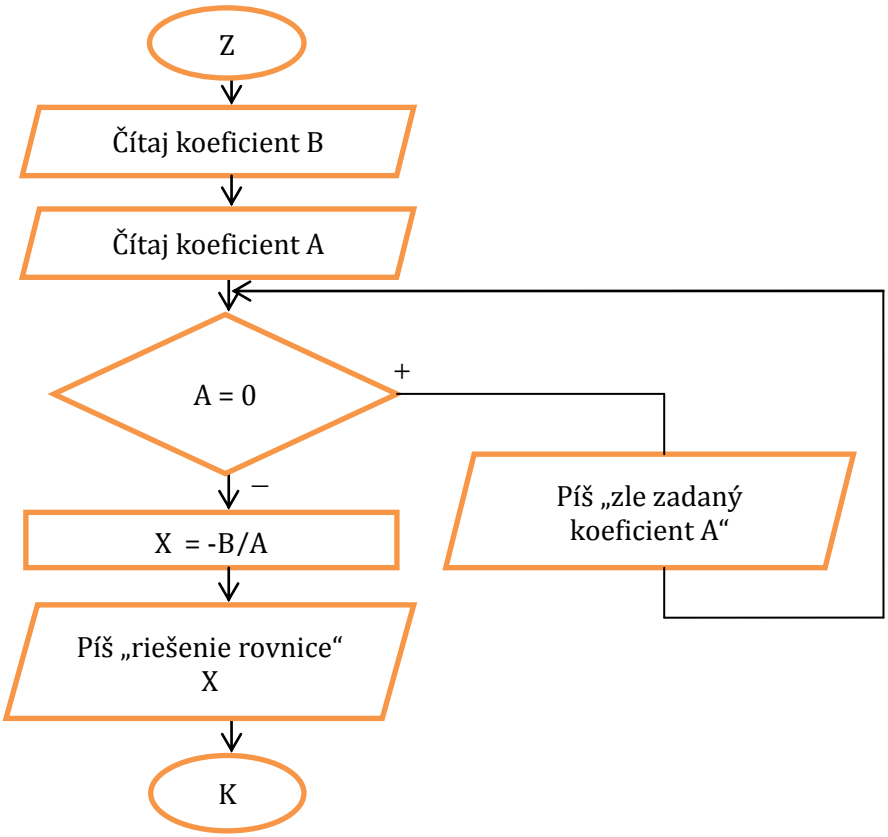
A – reálne číslo rôzne od 0

B - reálne číslo

Výstup:

X – reálne číslo , riešenie lineárnej rovnice

Vývojový diagram



Úloha 5

Návrh algoritmu pre výpočet koreňov kvadratickej rovnice $AX^2 + BX + C = 0$ v obore reálnych čísel.

Rozbor problému:

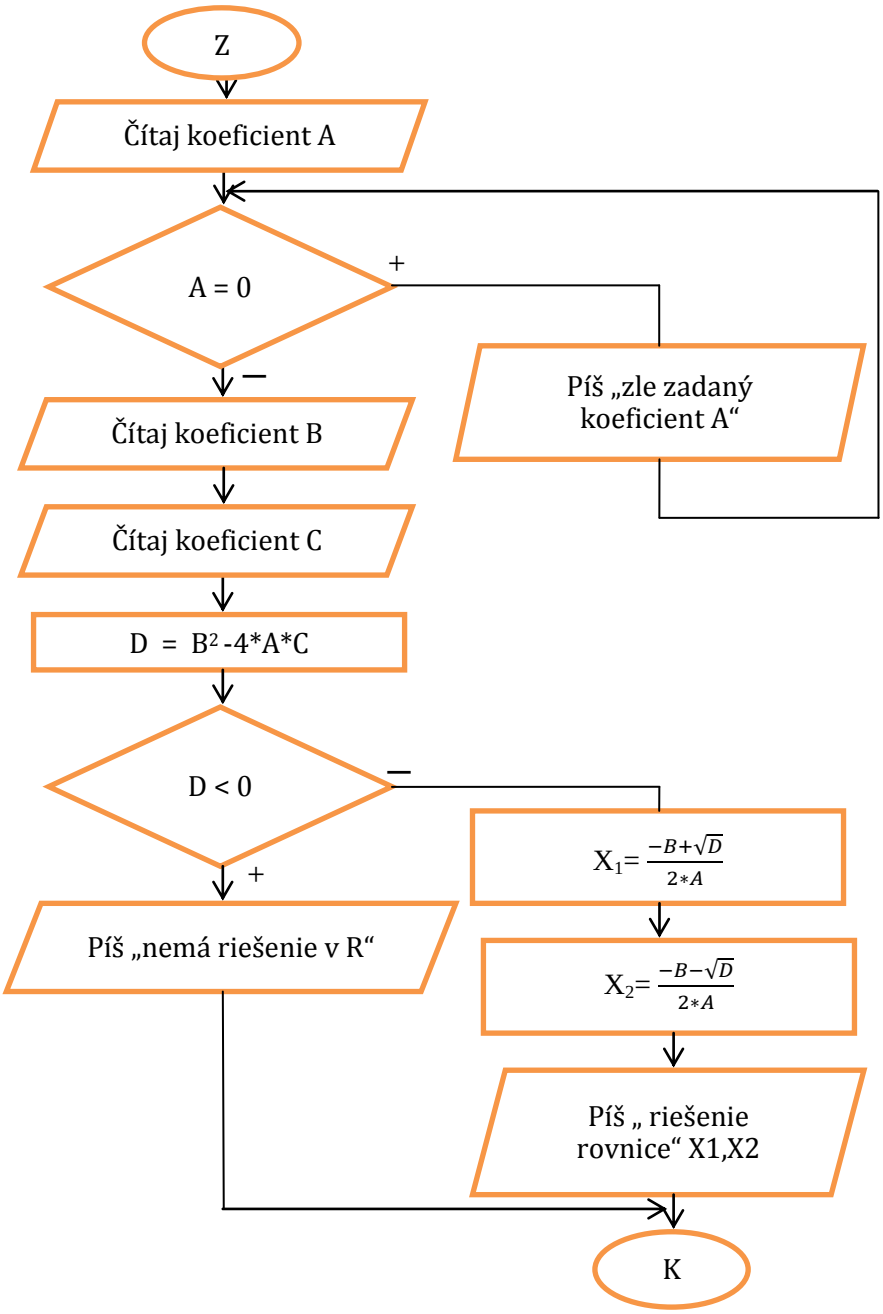
Pri výpočte koreňov kvadratickej rovnice budeme vychádzať zo vzorcov

$X_1=\frac{-B+\sqrt{D}}{2*A}$, $X_2=\frac{-B-\sqrt{D}}{2*A}$ kde D(diskriminant)$D=B^2-4*A*C$.

Z uvedených vzorcov môžeme stanoviť aj vstupné podmienky pre správny výpočet koreňov rovnice. V menovateli zlomku nesmie byť 0. Preto súčin $2*A \neq 0$, teda musí platiť $A \neq 0$. Ak máme riešiť rovnicu v obore reálnych čísel musí platiť že

diskriminant$D \geq 0$. V prípade ak by sa koeficient $A=0$, nebola by to kvadratická rovnica. Pri riešení úlohy na PC (vytvorením programu) však túto skutočnosť /podmienku/ musíme v postupe riešenia použiť. Ak by sme pri riešení kvadratickej rovnice zadali koeficient $A=0$ a nebola by v algoritme podmienka, že $A \neq 0$ program by havaroval.
Vstup:A≠0 reálne číslo, B a C reálne čísla, D>=0
Výstup:X_1, X_2 reálne čísla

Vývojový diagram

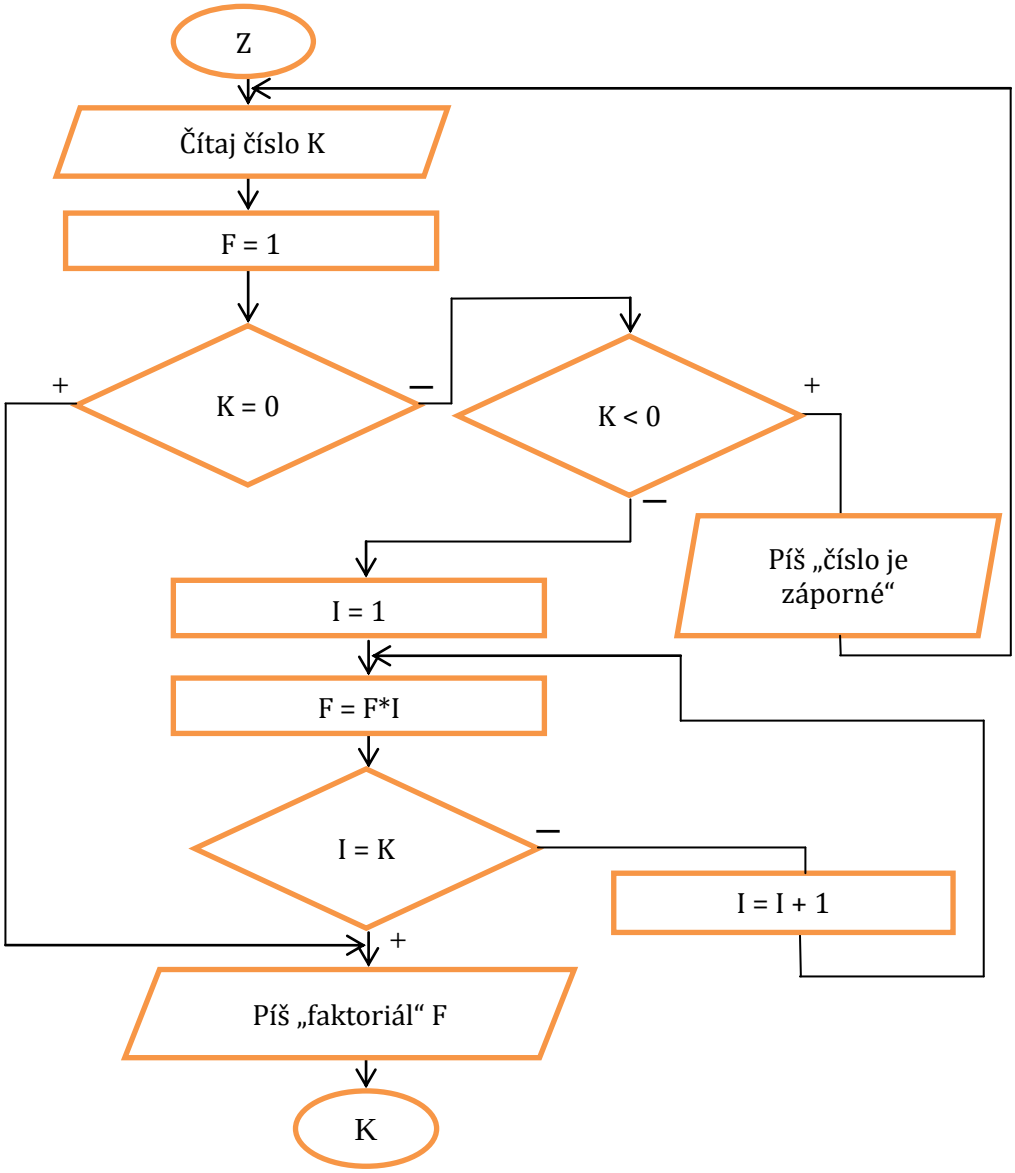


Úloha 6

Návrh algoritmu pre výpočet hodnoty faktoriálu z nezáporného celého čísla K.

Rozbor problému:
Hodnota faktoriálu z čísla *K* sa vypočíta ako súčin postupnosti čísel 1,2,3 .... *K*. Ak súčin budeme ukladať do premennej *F*, potom pre faktoriál z čísla *K* platí *F=1.2.3. .... .K* . Pri návrhu algoritmu nesmieme zabudnúť na fakt, že ak číslo *K=0* , potom faktoriál bude mať hodnotu 1. V prípade ak by bolo zadané číslo K<0, algoritmus musí vypísať o tom oznam a vrátiť proces realizácie naspäť na krok načítania čísla *K*.
Vstupné údaje: *K* - celé nezáporné číslo
Výstup: *F* – celé kladné číslo, kde *F=1.2.3. ..... .K*

Vývojový diagram

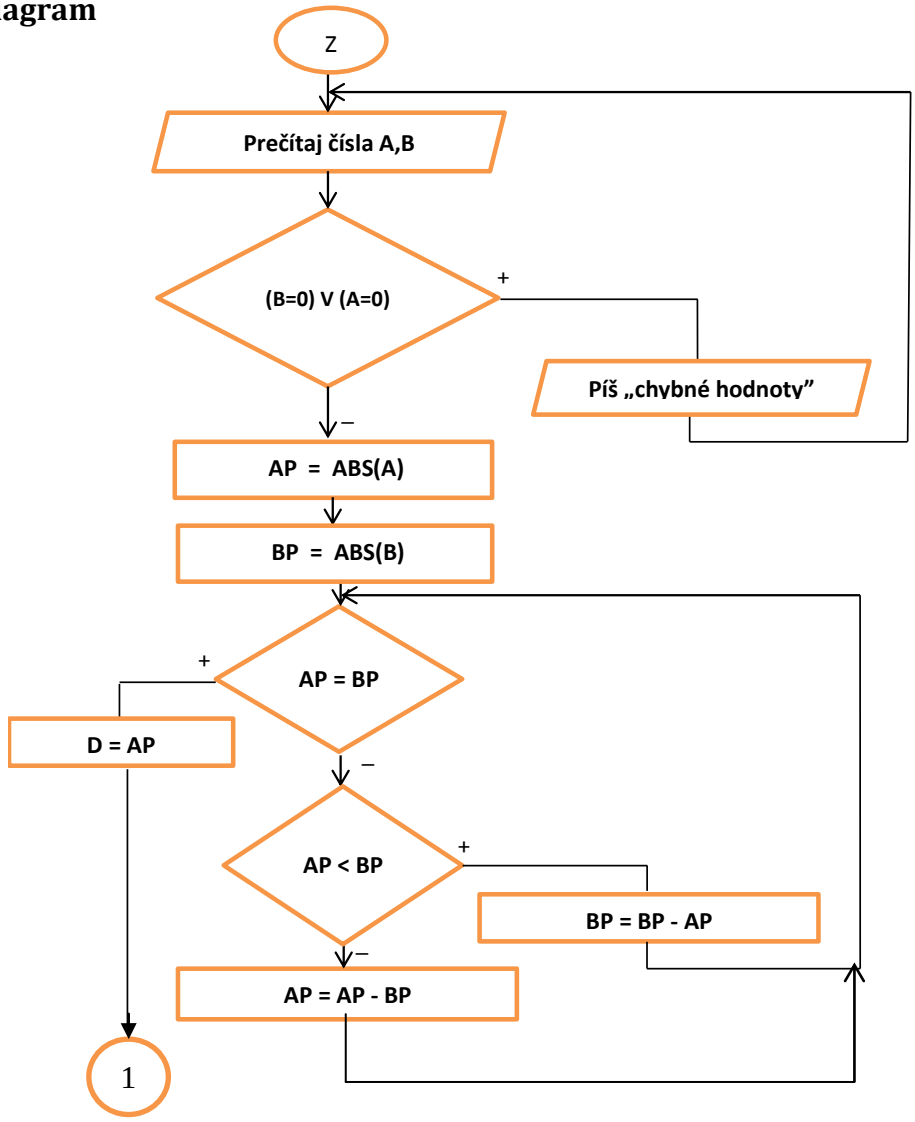


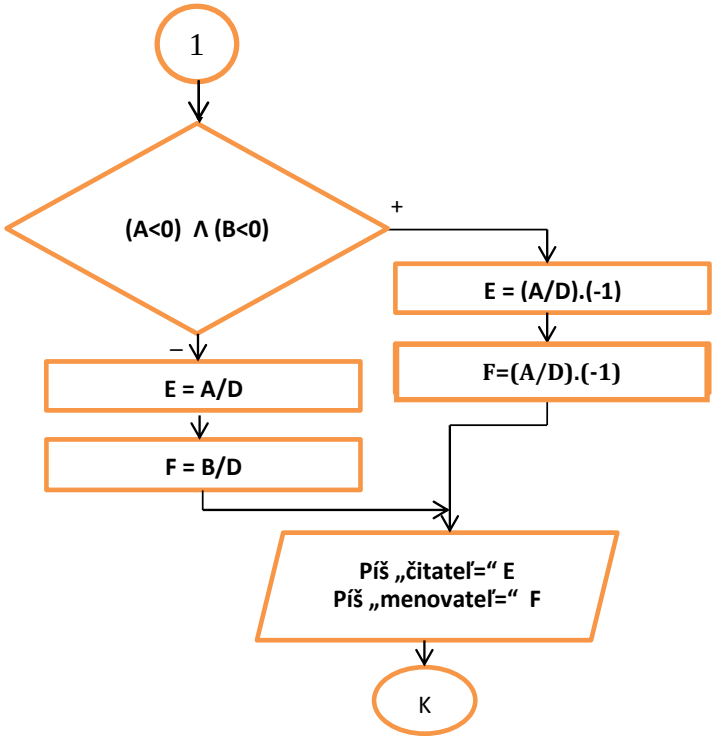
Úloha 7

Návrh algoritmu pre krátenie zlomkov, ktorých čitateľ i menovateľ je celé číslo.

Rozbor problému:
Krátenie zlomku vlastne znamená nájdenie najväčšieho spoločného deliteľa čitateľa a menovateľa zlomku. Pre nájdenie spoločného deliteľa môžeme použiť tzv. Euklidov algoritmus. Ak máme dve prirodzené čísla *A, B* potom podstata tohto algoritmu spočíva v tom, že odpočítavame väčšie číslo od menšieho pokým sú čísla rôzne. Ak výsledkom hľadania spoločného deliteľa bude hodnota 1 (čísla nemajú spoločného deliteľa) potom algoritmus vypíše na výstupe zlomok nezmenený ako na vstupe. Pritom čísla *A, B* musia byť rôzne ako 0. Ak použijeme Euklidov algoritmus, musíme zabezpečiť správnosť algoritmu v prípade, ak niektoré z čísel *A,B* bude záporné. Stačí si na začiatku algoritmu zapamätať znamienko čísel a samotný algoritmus realizovať s absolútnou hodnotou čísel. Ak bude čitateľ aj menovateľ menší ako 0, potom zlomok po krátení bude kladný.
Vstup: *A, B* - celé čísla
Výstup: *E, F* – celé kladné čísla pričom *E=A/D, F=B/D* . D je najväčší spoločný deliteľ čísel *A, B*.

