Algoritmus, problém, program

Problém

- je stav, v ktorom jestvuje rozdiel medzi tým, čo v danom momente máme a tým, čo chceme dosiahnuť.

Riešenie problému chápeme ako odstraňovanie rozdielu medzi aktuálnym stavom a tým,čo chceme dosiahnuť.

Algoritmus

- je postup, pomocou ktorého môžeme vyriešiť daný problém (napr. kuchársky recept, PC program...)
- je postupnosť príkazov (operácií), ktorá rieši daný problém. Na výbere programovacieho jazyka záleží, aké príkazy (operácie) máme k dispozícii.
- je postup, ktorého realizáciou získame zo zadaných vstupných údajov po konečnom počte činností v konečnom čase správne výsledky

Vývojový diagram

- grafické znázornenie algoritmu pomocou normatívnych značiek...

Program

- algoritmus napísaný v programovacom jazyku

Tvorbou programov sa zaoberá **softvérové inžinierstvo**. Zaoberá špecifikovaním, návrhom, vývojom a údržbou softvéru s využitím poznatkov informatiky, projektového manažmentu a ďalších oblastí. Vývoj softvéru prebieha v niekoľkých etapách.

Základné etapy tvorby programov:

1. **Rozbor problému**
   - spočíva v **presných** odpovediach,**čo** treba riešiť. Pozorne sformulovať zadanie problému a požiadavky na vznikajúci program.
   - **Cieľom rozboru** problému je úplne a jednoznačne obsiahnuť riešený problém a všetky podstatné podmienky prostredia a preveriť uskutočniteľnosť vývoja plánovaného programu.

     Pozostáva z:
     - rozdelenie problému na menšie podproblémy a tie na elementárne problémy
     - špecifikácie vstupných dát (štruktúra, formát, množina prípustných hodnôt)
     - špecifikácie výstupných dát (štruktúra, formát, funkčná závislosť od vstupných dát)
     - špecifikácie výnimočných situácií (napr. reakcia na chybné vstupné dáta)

2. **Návrh riešenia (vytvorenie algoritmu)** – **ako** daný problém riešiť; výsledkom je algoritmus (postupnosť príkazov vedúcich k splneniu úlohy)

   - Je to **jednoznačný zrozumiteľný postup** umožňujúci z daných vstupných údajov získať v konečnom čase po vykonaní konečného počtu krokov správne výsledky - riešenie danej úlohy.
   - Je to presne definovaná, **konečná postupnosť príkazov**, ktorých vykonanie umožní po konečnom počte krokov získať pre prípustné vstupné údaje zodpovedajúce výstupné údaje.
   - rozumieme ho ako **predpis na spracovanie**, ktorý je formulovaný natoľko presne, že ho môže realizovať mechanické alebo elektronické zariadenie.

- Príkladmi sú predpisy na sčítanie, odčítanie, násobenie a delenie čísel, Euklidov algoritmus na výpočet najväčšieho spoločného deliteľa. Algoritmy tiež pripomínajú kuchárske recepty, návody pre domácich majstrov a podobne. Tieto však nie sú definované natoľko presne, aby sa dali realizovať pomocou stroja.

3. **Realizácia (implementácia) vytvorenie programu** – proces prepísania navrhnutého algoritmu do programovacieho jazyka (aj príprava obrázkov, zvukových efektov...)

   - Program sa zostavuje pomocou exaktne definovanej **sady príkazov operátorov a syntaxe,** ktorú je možné previesť do tvaru vykonateľného počítačom.
   - Pri návrhu komplexnejšieho programu je potrebné **oddeliť používateľské rozhranie** od počítača. Preto celý realizovaný program rozdelí na nezávisle realizovateľné jednotky - moduly.