Vývojový diagram



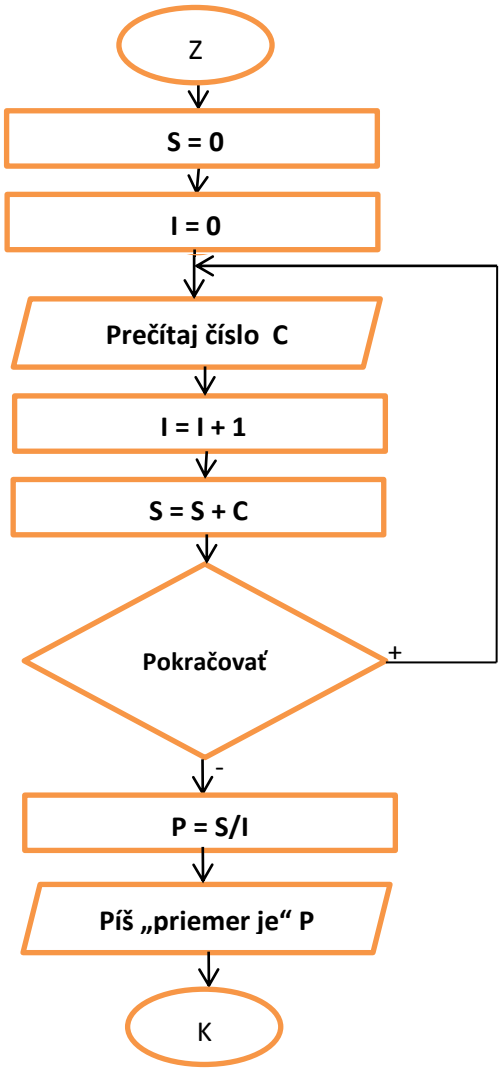


Úloha 8

Návrh algoritmu pre výpočet priemeru z dopredu neznámeho počtu čísel.

Rozbor problému:
K výpočtu priemeru potrebujeme poznať súčet hodnôt z ktorých počítame priemer a ich počet. Ak dopredu nepoznáme počet položiek, môžeme použiť v algoritme cyklus s neznámym počtom opakovaní. Podmienkou ukončenia cyklu môže byť napr. zadanie určitej hodnoty, ktorá bude signalizovať ukončenie zadávania vstupných hodnôt. V algoritme je dôležité definovať pomocnú premennú napr. *I*, ktorej počiatočná nulová hodnota sa bude po zadaní čísla pre výpočet priemeru zvyšovať o hodnotu 1. Po ukončení zadávania vstupných hodnôt bude v tejto premennej počet členov postupnosti a použijeme ju k výpočtu priemernej hodnoty. Ďalej potrebujeme definovať premennú napr. *S* , ktorej počiatočná nulová hodnota sa bude zvyšovať o hodnotu zadávaných čísel pre výpočet priemeru.
Vstup: *S=0, I=0, C* – postupnosť čísel pre výpočet priemernej hodnoty.
Výstup: *P= S/I* - priemerná hodnota z postupnosti čísel C.

Vývojový diagram

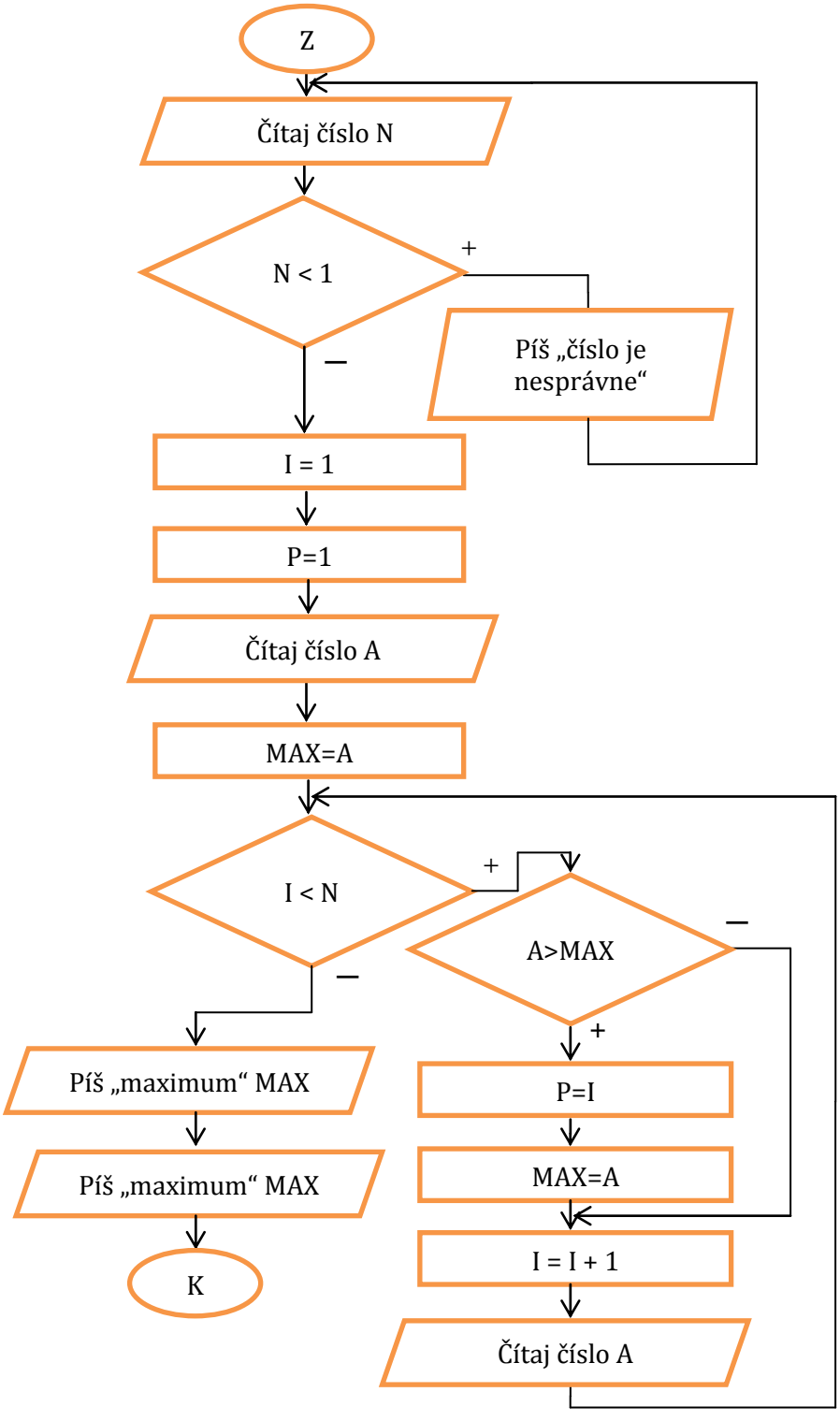


Úloha 9

Návrh algoritmu pre výpis maximálnej hodnoty z postupnosti *N* čísel a poradové číslo maximálneho čísla v postupnosti.

Rozbor problému:
V prípade, ak číslo *N = 1*, teda postupnosť má jeden člen, je tento zároveň aj maximum. V prípade, že*N>1* môžeme na začiatku hľadania maximálneho prvku považovať tiež prvý prvok postupnosti za maximálny. Pre zapamätanie si maximálneho prvku deklarujeme premennú napr. *MAX,* do ktorej na začiatku vložíme hodnotu prvého člena postupnosti. Do premennej *P* priradíme hodnotu *1*. Táto premenná bude obsahovať poradie maximálneho člena. Pri načítaní ďalších členov postupnosti ich hodnotu porovnáme s hodnotou v premennej *MAX*. Ak je táto hodnota väčšia, priradíme ju do premennej *MAX* a poradie tohto prvku (obsah premennej *I*) do premennej *P*. Poradie zadávaných členov budeme evidovať v pomocnej premennej *I*, ktorá po zadaní čísla zvýši svoju hodnotu o 1.Takto pokračujeme pokým nenačítame *N* hodnôt postupnosti.
Vstup: *N* – počet členov postupnosti
A –jednotlivé hodnoty postupnosti .
Výstup: *MAX* - maximálna hodnota z postupnosti čísel.
P- poradie maximálneho prvku v postupnosti

Vývojový diagram



Úloha 10

Návrh algoritmu pre zistenie, či zadané prirodzené číslo *N* je prvočíslo.

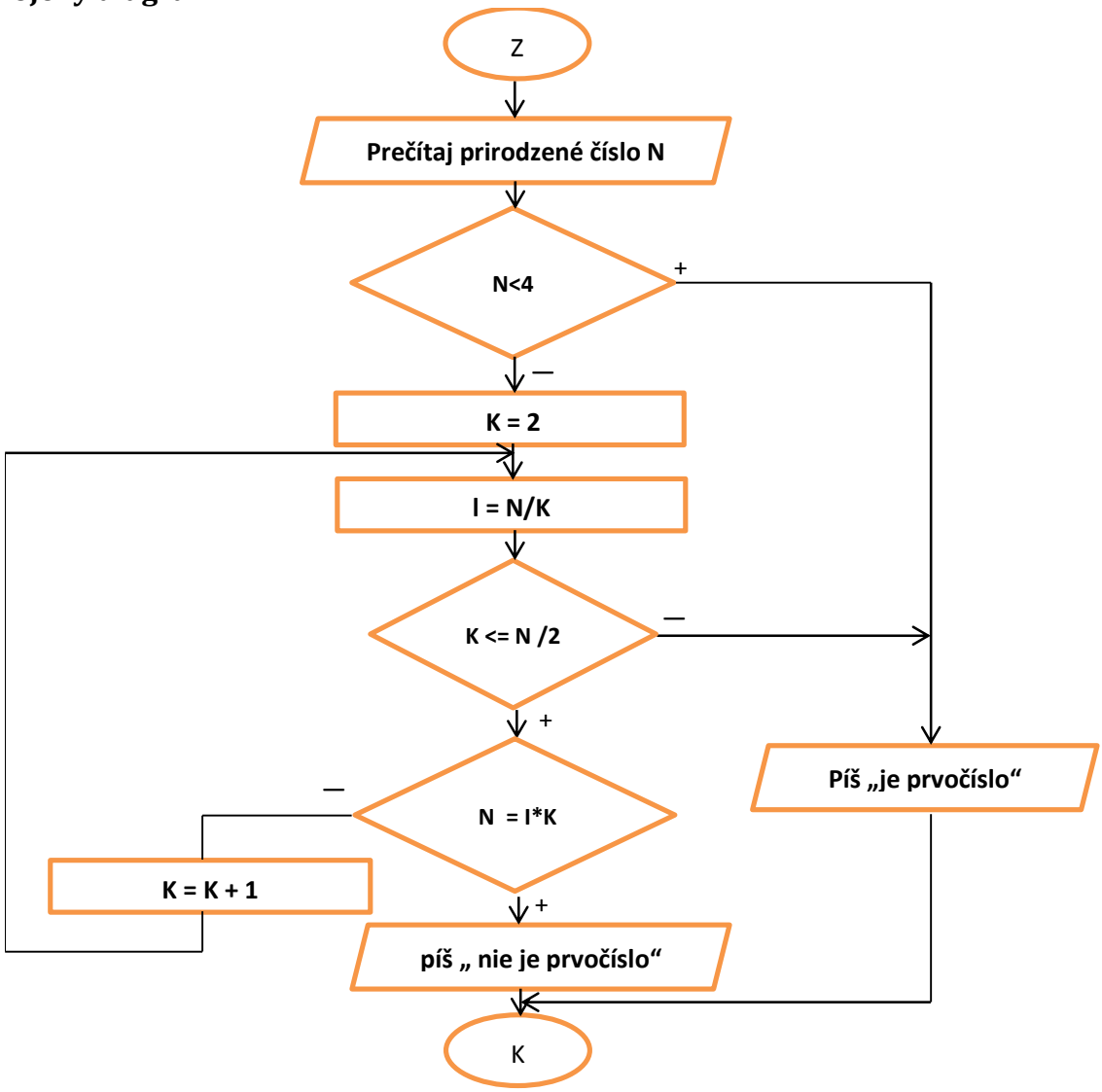
Rozbor problému:
Číslo *N* je prvočíslo, ak neexistuje také číslo *K* < *N* ktorým by číslo *N* bolo deliteľné. Podstata algoritmu spočíva v tom, že postupne pre *K = 4,5,.....,N-1* zisťujeme deliteľnosť čísla *N* číslom *K*. Ak má byť algoritmus efektívny stačí zisťovanie deliteľnosti čísla *N*

číslom *K pre K=4,5 ....N/2*. Ak číslo *N<4* potom je prvočíslo. Pri zisťovaní deliteľnosti použijeme celočíselné delenie. Ak menovateľ aj čitateľ je celé číslo, potom premenná *I=N/K* bude obsahovať výsledok celočíselného delenia čísel *N* a *K*. Je možné použiť aj efektívnejší spôsob - stačí zisťovať delitele do $\sqrt{N}$.

Vstup: N – prirodzené číslo

Výstup: Text : „číslo je prvočíslo“
„číslo nie je prvočíslo“

Vývojový diagram



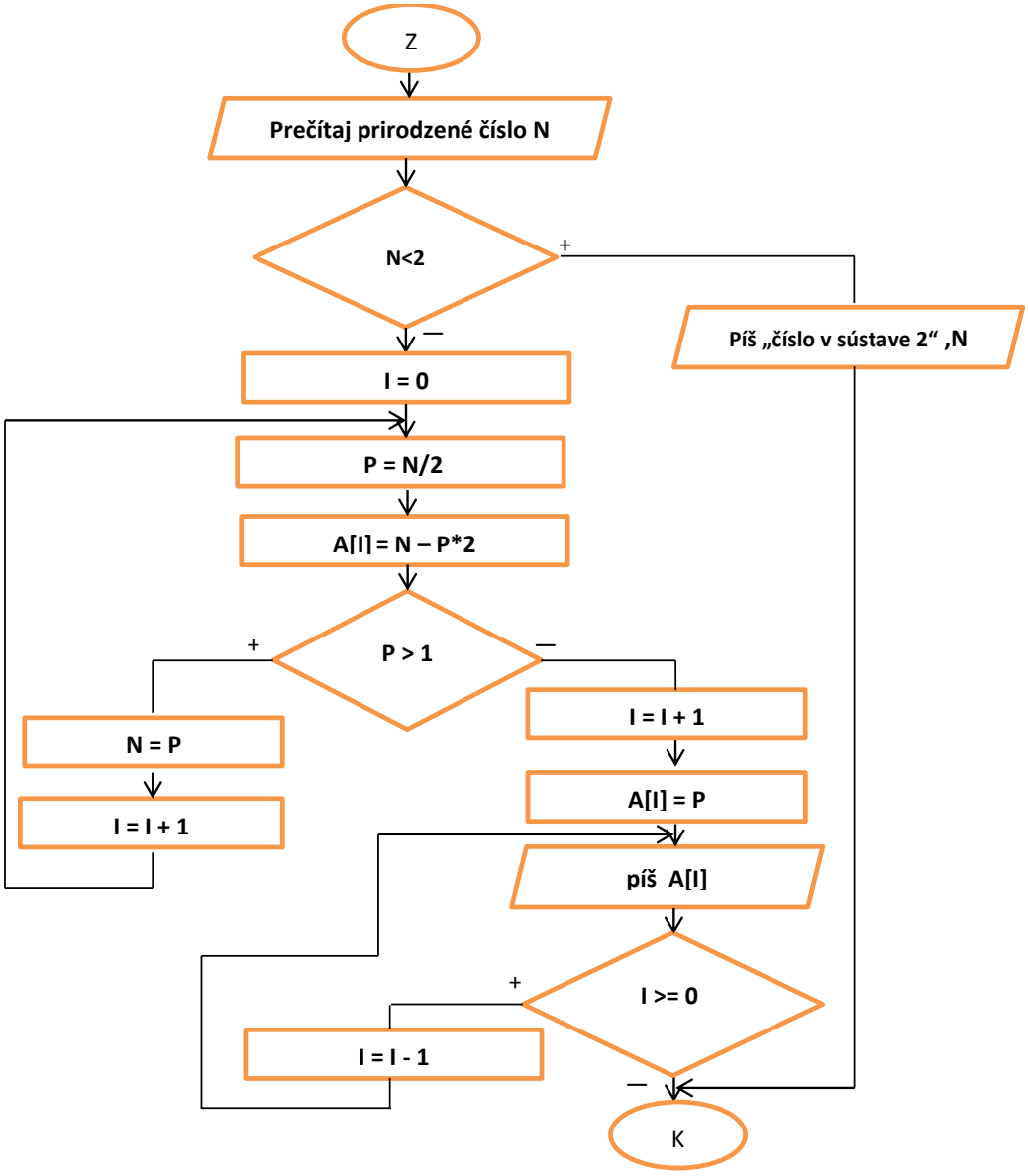
Úloha 11

Návrh algoritmu pre prevod prirodzeného desiatkového čísla do dvojkovej sústavy.

Rozbor problému:
Algoritmus prevodu čísla spočíva v celočíselnom delení desiatkového čísla základom sústavy do ktorej číslo prevádzame. Ak výsledok delenia je väčší alebo rovný ako základ sústavy, delenie sa opakuje s týmto výsledkom dovtedy, pokým nebude výsledok menší ako základ sústavy do ktorej číslo prevádzame. Pritom zvyšky po takomto delení predstavujú jednotlivé číslice prevedeného čísla. Poradie týchto číslic prevedeného čísla je také, že na poslednom mieste (mieste jednotiek) je číslica, ktorú pri realizovaní

podielu získame ako prvú. Preto je výhodné použiť jednorozmerné pole do ktorého postupne budeme ukladať jednotlivé číslice (zvyšky po delení) a po ukončení delenia prvky poľa vypíšeme v opačnom poradí.
Vstup: *N* – prirodzené číslo v desiatkovej sústave
Výstup: $A_1,A_2,....A_k$ - jednotlivé číslice prevedeného čísla.

Vývojový diagram



Úloha 12

Návrh algoritmu pre zistenie či zadané prirodzené číslo *N* je dokonalé číslo.

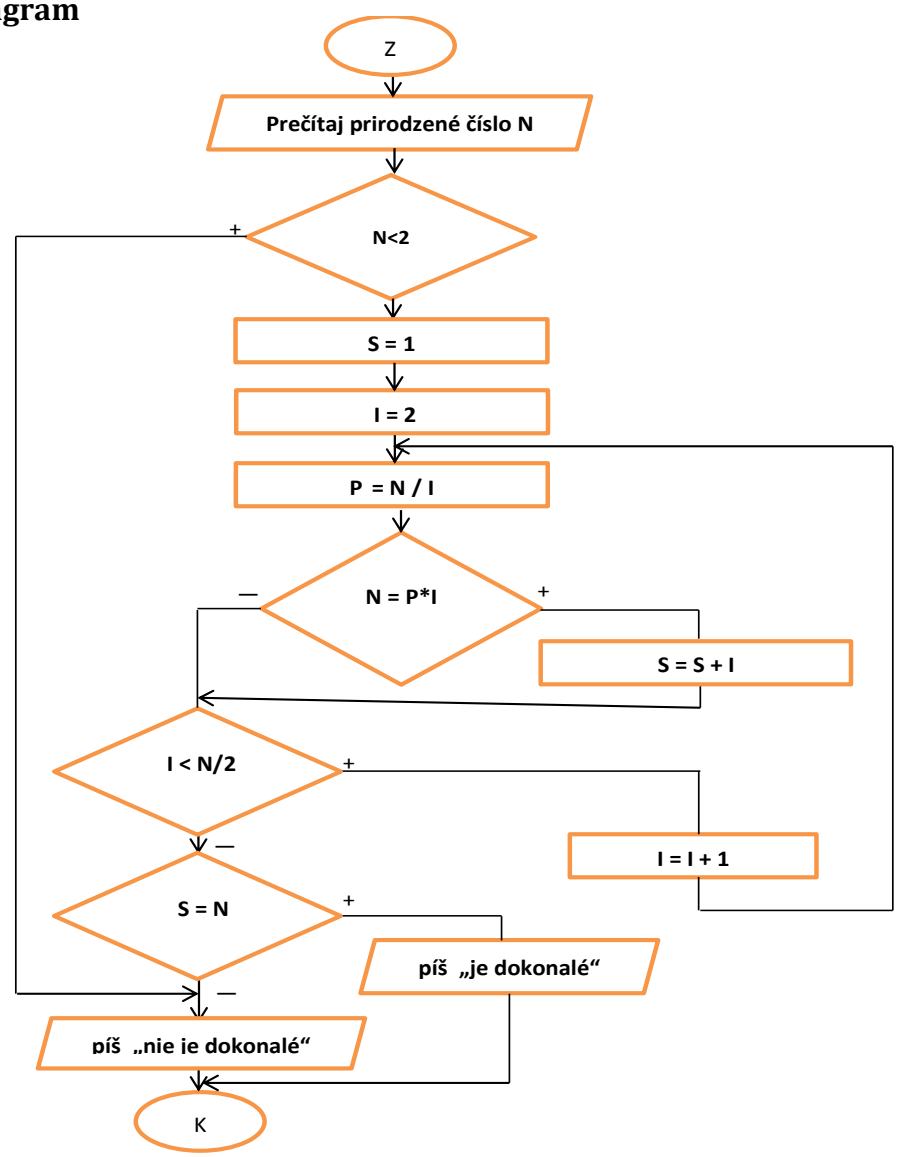
Rozbor problému:
Dokonalé číslo je také prirodzené číslo, ktoré sa rovná súčtu svojich vlastných deliteľov okrem seba samého. Podstata algoritmu teda spočíva v nájdení čísel, ktorými je číslo *N* deliteľné a porovnaním čísla *N* so súčtom jeho deliteľov. Pri návrhu algoritmu je potrebné deklarovať pomocnú premennú napr. *S* s počiatočnou hodnotou 1 a do tejto premennej postupne pripočítavať čísla, ktorými je číslo *N* deliteľné. Počiatočná hodnota

premennej *S=1* je nastavená preto, lebo každé číslo je deliteľné číslom 1. Do premennej *S* je potrebné pripočítať všetky čísla z intervalu *<2,K>*, kde *K=N/2* je výsledok celočíselného delenia čísla *N* hodnotou 2. Zisťovanie deliteľnosti čísla *N* číslami, ktoré sú väčšie ako *K* nemá zmysel. Vyplýva to aj z definície dokonalého čísla. V prípade, ak premenná *S =N* v príslušnom kroku algoritmu sa vypíše oznam, že číslo *N* je dokonalé, inak vypíše oznam, že číslo *N* nie je dokonalé.

Vstup: *N* – prirodzené číslo

Výstup: Text : číslo je dokonalé
číslo nie je dokonalé

Vývojový diagram



Úloha 13

Návrh algoritmu pre zistenie či zadané 3 čísla *A,B,C* môžu byť strany trojuholníka. Ak áno, akého trojuholníka.

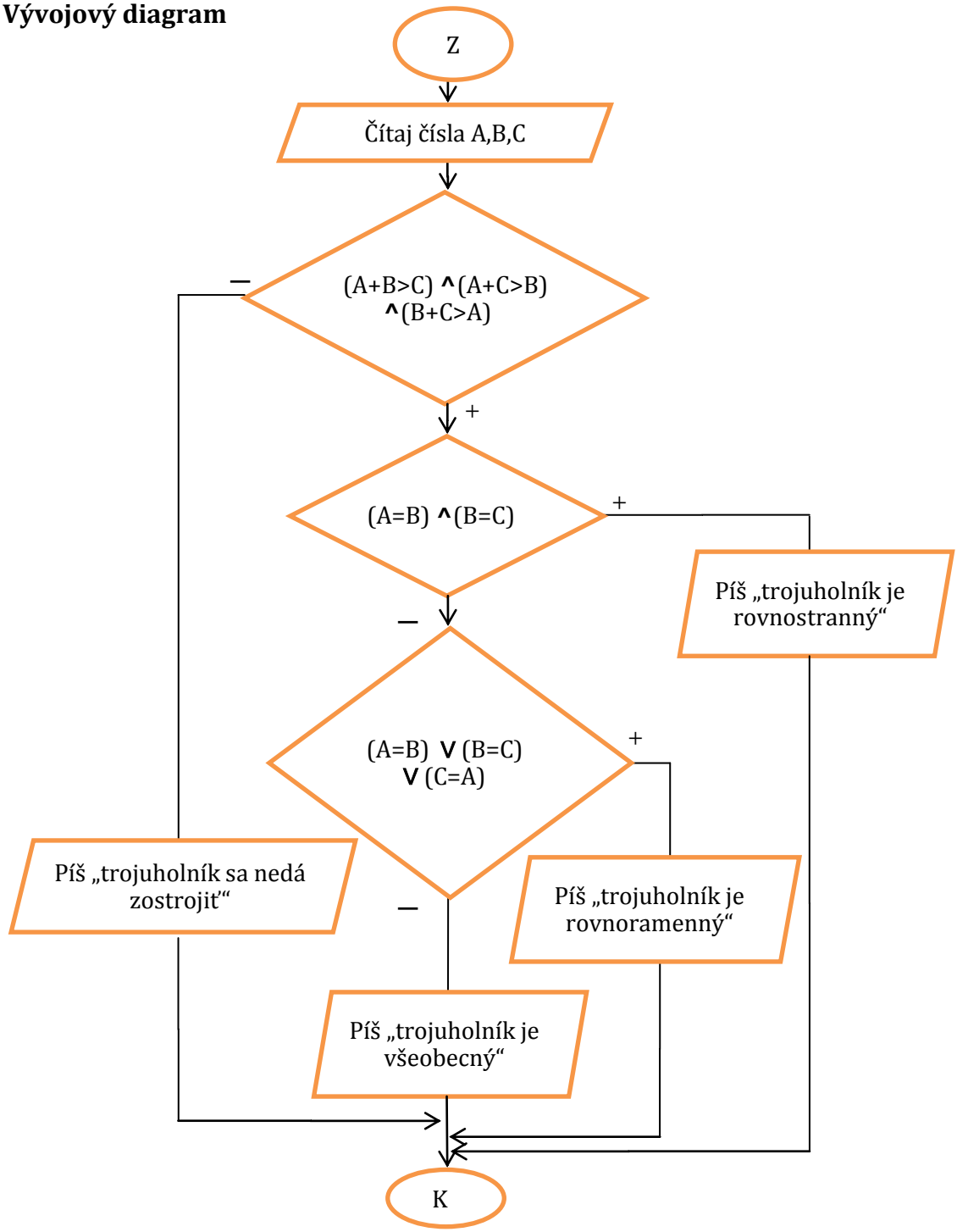
Rozbor problému:
Prirodzené čísla *A,B,C* môžu byť strany trojuholníka, ak platí trojuholníková nerovnosť. Súčet dvoch strán musí byť väčší ako strana tretia. Prvé kroky algoritmu zistia, či platí

trojuholníková nerovnosť. Ak neplatí algoritmus vypíše oznam, že trojuholník sa nedá zostrojiť a skončí. Ak platí, v ďalších krokoch zistíme, aký trojuholník z čísel *A,B,C* môže byť zostrojený.

Vstup: *A,B,C* -reálne čísla

Výstup: Text*:* trojuholník sa nedá zostrojiť
trojuholník sa dá zostrojiť –typ trojuholníka (všeobecný, rovnostranný, rovnoramenný)

Vývojový diagram

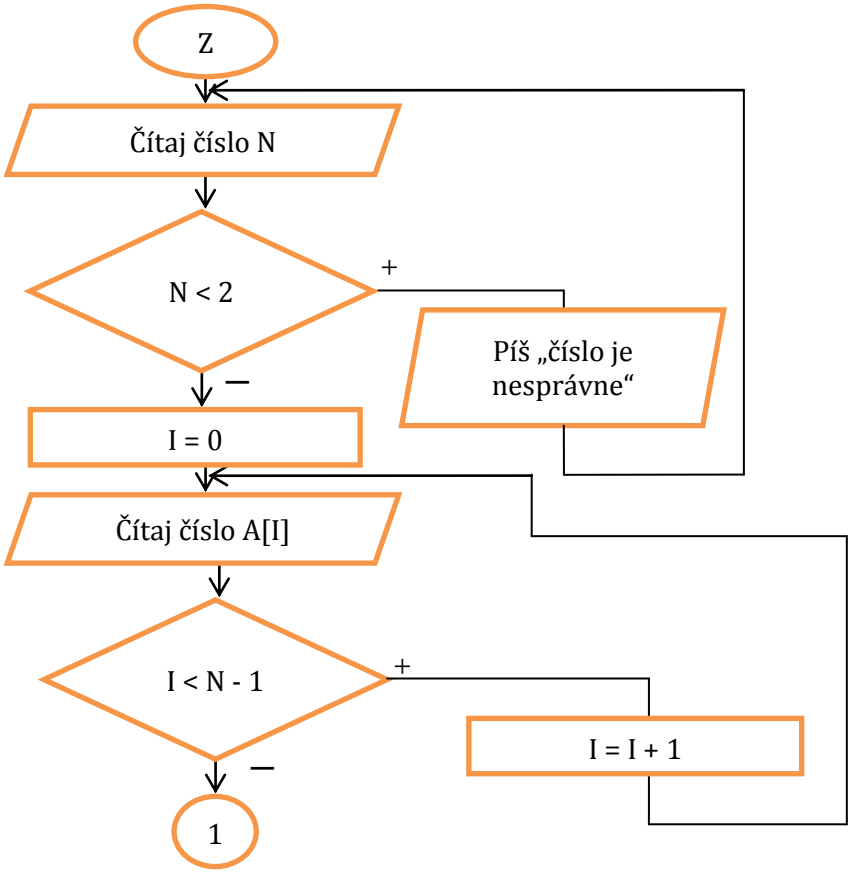


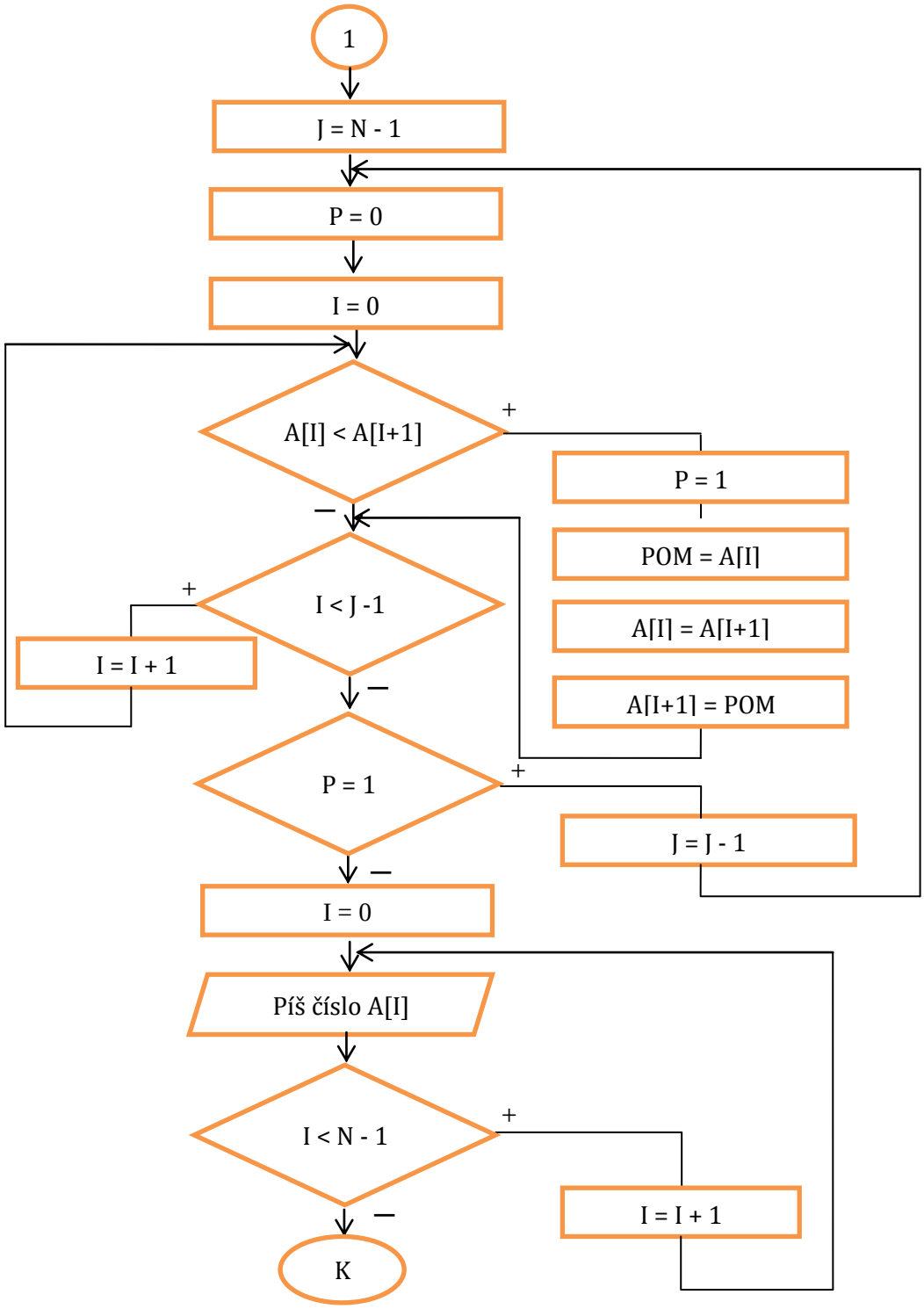
Úloha 14

Návrh algoritmu pre triedenie prvkov v jednorozmernom poli metódou bublesort.

Rozbor problému:
Podstata metódy triedenia je založená na postupnom porovnávaní 2 susedných prvkov poľa a na ich vzájomnej výmene ak nevyhovujú podmienke triedenia. Poznáme triedenie vzostupné a zostupné. Na začiatku prechodu poľom do pomocnej premennej *P* uložíme hodnotu 0. V prípade ak dôjde k výmene susedných prvkov poľa do premennej *P* nastavíme hodnotu 1. Pole bude utriedené vtedy ak pri prechode celým poľom sa neuskutoční ani jedna výmena dvoch susedných prvkov. Vtedy bude mať premenná *P* hodnotu 0, ktorá bola nastavená do premennej *P* na začiatku prechodu cez pole. Efektívnosť tohto algoritmu spočíva v tom, že pri zostupnom triedení v 1 cykle poľa sa najmenší prvok dostane na pravú stranu poľa. Tento prvok už svoju pozíciu nebude meniť čiže nemusíme ho brať do úvahy pri nasledujúcom prezeraní poľa preto môžeme skrátiť dĺžku poľa. Takýto posun najmenšieho prvku a tým skrátenie cyklu sa vykoná v každom jednom cykle.
Vstup: *N* – prirodzené číslo, počet prvkov poľa, *N>1* – vtedy má zmysel pole triediť
$A_1,A_2,.....A_N$ - prvky poľa
Výstup: $A_1,A_2,.....A_N$ - utriedené pole
pri vzostupnom triedení platí: $A_1<A_2<.....A_i<A_{i+1}<.....A_N$
pri zostupnom triedení platí : $A_1>A_2>.....A_i>A_{i+1}>.....A_N$

Vývojový diagram




1
J = N - 1
P = 0
I = 0
A[I] < A[I+1]
+
P = 1
POM = A[I]
A[I] = A[I+1]
A[I+1] = POM
I < J -1
+
I = I + 1
P = 1
+
J = J - 1
I = 0
Píš číslo A[I]
I < N - 1
+
I = I + 1
K

Úloha 15

Návrh algoritmu pre zistenie roku, dňa a mesiaca narodenia zo zadaného rodného čísla.