     To umožňuje:
     - rozdeliť úlohy medzi viacerých programátorov
     - opätovne využiť hotové moduly
     - sprehľadniť program
     - odbremeniť ostatných programátorov skrytím podrobností, čím sa urýchli vývoj
     - uľahčiť hľadanie chyby a zjednodušiť údržbu
     - nahradiť modul iným modulom s tou istou funkčnosťou
     - doplniť program o ďalší modul

     Pri tvorbe modulov platia tieto zásady:
     - Udrž to malé - Modul by mal obsahovať čo najmenej funkcií.
     - Udrž to prehľadné - Modul by mal obsahovať čo najmenej funkcií z cudzích modulov a mal by obsahovať komentár k jednotivým príkazom
     - Udrž to na jeden účel - V module by sa mali nachádzať iba funkcie ktoré slúžia na jeden účel.

4. **Údržba (testovanie a ladenie)** – softvér sa začne používať; odhaľovanie a oprava skrytých chýb; optimalizácia (optimalizácia kódu); prispôsobenie sa meniacim sa požiadavkám používateľov, vývoj novších verzií

Optimalizovanie kódu: znamená, že sa pokúšame dosiahnuť, aby sa program vykonával rýchlejšie, aby spotreboval menej pamäte, alebo aby jeho chod bol plynulý.

Vyladiť môžeme program pomocou:
- krokovaním – spúšťanie programu po riadkoch, príkazoch
- kontrolou obsahu premenných
- zastavením programu na zvolenom mieste

Logické chyby:
- zistené počas alebo po skončení behu programu:
  vedúce k predčasnému zastaveniu programu s chybovou správou (run time errors) napr.: delenie nulou, súbor na otvorenie nenájdený, výpočet väčšieho čísla, aké sme si zadefinovali a pod.
- vedúce **k chybným výsledkom**,
  zacykleniu programu (zastavenie vykonávania programu: Run - Program Reset) a pod.

Vlastnosti algoritmu

1. Elementárnosť - *postup je zložený z činností, ktoré sú pre realizátora elementárne, zrozumiteľné.*

Napr. deti na 1. stupni ZŠ už vedia násobiť, zistiť výsledok súčinu 2.2.2.2.2.2 je pre nich elementárnou operáciou, ale vypočítať 2^6 už nevedia.

Iný príklad - pre niekoho je elementárnou činnosťou návod v recepte - "pridaj štipku soli, dochuť podľa chuti", pre iného je to problémom.

Človek má jednu výhodu- dokáže sa učiť a zvláda stále dokonalejšie elementárne činnosti, ktoré kombinuje do ešte zložitejších postupov. Túto vlastnosť však nemajú počítače, majú iba obmedzenú sadu elementárnych činností z oblasti spracovania informácií.

2.Determinovanosť - *postup je zostavený tak, že je v každom momente jeho vykonávania jednoznačne určené, aká činnosť má nasledovať, alebo či sa už postup skončil.*

Ľudia podvedome dokážu chápať postupy, v ktorých nie je zvýraznené poradie vykonávaných činností, prípadne, čo a ako dlho opakovať. Pre nemysliace zariadenie je však uvedenie poradia nevyhnutnosťou.

Napr. človek pochopí čo má urobiť, keď mu povieme: "Skôr ako prejdeš cez cestu, pozri sa či nejde auto". Počítaču by sme to museli rozpísať do krokov:

1. Pozri doľava
2. Ak ide auto opakuj krok 1.
3. Prejdi do polovice ulice.
4. Pozri doprava.
5. Ak ide auto opakuj krok 4.
6. Prejdi na druhú stranu ulice.

3. Rezultatívnosť - *postup dáva pre rovnaké vstupné údaje vždy rovnaké výsledky (ak skončí).*

Opäť v bežnom živote nič nezvyčajné. Napr. Dve kuchárky podľa toho istého receptu nenapečú nikdy to isté.

4. Konečnosť - *postup skončí vždy v konečnom čase a po vykonaní konečného počtu činností.*

Skúsenosti nám umožňujú prerušiť nejaký postup v momente, kedy vidíme, že nevedie k požadovanému výsledku. Skúsenosti však od nemysliaceho zariadenia nevyžadujeme, preto musíme formulovať postup riešenia tak, aby určite skončil.

Algoritmus by nemal vyzerať napr. takto:

1. Mysli si číslo.
2. Pokiaľ sa nebude rovnať 1, odčítaj od neho 2.

V prípade zadania párneho, desatinného alebo záporného čísla bude počítač odčítavať 2 donekonečna.