Rozbor problému:

Rodné číslo má presne danú štruktúru. Môžeme ho symbolicky vyjadriť ako *RRMMDD.* Pritom význam jednotlivých číslic je nasledovný: *RR* – posledné dve číslice z roku napr. 99 znamená rok 1999, *MM*- znamená mesiac napr. 08 znamená mesiac august, a DD – je deň narodenia napr. 23. Pri čísle *MM* v ktorom je informácia o mesiaci narodenia musíme ešte zohľadniť prvú číslicu v mesiaci, v ktorej je zakódovaná informácia o pohlaví osoby, ktorej patrí rodné číslo. Ak je prvá číslica v mesiaci rovná hodnote 5 rodné číslo patrí žene, ktorá sa narodila v mesiaci vyjadrenom druhou číslicou mesiaca *MM*. Podľa pravidiel tvorby rodných čísel sa v rodnom čísle k mesiacu ženy pripočítava hodnota 50. Ak je táto číslica rovná hodnote 0, rodné číslo patrí mužovi narodenom v mesiaci vyjadrenom druhou číslicou mesiaca *MM*. V oboch prípadoch sa jedná o mesiac menší ako 10, pretože musí byť vyjadrený jednou číslicou. V prípade ak je mesiac narodenia väčší ako 9, teda 10, 11 a 12 u mužov toto číslo vyjadruje zároveň mesiac narodenia. Ak je prvá číslica v mesiaci rovná hodnote 6, znamená to, že rodné číslo patrí žene narodenej v mesiaci 10 až 12. Podstata algoritmu na zistenie dátumu narodenia z rodného čísla teda spočíva v rozdelení rodného čísla na tri čísla, z ktorých prvé bude vyjadrovať rok, druhé mesiac a tretie deň. K riešeniu danej úlohy môžeme pristupovať viacerými spôsobmi. Môžeme rodné číslo načítať ako postupnosť ascii znakov. Potom využiť existenciu reťazcových funkcií v každom vyššom programovacom jazyku a postupne z reťazca znakov vybrať príslušnú dvojicu, túto premeniť na číslo a dekódovať podľa vyššie spomenutých pravidiel. Druhý prístup je založený na prevedení určitých matematických operácií pomocou ktorých dokážeme zistiť – vybrať z rodného čísla číslice potrebné k dekódovaniu rodného čísla na dátum narodenia. Ak číslo v tvare *RRMMDD* delíme hodnotou 10000 a jedná sa o celočíselné delenie, dostaneme celé číslo *ROK=RRMMDD/10000*. Toto číslo predstavuje rok narodenia. K tomu aby sme dokázali jednoznačne identifikovať rok, musíme prihliadať aj na číslo uvedené za znakom / v rodnom čísle. Ak je *ROK < 53* a za znakom /, je trojmiestne číslo jedná sa o rok narodenia 1900 a viac. Ak je číslo štvormiestne potom bude rok 2000 a viac. Ak od pôvodného rodného čísla odpočítame hodnotu *ROK* vynásobenú číslom 10000 dostanem číslo *MMDD*. Mesiac z tohto čísla zistíme podobne ako v predchádzajúcom prípade. Vydelíme ho číslom 100. Teda *MEISAC=MMDD/100* - celočíselné delenie. Môžu nastať tieto prípady:

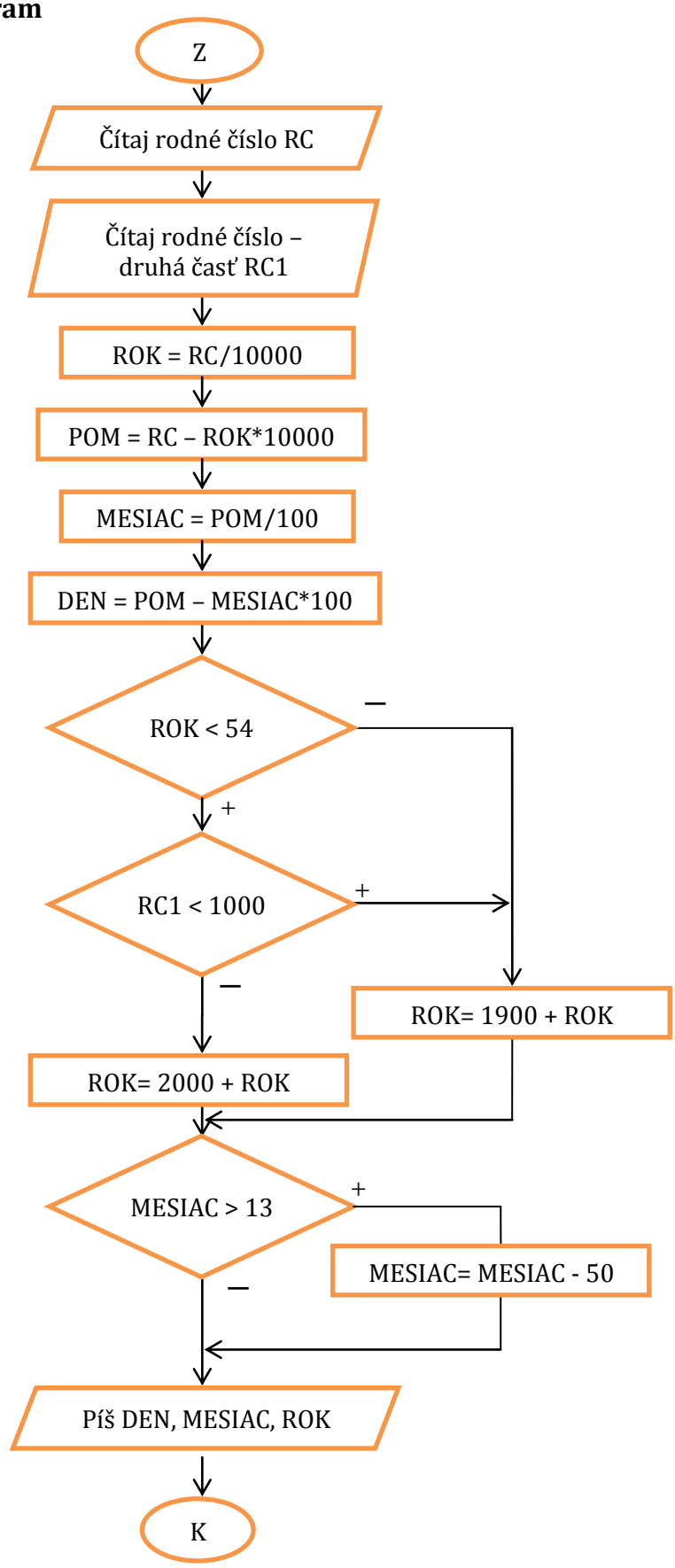
- *MESIAC < 13* potom je to mesiac narodenia muža bez zmeny.
- *MESIAC >50* potom je to mesiac narodenia ženy. Skutočnú hodnotu mesiaca dostaneme odpočítaním hodnoty 50 od položky MESIAC.

Deň získame tak , že súčin premennej *MESIAC* a čísla 100 odpočítame od pôvodného čísla *MMDD*.

Vstup: rodné číslo v tvare *RRMMDD*, prirodzené číslo, alebo reťazec znakov – v tom prípade prvým krokom algoritmu bude prevod reťazca na prirodzené číslo. *POR* – prirodzené číslo – druhá časť rodného čísla (za znakom /).

Výstup: *DEN.MESIAC.ROK* – dátum narodenia z rodného čísla.

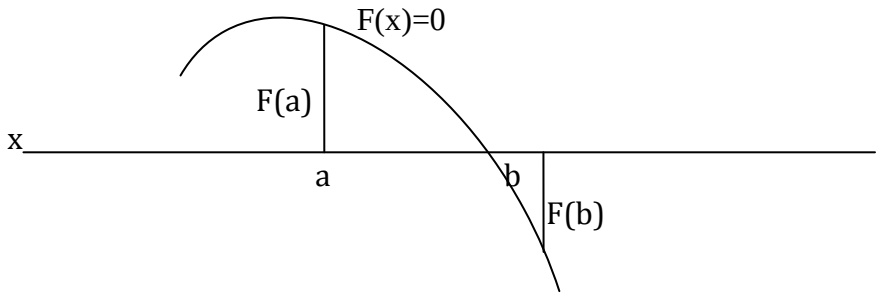
Vývojový diagram



Úloha 16

Návrh algoritmu pre nájdenie približného riešenia algebralickej rovnice F(x)=0 s presnosťou *e* na intervale <*a,b*>.

Rozbor problému:
Ak rovnica F(x) = 0 má riešenie, tak je to v bode, v ktorom funkcia F(x) pretína os x. Z toho vyplýva, že ak tento bod je v intervale<*a,b*> musí byť funkčná hodnota v jednom bode kladná a v druhom záporná. Platí to vtedy ak súčin funkčných hodnôt v týchto bodoch intervalu je menší ako nula ako ukazuje obr. 5.

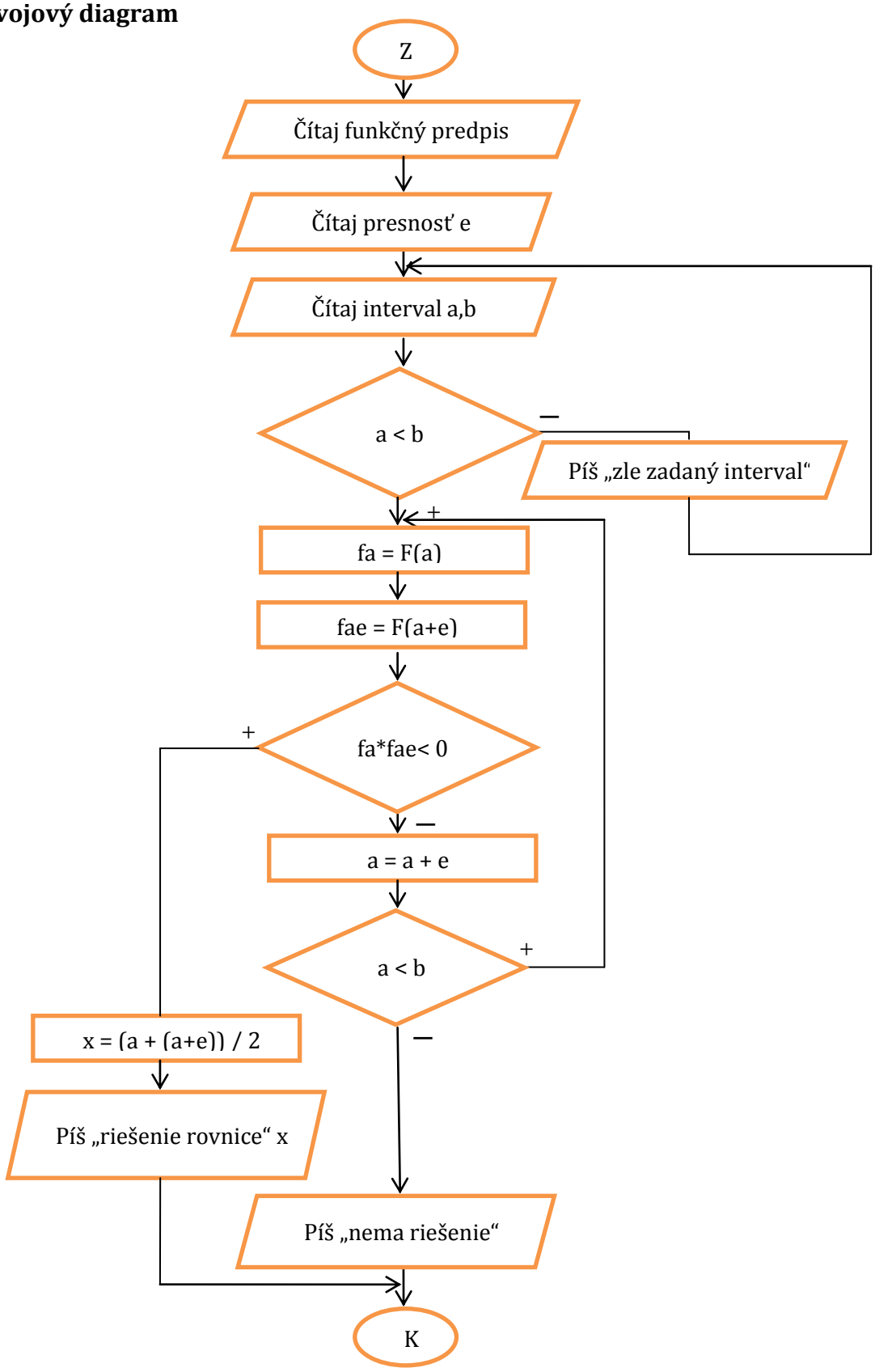


Obr. 5: Graf funkcie na intervale <*a,b*> (vlastný zdroj).

Podstata algoritmu nájdenia približného riešenia s presnosťou *e* spočíva v postupnom zisťovaní funkčných hodnôt *F(a), F(a+e).* Ak súčin týchto funkčných hodnôt je menší ako nula môžeme povedať, že na intervale *<a,a+e>* má rovnica riešenie, pretože ako vidieť aj z obrázka č.5 musí pretínať os *x*. Ak je súčin kladný potom sú obidve funkčné hodnoty *F(a), F(a+e)* kladné, alebo záporné. Teda nepretínajú os *x.* Preto ľavý bod intervalu <*a,b*>, bod *a* zvýšime o hodnotu *e* teda *a=a+e* a opätovne zisťujeme funkčné hodnty *F(a), F(a+e).* Interval *a<b* sa tak neustále zmenšuje. Takto postupujeme pokým je *a<b* a zároveň platí, že súčin *F(a)*F(a+e)>0*. V prípade ak hodnota *a>=b* a súčin *F(a) * F(a+e)>0*, rovnica nemá riešenie na intervale <*a,b*>. Výhodou tohto algoritmu je to, že zistí na intervale <*a,b*> všetky riešenia. Existuje algoritmus (časovo efektívnejší) hľadania riešenia algebralickej rovnice delením intervalu. Tento algoritmus je časovo efektívnejší ale nemusí na intervale <*a,b*> nájsť všetky riešenia. Po načítaní vstupných údajov – hodnôt intervalu *c* je potrebné zaistiť aby platilo *a<b.*V prípade, ak vzťah neplatí, algoritmus sa vráti späť na krok s načítaním hodnôt intervalu. Je možné uvažovať aj s tým, že algoritmus v prípade ak *a>b* vymení navzájom hodnoty intervalu <*a,b*>.

Vstup: *a,b* – čísla, interval v ktorom hľadáme riešenie
e –presnosť s ktorou chceme nájsť riešenie, malé reálne číslo
F(x)– funkčný predpis algebralickej funkcie.
Výstup: *x* –reálne číslo, riešenie rovnice
Text: Rovnica nemá na intervale <*a,b*> riešenie.

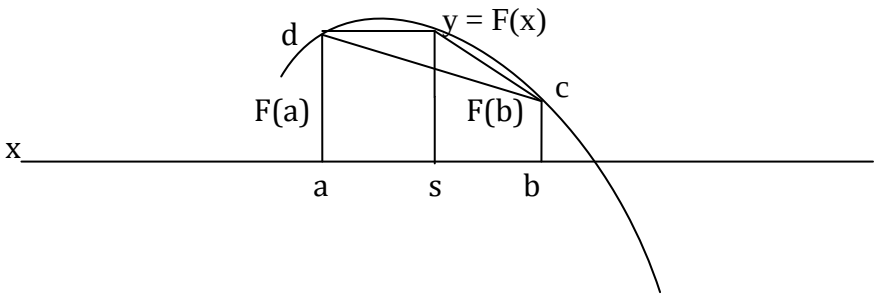
Vývojový diagram



Úloha 17

Návrh algoritmu pre výpočet určitého integrálu funkcie *y= F(x)* na intervale <*a,b*> lichobežníkovou metódou.

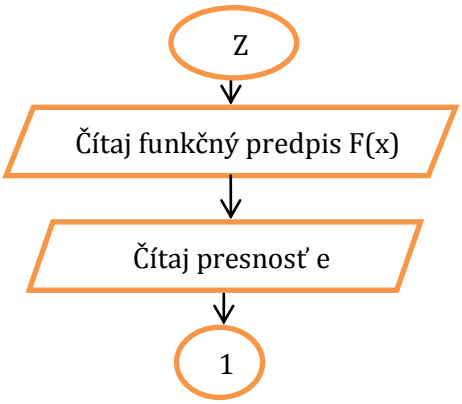
Rozbor problému:
Riešiť určitý integrál znamená vypočítať plochu ohraničenú intervalom <a,b>, grafom funkcie *y=F(x)* a osou x ako ukazuje obrázok č. 6. Reálne použitie metódu približného riešenia určitého integrálu lichobežníkovou metódou je podmienené existenciou počítačového programu, ktorý by realizoval výpočet. Tieto metódy predpokladajú vykonať veľké množstvo výpočtov, ktoré by pri „ručnom" spracovaní nebolo možné.

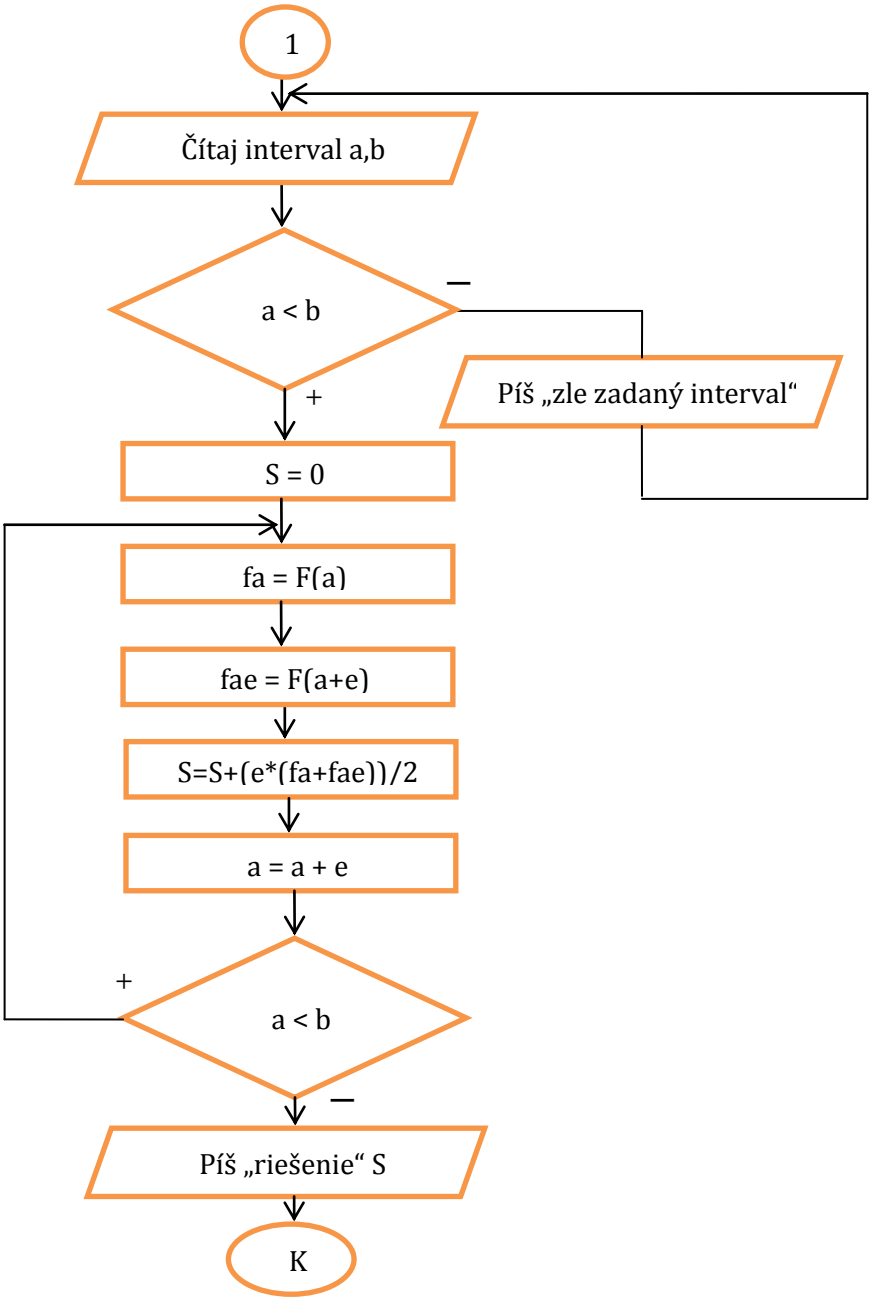


Obr. 6: Učitý integrál funkcie na intervale <*a,b*> (vlastný zdroj).

Ak pripustíme určitú chybu pri výpočte určitého integrálu, stačí vypočítať plochu lichobežníka, ako ukazuje obrázok. Poznáme 3 strany lichobežníka, ktorými sú *b-a, F(a),F(b).* Z týchto známych hodnôt môžeme obsah lichobežníka vypočítať podľa vzťahu *P=((b-a)*(F(a)+f(b)))/2.* Ak rozdelíme interval <*a,b*> plocha lichobežníka sa rozdelí na dva lichobežníky, ktorých súčet obsahov zmenší chybu výpočtu určitého integrálu. Interval <*a,b*> môžeme takto rozdeliť na *M* častí a vypočítať obsah každého z nich. Súčet obsahov takto vzniknutých lichobežníkov predstavuje riešenie našej úlohy. Veľkosť chyby, ktorej sa pri výpočte dopustíme závisí od hodnoty *M*, ktorá rozdelí interval <*a,b*> na malé lichobežníky.
Vstup: *a,b* – čísla, interval na ktorom počítame určitý integrál
e –presnosť s ktorou chceme nájsť riešenie, malé reálne číslo
F(x)– funkčný predpis algebralickej funkcie.
Výstup: *P* – reálne číslo, hodnota určitého integrálu

Vývojový diagram





Úloha 18

Návrh algoritmu pre výpočet BMI indexu (indexu telesnej hmotnosti).

Rozbor problému:
Index telesnej hmotnosti patrí medzi najviac používané metódy merania obezity. Počíta sa ako hmotnosť v kilogramoch delená druhou mocninou výšky v metroch. Vzorec pre výpočet:$BMI=M/V^2$.Podľa vypočítanej hodnoty BMI sa určí stupeň obezity resp. nadváhy a normálnej váhy. Pri zadaní vstupných údajov algoritmus prevedie čiastočnú kontrolu vstupných údajov, ktoré nesmú byť záporné.

Vstup: *M* – prirodzené číslo udávajúce váhu v kg.

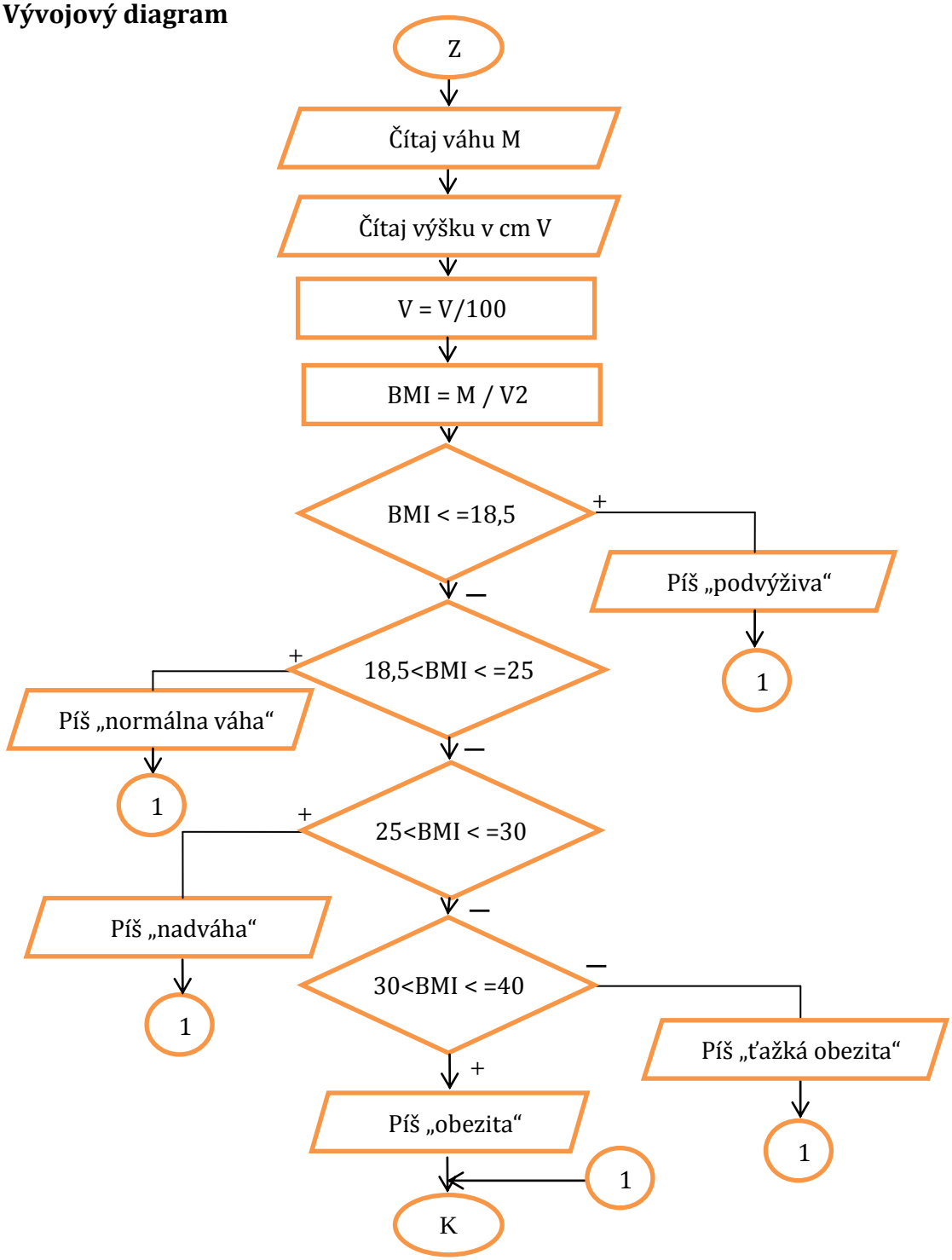
V – prirodzené číslo -výška v cm. Do vzorca pre výpočet sa zadáva výška v metroch. Vzhľadom k tomu, že výška sa najčastejšie udáva v centimetroch,

na vstupe použijeme centimetre a algoritmus prevedie prepočet centimetrov na metre, ktoré sa použijú pri výpočte

Výstup: BMI=M/V²*P* – reálne číslo, hodnota indexu.

BMI<= 18,5	podvýživa
18,5<BMI<=25	normálna váha
25<BMI<=30	mierna nadváha
30<BMI<=40	obezita
BMI>40	ťažká obezita

Vývojový diagram

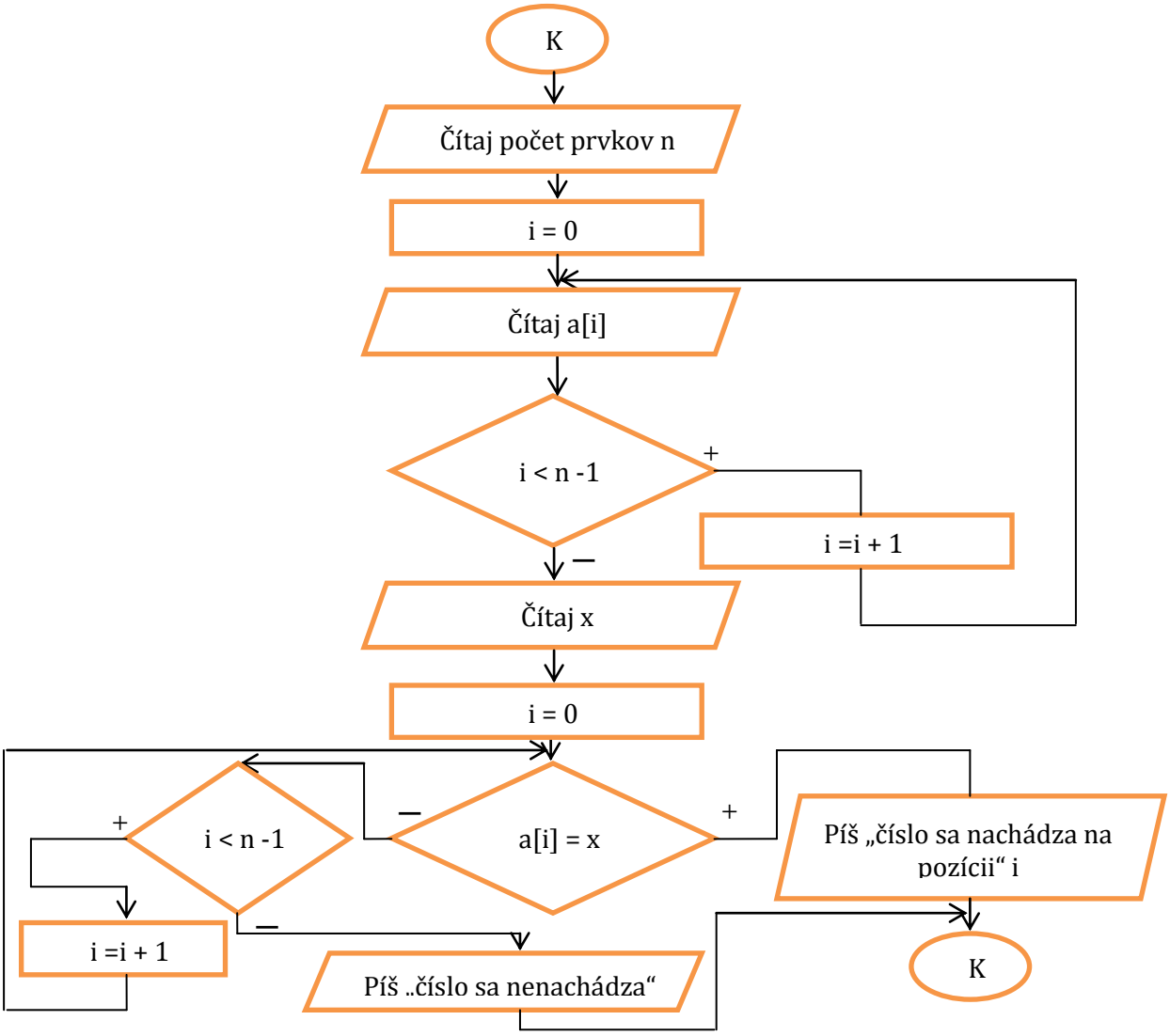


Úloha 19

Návrh algoritmu pre vyhľadanie prvku v neutriedenom poli.

Rozbor problému:
Pri spracovaní údajov je veľmi často potrebné vyhľadávať konkrétnu hodnotu v nejakej množine údajov, ktoré môžu a nemusia byť utriedené. Od toho aj závisí efektívnosť algoritmu vyhľadávania. Najjednoduchší, ale aj málo efektívny je algoritmus sekvenčného prehľadávania v neutriedenom poli. Rýchlosť nájdenia hľadaného prvku záleží od polohy prvku v poli. Podstata algoritmu je v postupnom - sekvenčnom prehľadávaní prvkov poľa, z ktorých každý sa porovná s hľadanou hodnotou. V prípade, ak sa nájde zhoda, cyklus prehľadávania končí, inak sa po prechode celým poľom vypíše oznam, že hľadaná hodnota sa v poli nenachádza.
Vstup: a_1, a_2..... a_n – neutriedené pole reálnych čísel
n – prirodzené číslo, udáva počet prvkov v poli
x - hľadaná hodnota v poli a
Výstup: p – prirodzené číslo – poradie nájdeného prvku v poli
Text: číslo sa v poli nachádza
číslo sa v poli nenachádza

Vývojový diagram

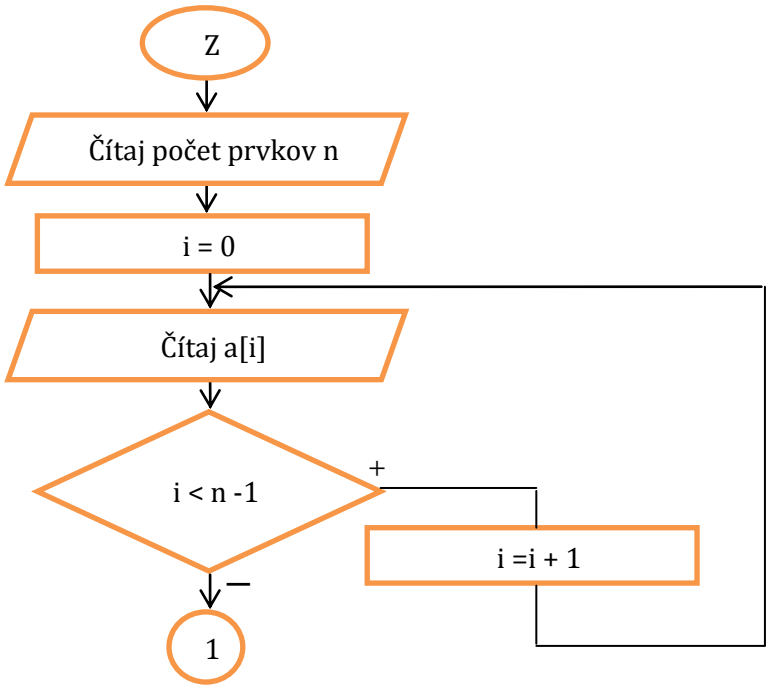


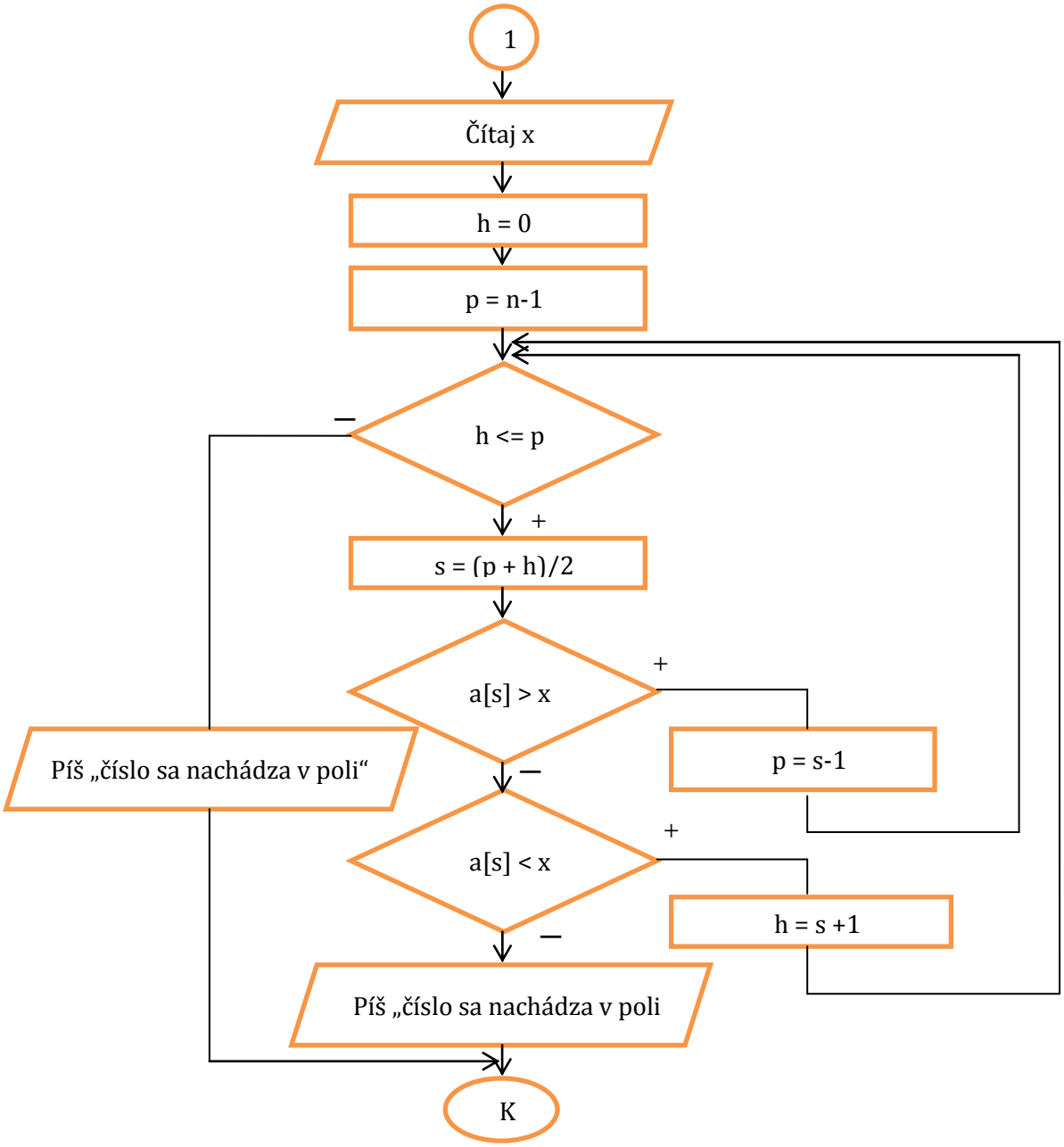
Úloha 20

Návrh algoritmu pre vyhľadanie prvku v utriedenom poli.