Niektoré algoritmy vyzerajú teoreticky konečné, ale prakticky nerealizovateľné. Napr. máme zistiť počet zrniek piesku na pláži. Môžeme to urobiť tak, že spočítame približný počet zrniek na lopate, prehádžeme celú pláž lopatou a zapamätáme si počet lopát.

V takomto prípade problém treba zjednodušiť a riešiť ho len približne, s určitou chybou.

5. Hromadnosť - *postup je aplikovaný na celú triedu prípustných vstupných údajov.*

Napr. v bežnom živote je cieľom naučiť deti ako sa násobia ľubovoľné dve čísla, nie koľko je 2x2.

Napriek tomu sa dnes stretávame s jednoúčelovými postupmi, ktoré nemajú premenlivé vstupné údaje napr. programy pre automat. práčku a pritom sú algoritmami. Hromadnosť postupu je skrytá v tom, že postup pripúšťa premenlivé vstupné údaje a umožňuje nám riešiť úlohy podobného typu.

6. Efektívnosť - *postup sa uskutočňuje v čo najkratšom čase a s využitím čo najmenšieho počtu prostriedkov.*

Často býva cieľom zostaviť hocijaký, ale funkčný algoritmus, potom ho v prípade potreby a s lepšou znalosťou problému vylepšovať. Ako kritéria efektívnosti slúžia časová a pamäťová zložitosť algoritmu.

Napr. máme spočítať prvých 50 prirodzených čísel. Môžene sčitovať postupne 1+2+3+..., alebo sčitovať dvojice 1+50=51, 2+49=51, 3+48=51, ...,25+26=51 a takýchto dvojíc je polovica z počtu čísel, teda 25. Vynásobením 25x51=1275 dostaneme hľadaný výsledok. Je jasné, že druhý postup je efektívnejší a možno ho aj zovšeobecniť pre ľubovoľný počet prvých prirodzených čísel N. Vzorec je SÚČET = N/2 x (N+1).

- Algoritmus nazývame *čiastočne správny*, ak v prípade, že skončí, dáva vždy správne výsledky.
- Algoritmus nazývame *konečný*, ak skončí v konečnom čase pre ľubovoľné vstupné údaje.
- Algoritmus nazývame ***správny***, ak je čiastočne správny a konečný.

Zápis algoritmu

Existuje niekoľko spôsobov, nie všetky sú vhodné na spracúvanie počítačom. Slúžia skôr na to, aby sme algoritmus vedeli **ľahšie navrhnúť**.

Príklad algoritmu popísaného prirodzeným jazykom možno uviesť ktorýkoľvek **recept z kuchárskej knihy.** Nie je v ňom výnimkou takýto popis činností:

… cesto položíme na vymastený a múkou posypaný plech a ihneď ho natrieme vopred pripravenou plnkou...

Autor receptu predpokladal, že ho použije skúsená kuchárka. Tá dobre vie, že skôr než cesto položí na plech, musí ho vymastiť a posypať múkou. Vtedy už má plnku pripravenú, pretože ju bude potrebovať, len čo cesto položí na plech.

V recepte – algoritme však autor všetky tieto aktivity uviedol až po „príkaze“ položiť cesto na plech. Algoritmus v prirodzenom jazyku môže byť naozaj neformálny. Predpokladá sa že človek si je schopný niektoré veci domyslieť, a preto niekedy obsahuje aj drobné nepresnosti (niekedy dokonca nejednoznačnosti).

V zápise algoritmu, ktorý je určený pre počítač, nie sú nepresnosti a nejednoznačnosti prípustné.

1.Prirodzeným jazykom

Prvým krokom na upresnenie zápisu algoritmu je jeho zápis v takom jazyku, ktorý obsahuje len niekoľko vetných konštrukcií **prirodzeného jazyka**. Tieto konštrukcie vyjadrujú aktivity vykonané pri spracúvaní algoritmu.