Rozbor problému:
Veľmi efektívnym algoritmom vyhľadávania určitej hodnoty v utriedenom poli je metóda binárneho vyhľadávania. Metóda spočíva v postupnom delení poľa na dve časti. Podľa počtu prvkov poľa algoritmus určí aritmetický stred poľa. Porovnaním z hľadanou hodnotou sa tá časť poľa v ktorej vzhľadom k utriedenému poľu hľadaná hodnota nemôže byť , vynechá z ďalšieho vyhľadávania. Ak bol hľadaný prvok náhodou presne v strede poľa, algoritmus sa úspešne ukončí a ďalšie prehľadávanie nie je nutné.
Ak sa v strede poľa nachádzal prvok väčší ako hľadaný, je zrejmé, že potom bude niekde v ľavej polovici poľa, teda medzi prvkami menšími ako je aktuálny prvok. Ak je v strede poľa prvok menší ako hľadaný, bude sa hľadaný prvok nachádzať v pravej polovici poľa. Keďže takto pri každom porovnaní vieme s istotou určiť, že v niektorej z polovíc sa hľadaný prvok určite nenachádza, pokračujeme vo vyhľadávaní iba v tej druhej polovici, ktorá bude pre algoritmus od toho momentu novým vyhľadávacím poľom. Hranice tohto poľa sú pôvodná ľavá hranica poľa a stred – 1 prvok alebo stred + 1 prvok a pôvodná pravá hranica poľa podľa toho, v ktorej polovici poľa má vyhľadávanie pokračovať. Opakovaním tohto postupu buď nájdeme hľadaný prvok, alebo sa postupne pravá a ľavá hranica budú približovať až dovtedy, pokiaľ sa nestotožnia a nebudú na rovnakom mieste v poli.
Vstup: $a_1, a_2 ..... a_n$ – utriedené pole reálnych čísel
n – prirodzené číslo, udáva počet prvkov v poli
x - hľadaná hodnota v poli a
Výstup: p – prirodzené číslo – poradie nájdeného prvku v poli
Text: číslo sa v poli nachádza
číslo sa v poli nenachádza

Vývojový diagram





Úloha 21

Návrh algoritmu pre výpočet skalárneho súčinu dvoch vektorov.

Rozbor problému:
Skalárny súčin dvoch vektorov predstavuje súčin veľkostí vektorov a kosínusu uhla, ktorý navzájom zvierajú. Pri riešení tejto úlohy uvažujme s n rozmerným priestorom. Každý vektor musí mať teda n prvkov. Ak máme dva vektory $\vec{a},\vec{b}$ potom skalárny súčin vypočítame podľa vzťahu :$s = |a| * |b| * cos\ \varphi$, alebo $s = a_1*b_1 + a_2*b_2 ....+a_n*b_n$ .Ak sú dva vektory kolmé, ich skalárny súčin je rovný nule, pretože cos 90° = 0. Pri výpočte skalárneho súčinu zadaním uhla ktorý zvierajú, potrebujeme vypočítať ešte veľkosť vektora. Ten sa vypočíta podľa vzťahu : $|a|=\sqrt{(a_1^2 + a_2^2 +..a_n^2)}$,$|b|=\sqrt{(b_1^2 + b_2^2 +..b_n^2)}$. Pri návrhu algoritmu musíme ošetriť vstupné údaje. Prvky vektora môžu byť reálne

čísla. Počet prvkov *n* musí byť prirodzené číslo. Obidva vektory musia mať rovnaký počet prvkov.

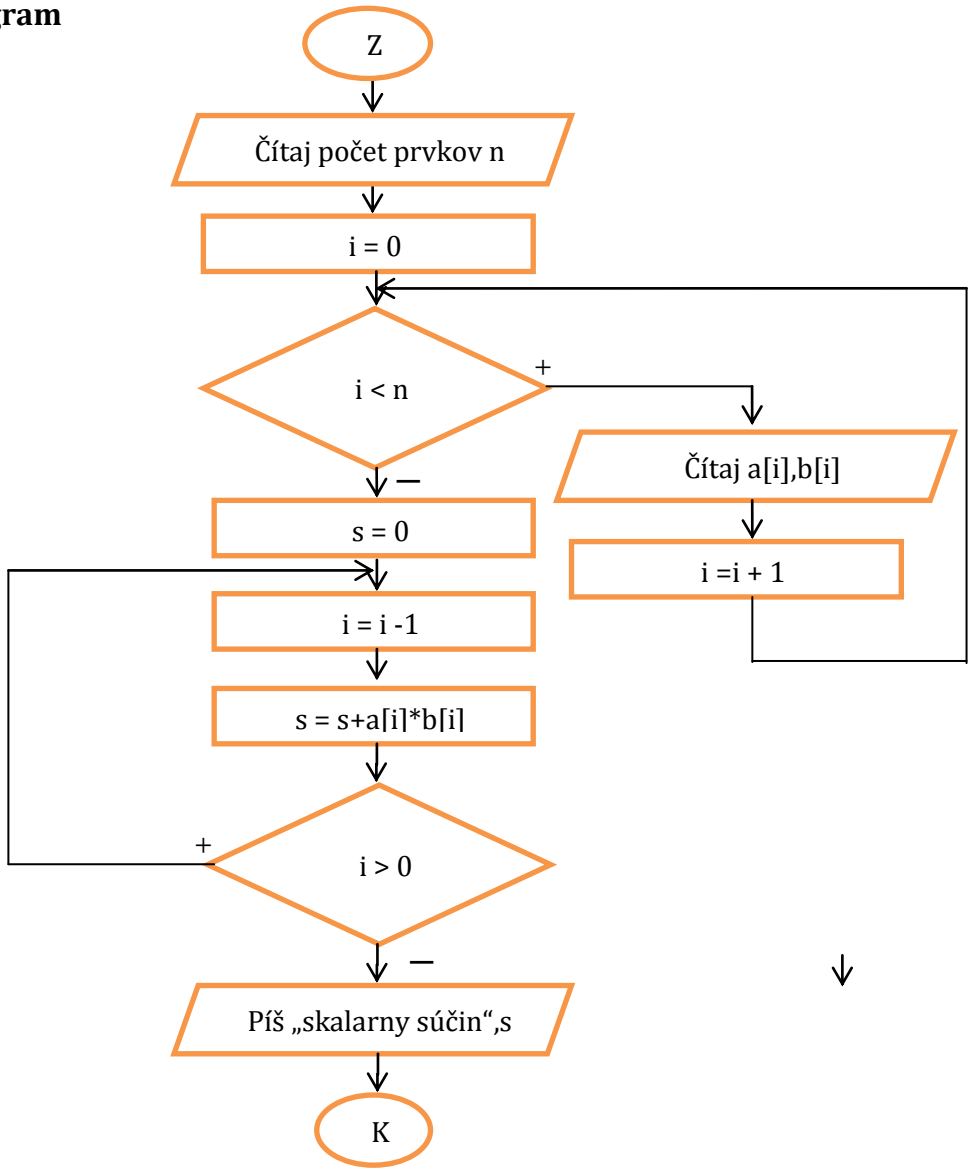
Vstup: *n* – prirodzené číslo udávajúce počet prvkov vektorov.

$a_1,a_2,.....a_n$- jednotlivé prvky vektora $\vec{a}$, reálne čísla

$b_1,b_2,.....b_n$- jednotlivé prvky vektora$\vec{b}$, reálne čísla

Výstup: s –reálne číslo skalárny súčin dvoch vektorov

Vývojový diagram



Úloha 22

Návrh algoritmu pre výpočet počtu platidiel z postupnosti *m* hodnôt.

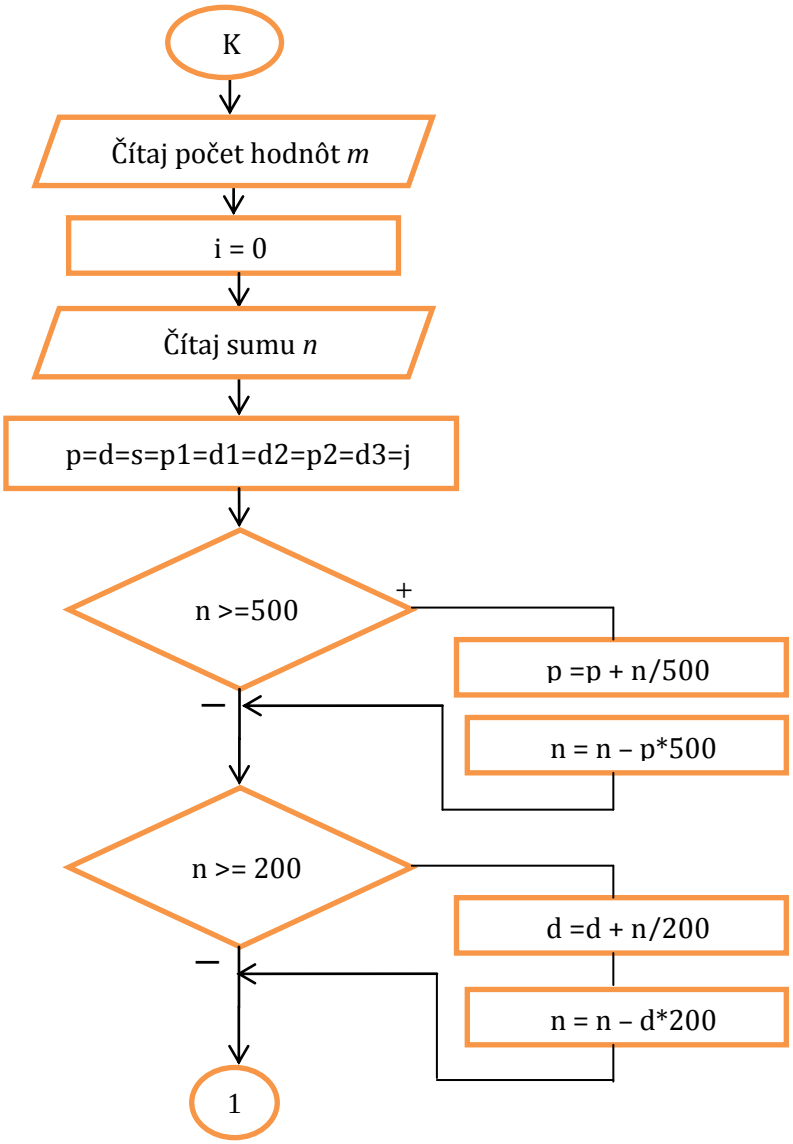
Rozbor problému:

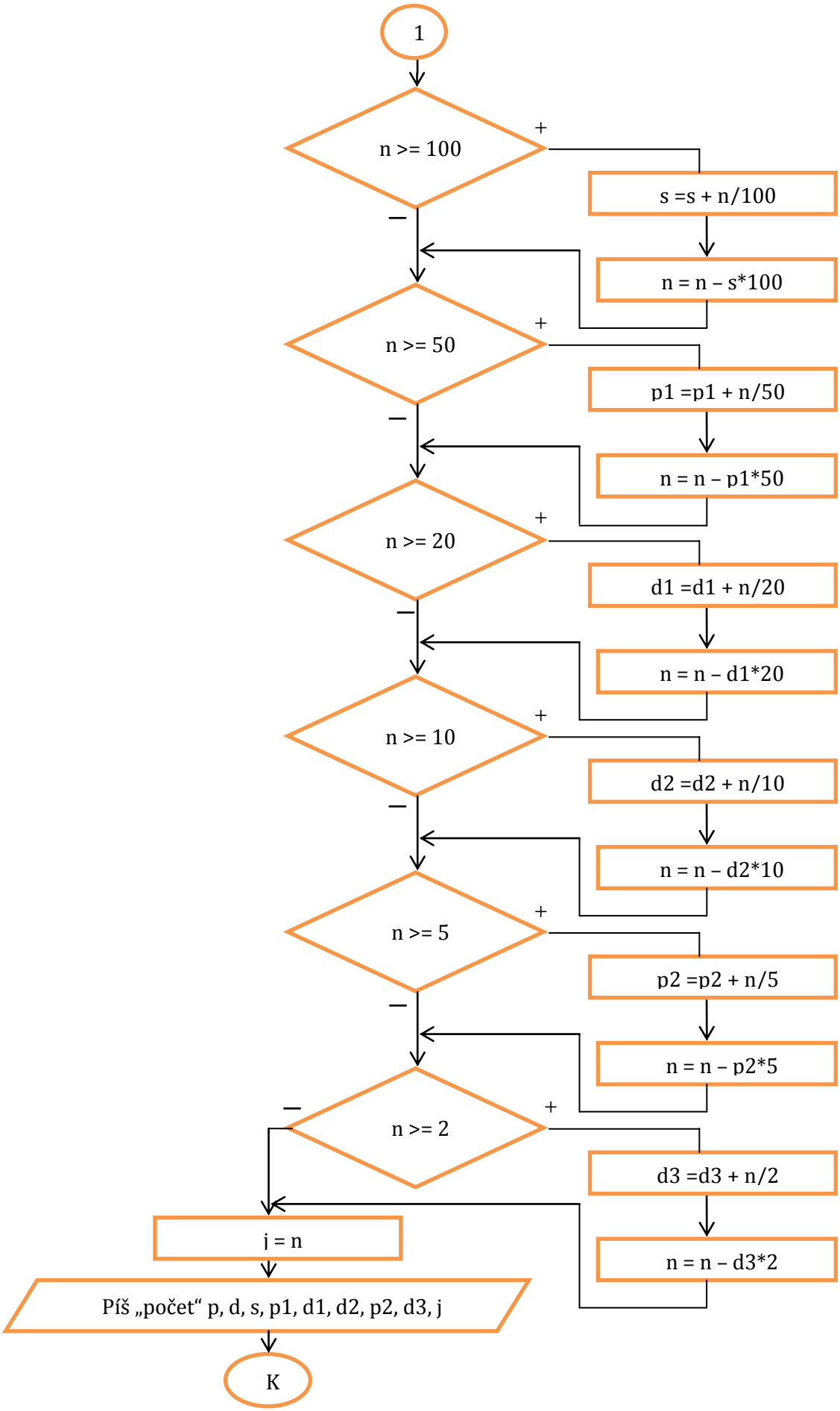
Nech *n* je prirodzené číslo, ktoré vyjadruje určitú sumu peňazí. Ak predpokladáme našu menu v eurách, potom má algoritmus zistiť počty jednotlivých platidiel (bankoviek -500, 200, 100, 50, 20, 10, 5 a mincí – 2, 1), ktorých súčet sa bude rovnať zadanej sume *n*. Pri návrhu algoritmu uvažujeme z možnosťou chybného zadania vstupných hodnôt. Musí platiť *n>0.*Pre *n=0* nemá zmysel zisťovať počty platidiel lebo všetky sa budú rovnať 0. Ak

v algoritme použijeme celočíselné delenie, potom príkaz *p=n/500* priradí premennej *p* hodnotu ktorá vyjadruj počet 500 eurových bankoviek v sume *n*. Po odpočítaní hodnoty vyjadrenej súčinom počtu 500-eurových bankoviek (premenná *p*) a hodnoty 500, od pôvodnej hodnoty *n* dostaneme novú hodnotu premennej *n=n-500*p* . Z tejto novej hodnoty *n* budeme rovnakým spôsobom zisťovať počet 200, 100 .....5, 2, 1 eurových platidiel. V prípade ak na vstupe máme *m* hodnôt (napr. mzda viacerých pracovníkov), deklarujeme premenné *p, d, s, p1, d1, d2, p2, d3, j*na začiatku s hodnotou 0. Do premenných budeme pripočítavať počet jednotlivých druhov platidiel v poradí od 500,200 ..... 5, 2, 1 pre každú z *m* zadaných hodnôt. Algoritmus teda v cykle od 1 po *m* bude načítavať hodnoty do premennej *n* a pre každú hodnotu vypočíta počet jednotlivých platidiel meny, ktorý pripočíta k príslušnej premennej *p, d, s, p1, d1, d2, p2, d3, j.*

Vstup: *m* -počet čísel *n* pre výpočet počtu platidiel, *m>0*
n - hodnota z ktorej počítame počet platidiel, *n>0*
Výstup: *p, d, s, p1, d1, d2, p2, d3, j* prirodzené čísla vyjadrujúce počty jednotlivých platidiel v poradí 500, 200, 100, 50, 20, 10 ,5, 2, 1.

Vývojový diagram



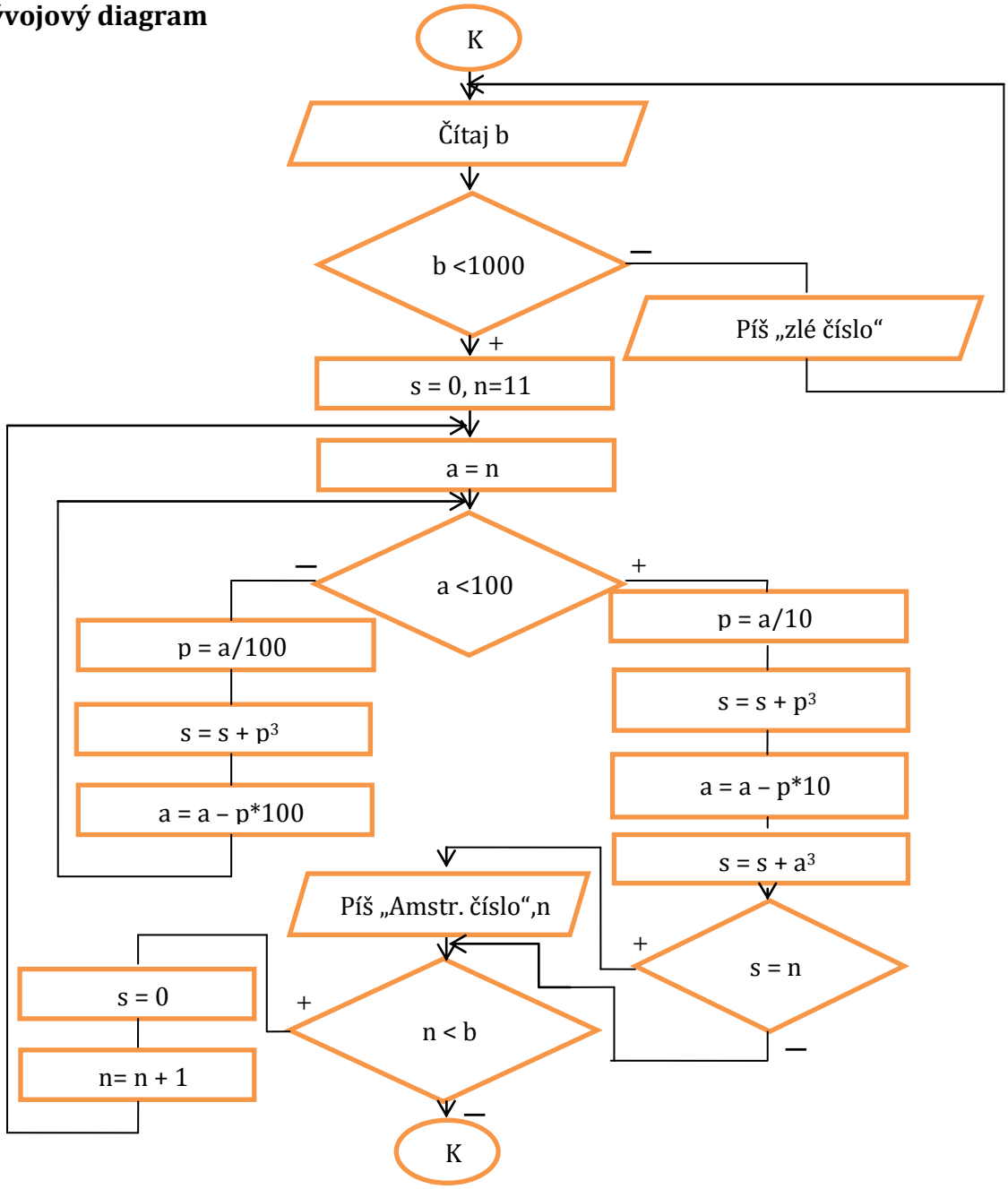

1
n >= 100
+
s =s + n/100
–
n = n – s*100
n >= 50
+
p1 =p1 + n/50
–
n = n – p1*50
n >= 20
+
d1 =d1 + n/20
–
n = n – d1*20
n >= 10
+
d2 =d2 + n/10
–
n = n – d2*10
n >= 5
+
p2 =p2 + n/5
–
n = n – p2*5
n >= 2
–
+
d3 =d3 + n/2
n = n – d3*2
j = n
Píš „počet“ p, d, s, p1, d1, d2, p2, d3, j
K

Úloha 23

Návrh algoritmu pre vyhľadanie Armstrongových čísel menších ako 1000.

Rozbor problému:
Armstrongovo číslo je také v ktorom súčet tretích mocnín jednotlivých jeho cifier je rovný samotnému číslu. Takým číslom je napr. číslo 153. Teda platí $153=1^3+5^3+3^3$. Z uvedeného vyplýva, že ak na vstupe zadáme číslo *b* algoritmus musí previesť kontrolu všetkých čísel z intervalu *<a,b>*, či súčet tretích mocnín jednotlivých číslic sa rovná týmto číslam. Najskôr je potrebné v každom čísle zistiť jeho číslice a potom vypočítať súčet ich tretích mocnín. Bez výpočtu vieme povedať, že také číslo určite nebude jednociferné. Preto stačí nastaviť v intervale *<a,b>* hodnotu *a>10*.
Vstup: *b* – prirodzené číslo, *b<1000*.
Výstup: *c*–prirodzené číslo, súčet tretích mocnín jeho číslic sa rovná číslu c
10<c<1000.

Vývojový diagram

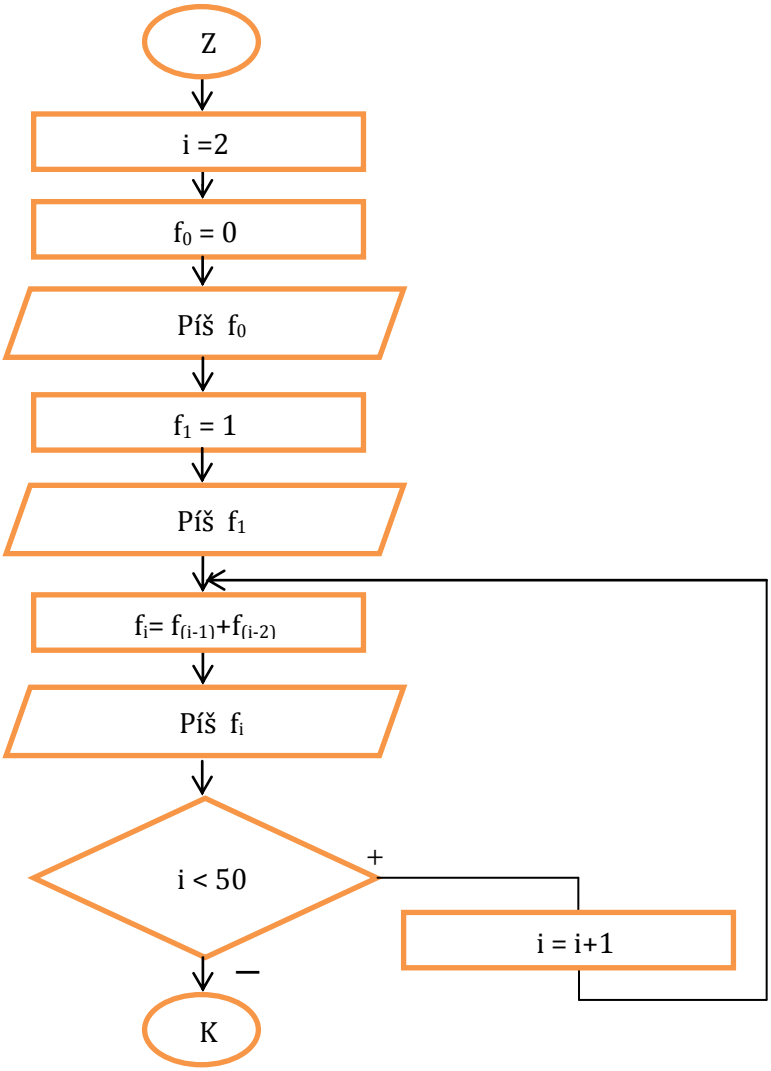


Úloha 24

Návrh algoritmu pre výpočet prvých 50 prvkov Fibonacciho postupnosti.

Rozbor problému:
Pri Fibonacciho postupnosti je každé číslo počítané ako súčet dvoch predchádzajúcich: 0,1, 1, 2, 3, 5, 8, 13, ..., teda platí $f_0=0$, $f_1=1$,$f_2=1$ ....., $f_i=f_{i-1}+f_{i-2}$. Deklarujeme si jednorozmerné pole *f* pričom do prvých prvkov poľa priradíme hodnoty $f_0=0$, $f_1=1$.Každý nasledujúci prvok potom vypočítame podľa vzťahu fi=fi-1+fi-2 pričom i=2,3,.....50.
Vstup: f0=0, f1=1 –prvé dva členy postupnosti
Výstup: postupnosť členov f0=0, f1=1, f2=1 ....., fi=fi-1+fi-2

Vývojový diagram

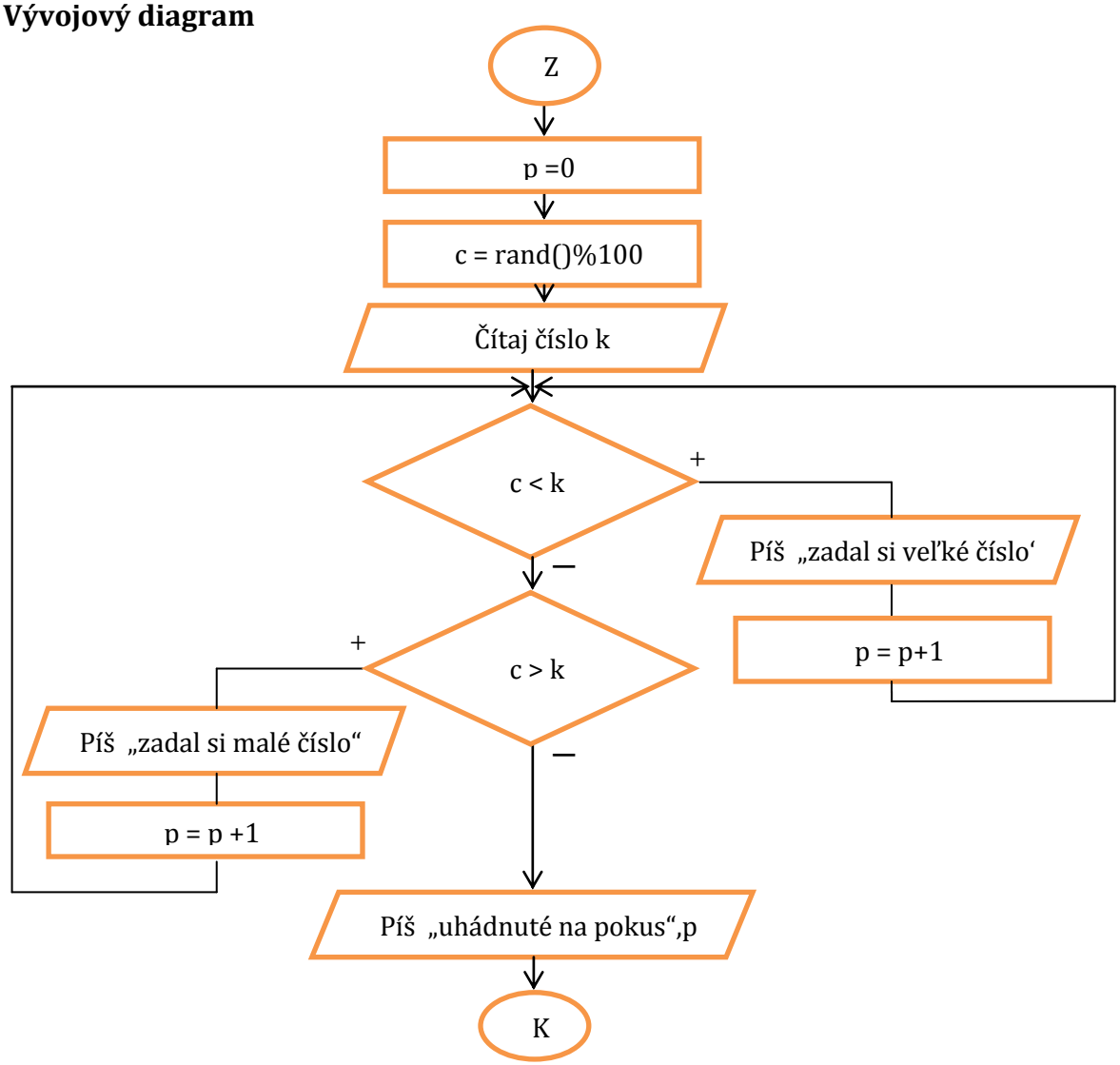


Úloha 25

Návrh algoritmu pre hádanie náhodne vygenerovaného čísla *c<100* a výpis počtu pokusov hádania.

Rozbor problému:
Algoritmus by mal simulovať hru „mysli si číslo“. Počítač vygeneruje číslo z intervalu <0,100>, uloží do premennej *c*.V cykle s podmienkou na konci zadáme číslo do premennej k. Toto číslo sa porovnáva s vygenerovaným číslom a podľa výsledku porovnania sa vypíše oznam či je myslené číslo menšie alebo väčšie. V prípade ak sa rovnajú vypíše sa oznam “číslo uhádnuté na pokus p”. Premenná p má na začiatku algoritmu hodnotu 0 a po každom neúspešnom pokuse sa zvýši o hodnotu 1.
Vstup: *c* – vygenerované celé kladné číslo, *c<100*
Výstup: p –prirodzené číslo, počet pokusov hádania čísla *c*

Vývojový diagram



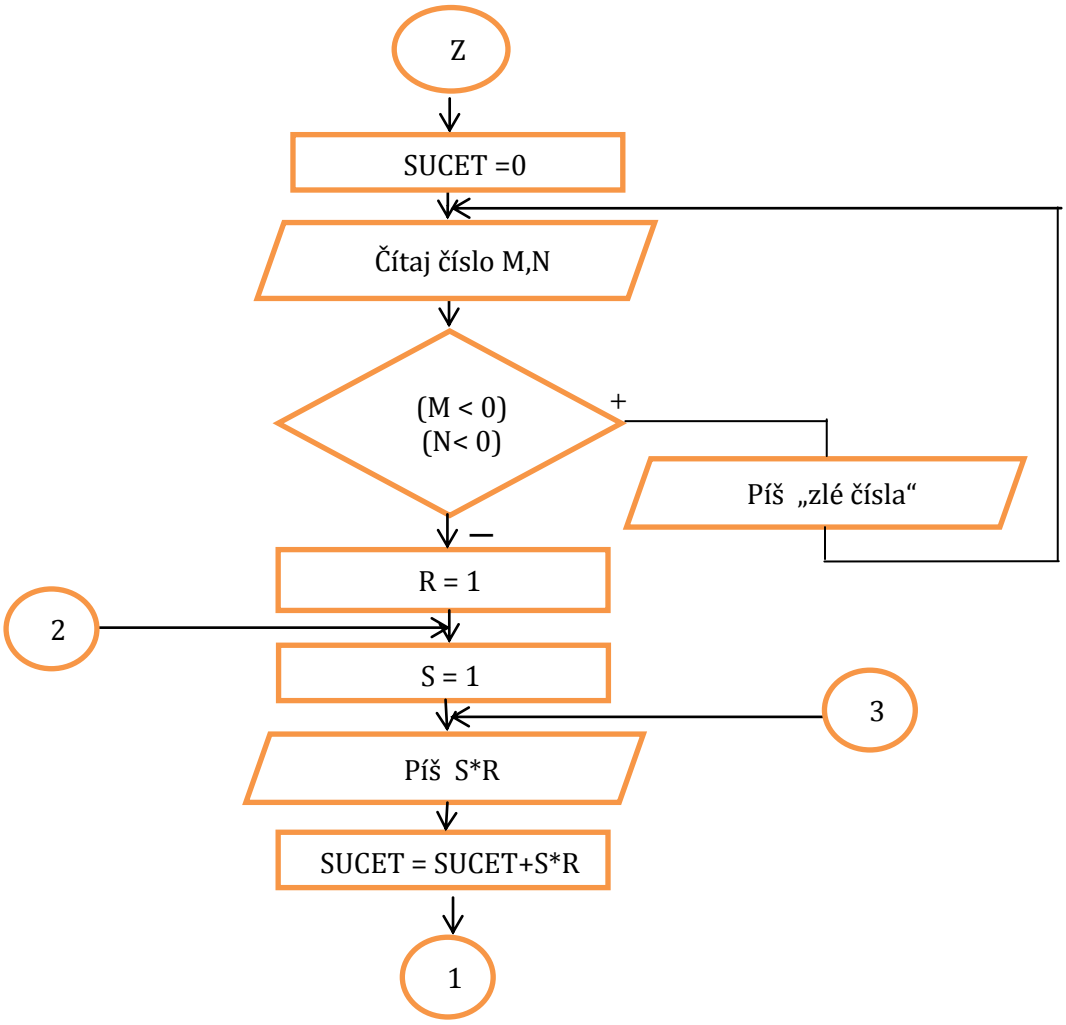
Úloha 26

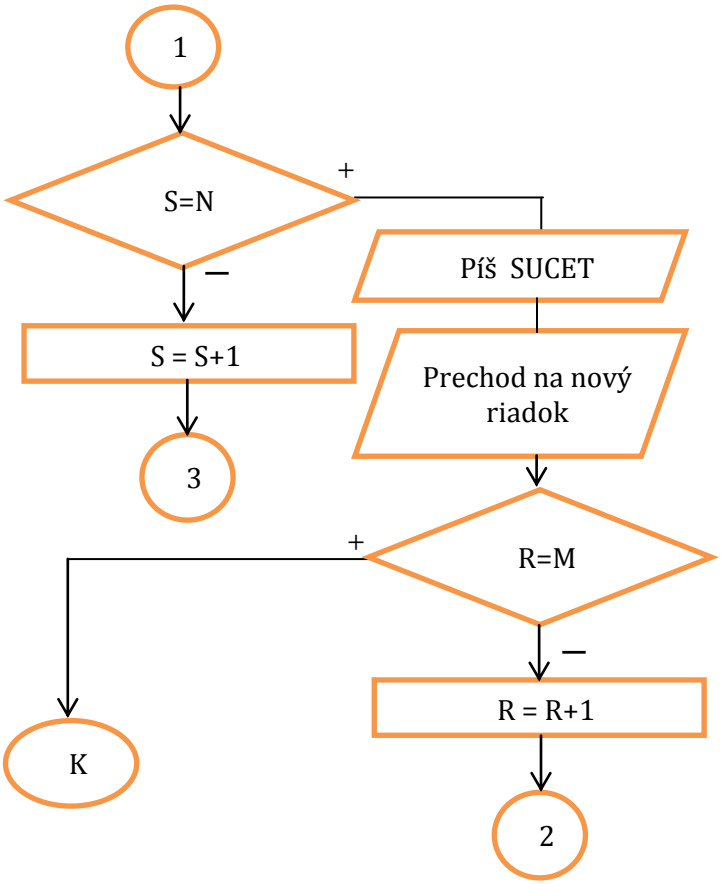
Návrh algoritmu pre výpočet a výpis tabuľky (matice) celých čísel. Tabuľka má M riadkov a N stĺpcov. Pre jednotlivé riadky tabuľky platí, že prvý riadok tvoria čísla 1,2,3,.....N a riadok K je zároveň K násobok prvého. Posledný N+1 stĺpec tabuľky bude súčtom príslušného riadku.

Rozbor problému:
Pre vytvorenie tabuľky je potrebné zadať dve celé kladné čísla M a N, ktoré vyjadrujú počet riadkov a stĺpcov tabuľky. Úlohu môžeme riešiť použitím dvoch cyklov so známym počtom opakovaní. Vonkajší cyklus bude určovať riadok tabuľky (deklarujeme riadiacu premennú cyklu R). Vnútorný cyklus bude pre konkrétne R určovať stĺpec tabuľky (premenná S). Je potrebné deklarovať ešte pomocnú premennú SUCET, ktorá pri prechode do nového riadku a na začiatku algoritmu nadobudne hodnotu 0. Pri výpočte prvkov konkrétneho riadku tieto budeme pripočítavať do premennej SUCET a vypíšeme ako posledný prvok N+1 v riadku.

Vstup: *M*-celé kladné číslo- počet riadkov tabuľky
N- celé kladné číslo- počet stĺpcov tabuľky
Výstup: vypísaná tabuľka

Vývojový diagram





Úloha 27

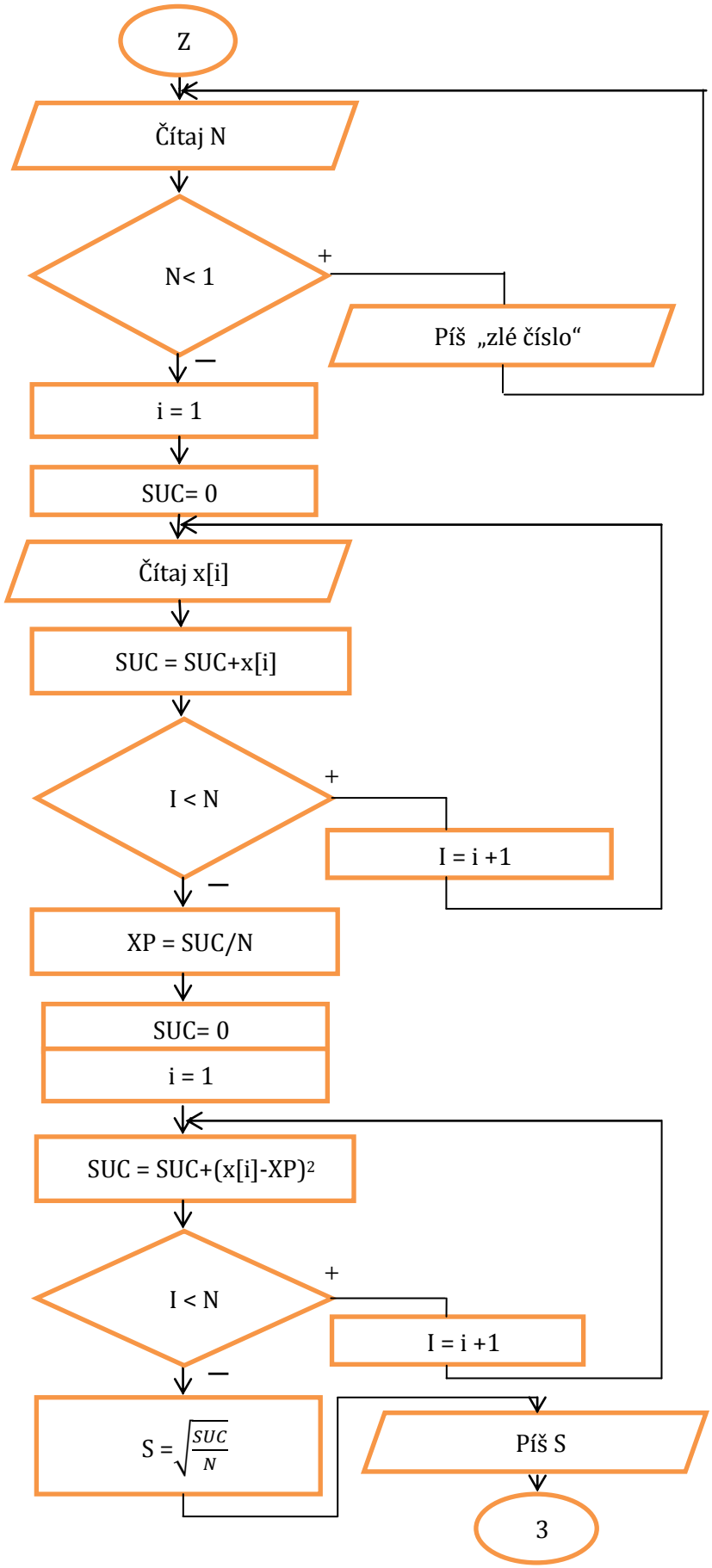
Návrh algoritmu pre výpočet smerodajnej odchýlky v štatistickom súbore s N prvkami.

Rozbor problému:
Smerodajná odchýlka vyjadruje v teórii pravdepodobnosti a štatistike strednú kvadratickú odchýlku jednotlivých hodnôt štatistického súboru. K výpočtu teda potrebujeme vypočítať najskôr priemernú hodnotu zo štatistického súboru a následne smerodajnú odchýlku. Smerodajná odchýlka je vlastne druhá odmocnina rozptylu. Vzorec pre výpočet smerodajnej odchýlky je : S= $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\bar{x})^2}$. Z uvedeného vzťahu vyplývajú aj vstupné požiadavky. Hodnota *N* musí byť rôzna od nuly. Hodnota *N* musí byť teda celé kladné číslo. Aby štatistické zisťovanie malo praktickú interpretáciu hodnota *N* musí byť dostatočne vysoká. V algoritme budeme kontrolovať iba skutočnosť, či je táto hodnota kladná. Pri načítaní hodnôt štatistického súboru zároveň v jednom cykle môžeme vypočítať súčet hodnôt, ktorý použijeme na výpočet priemeru. V druhom cykle vypočítame hodnotu smerodajnej odchýlky.

Vstup: *N*-celé kladné číslo- počet hodnôt v štatistickom súbore

Výstup: S= $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\bar{x})^2}$- smerodajná odchýlka.

Vývojový diagram



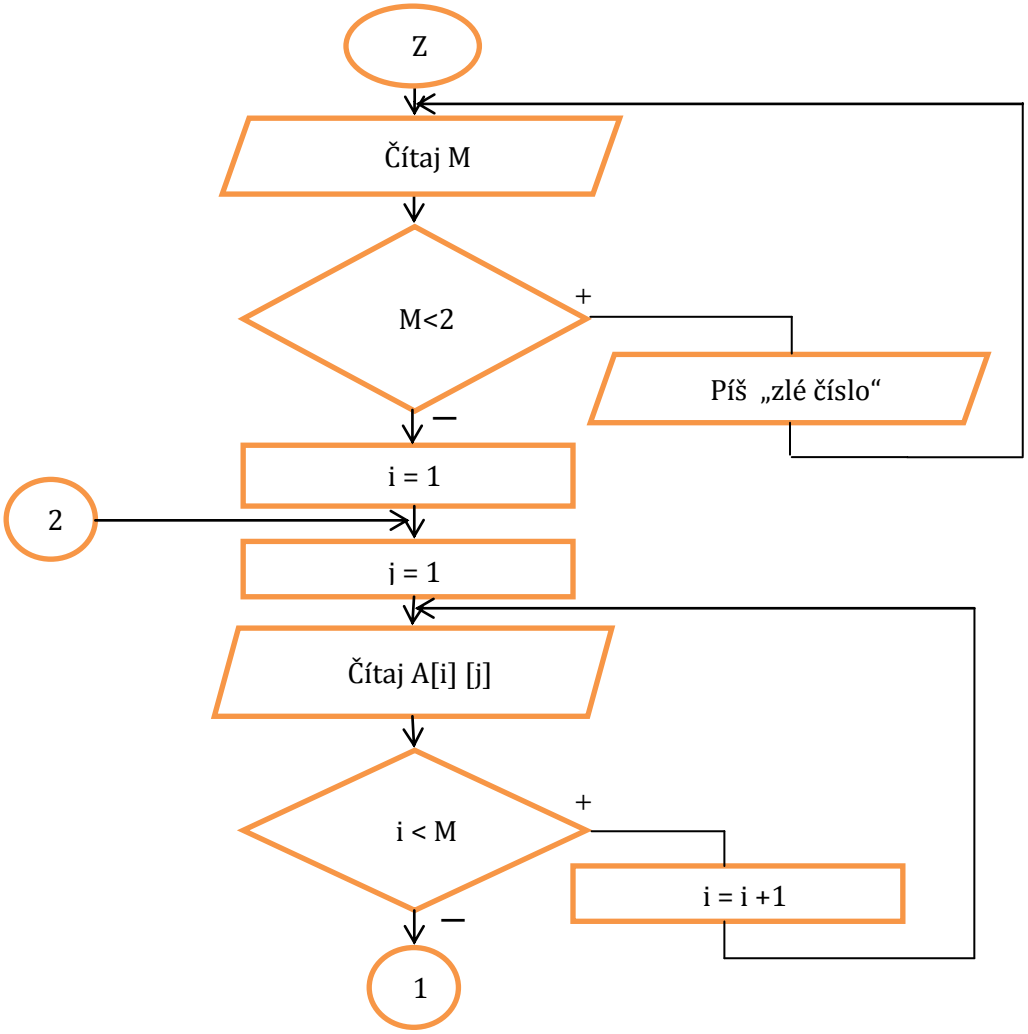
Úloha 28

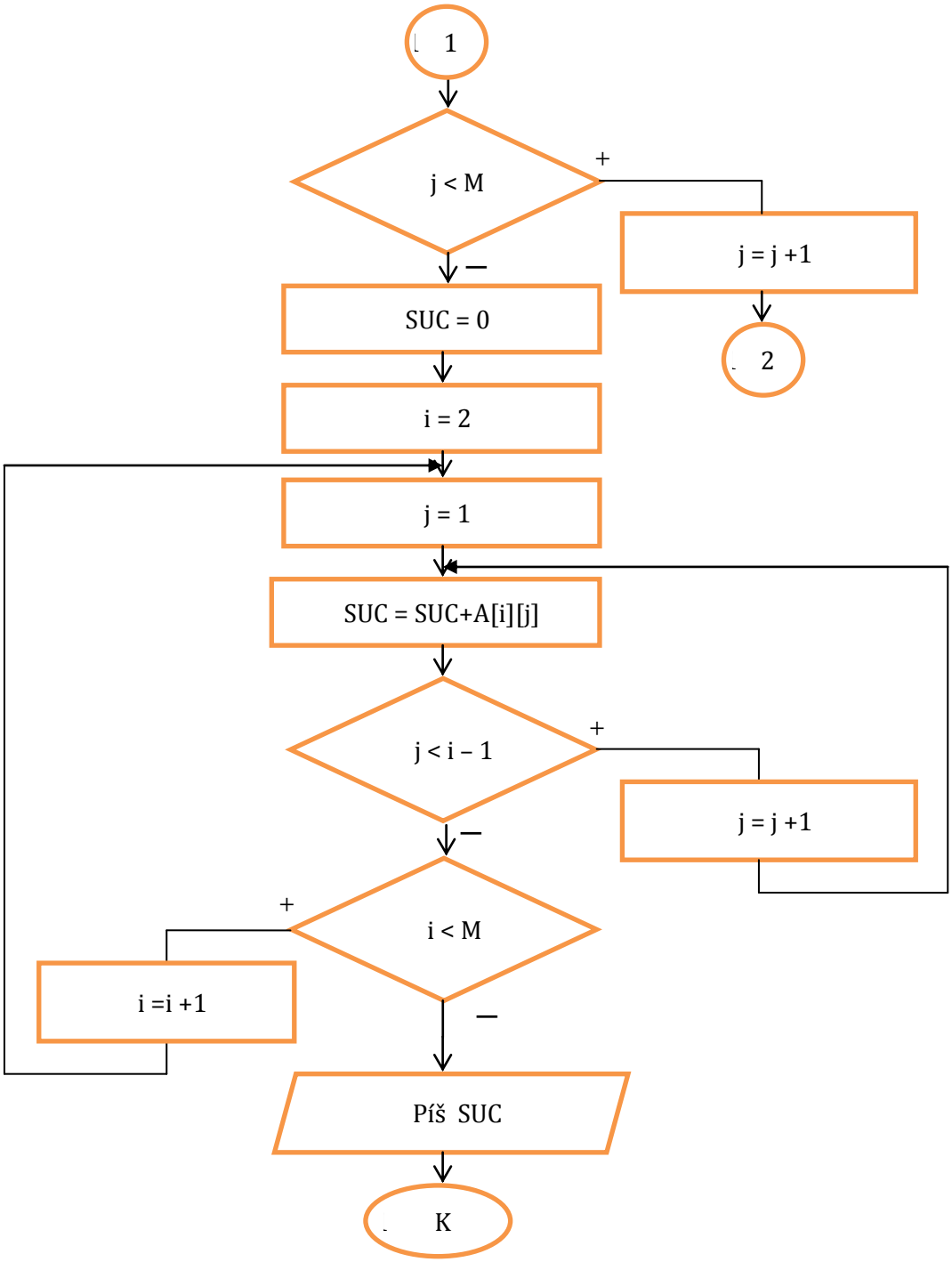
Návrh algoritmu pre načítanie tabuľky (štvorcovej matice) s rozmerom M riadkov a M stĺpcov a výpočet súčtu z hodnôt prvkov tabuľky nachádzajúcich sa pod hlavnou diagonálou.

Rozbor problému:
Ak si označíme jednotlivé prvky tabuľky premennou $A_{i,j}$, kde *i* označuje riadok a *j* stĺpec tabuľky, potom prvky $A_{i,i}$ pre i=1,2,3……M, sú prvky na hlavnej diagonále. Prvky pod hlavnou diagonálou budú teda tie pre ktoré platí, že sa nachádzajú v riadku *2,3…..M* a stĺpci *1,2,….(M-1)*. Na výpočet súčtu prvkov pod diagonálou použijeme v algoritme dva cykly. Riadiaca premenná vonkajšieho cyklu *i* bude predstavovať riadok tabuľky. Preto bude nadobúdať hodnoty *2,3….M*. Riadiaca premenná vnútorného cyklu *j* bude predstavovať stĺpec tabuľka a jej hodnota v príslušnom riadku *i* bude *1,2,….(i-1).* Na začiatku algoritmu je ešte potrebné deklarovať premennú *SUC* s počiatočnou nulovou hodnotou, do ktorej sa bude ukladať súčet prvkov pod diagonálou a zadať hodnotu *M* – počet riadkov a zároveň stĺpcov tabuľky. Pre číslo *M* musí platiť *M>1*.

Vstup : *M* – celé kladné číslo , počet riadkov a stĺpcov tabuľky, *M>1*
$A_{i,j}$ – prvky tabuľky
Výstup: *SUC* – súčet prvkov tabuľky nachádzajúcich sa pod hlavnou diagonálou.

Vývojový diagram




1
j < M
+
j = j +1
2
SUC = 0
i = 2
j = 1
SUC = SUC+A[i][j]
j < i – 1
+
j = j +1
i < M
+
i =i +1
Píš SUC
K

ZÁVER

V predkladanej overenej pedagogickej skúsenosti sme poukázali na jednu z možností ako formovať, rozvíjať algoritmické myslenie našich žiakov. Je to možné vtedy ak pri riešení akéhokoľvek problému prísne dodržujeme zásady tvorby správneho algoritmu. Je predpoklad, že žiaci po získaní takýchto návykov dokážu mať komplexný, analytický pohľad na riešenie úloh, s ktorými sa budú v živote stretávať. To bolo aj jedným z cieľov práce.

V jednotlivých úlohách práce sme sa snažili v rozbore problému naznačiť myšlienkový postup pri riešení danej úlohy. Tento samozrejme treba najmä pri zložitejších úlohách považovať za určitý návrh , návod a každý učiteľ, alebo žiak ho môže obohatiť nájdením vhodnejšieho, efektívnejšieho postupu. Často žiaci pri písaní programov v nejakom konkrétnom programovacom jazyku zanedbávajú overený postup jednotlivých etáp tvorby programov. Dostatočný priestor nevenujú práve analýze úlohy. Tak sa stáva, že hotový odladený program zdanlivo funguje správne. Ak nastane situácia, ktorú z dôvodu nedostatočne prevedenej analýzy programátor nezohľadnil, program havaruje, alebo dáva nesprávne výsledky.

Naša skúsenosť z vyučovacích hodín informatiky, najmä tematického celku „Postupy, riešenie problémov, algoritmické myslenie“ nás priviedla k presvedčeniu, že pri návrhu algoritmov je učiteľ iba koordinátor, ktorý usmerňuje žiakov. Samotný návrh musia urobiť žiaci. Učiteľ ukáže žiakom určité overené postupy, zásady ktoré ovplyvňujú myslenie žiaka pri tvorbe algoritmu. Učiteľ však nemôže žiaka prinútiť aby prijal jeho myslenie. Preto aj hodiny informatiky sú pre učiteľa náročné. Musí pochopiť myslenie žiaka pri tvorbe algoritmu, ktoré môže byť jedinečné a pritom správne. Určitý algoritmus je možné prijať, osvojiť si ho, ale nie sa ho naučiť naspamäť. To by neviedlo k rozvíjaniu algoritmického myslenia žiakov.

Túto prácu odporúčame ako návrh takýchto postupov, zásad pri tvorbe algoritmov. Zároveň aj ako pomôcku pre učiteľov, ktorá by mala zefektívniť a uľahčiť ich prípravu na vyučovaciu hodinu informatiky.

ZOZNAM BIBLIOGRAFICKÝCH ZDROJOV

1. GVOZDIAK L., HANULOVÁ Ľ., JANKOVIČOVÁ A., MOLNÁR Ľ. 1977. Programovanie pre číslicové počítače. 2. vydanie. Alfa, Bratislava 1977. ISBN: 63-553-77

2. KERNIGHAN W., 1988. Programovací jazyk C. The C ProgramingLanguage. Alfa, Bratislava. 1988 ISBN: 063-075-88 PJC

3. http://www.statpedu.sk/files/documents/svp/gymnazia/vzdelavacie_oblasti/informatika_isced3a.pdf