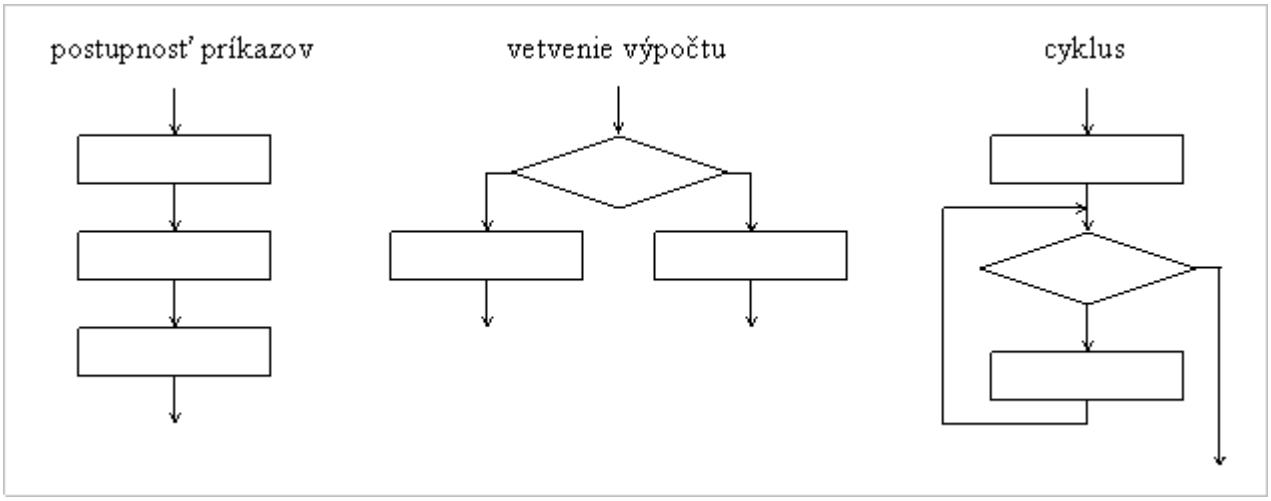
Iným spôsobom vyjadrenia algoritmu je **algoritmický jazyk**. Často je to akási zmes **prirodzeného jazyka** a **príkazov nejakého konkrétneho programovacieho jazyka**, do ktorého plánujeme algoritmus prepísať. Od zápisu algoritmu v algoritmickom jazyku k jeho zápisu v programovacom jazyku je už len malý krôčik. Často stačí nahradiť niektoré slová a zápisy skutočnými **príkazmi programovacieho jazyka**.

Napr. program na hľadanie lacnejšej z dvoch chát zapísaný v algoritmickom jazyku by mohol vyzerať takto:

píš „ Zadaj cenu stravy a ubytovania na prvej chate“
prečítaj Cena1
píš „Zadaj cenu dopravy na prvú chatu“
prečítaj Doprava1
píš „Zadaj počet dní trvania výletu“
prečítaj PočetDní
Chata1 ← Cena1 * PočetDní
Náklady1 ← Chata1 + Doprava1
píš „Zadaj cenu stravy a ubytovania na druhej chate“
prečítaj Cena2
píš „Zadaj cenu dopravy na druhú chatu“
prečítaj Doprava2
Chata2 ← Cena2 * PočetDní
Náklady2 ← Chata2 + Doprava2
ak Náklady1 < Náklady2 **tak píš** „Chata 1 je lacnejšia.“
ak Náklady1 > Náklady2 **tak píš** „Chata 2 je lacnejšia.“
ak Náklady1 = Náklady2 **tak píš** „Obe chaty sú rovnako drahé.“

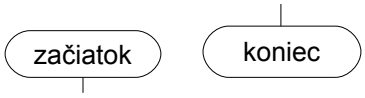
2. Grafickým zápisom.

Vývojový diagram znázorňuje rôzne typy príkazov rôznymi tvarmi blokov:

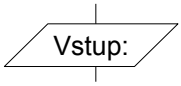


dohodnuté značky:

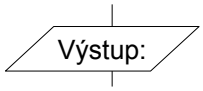
začiatok a koniec programu:



vstup údajov:

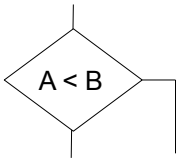


výstup údajov:

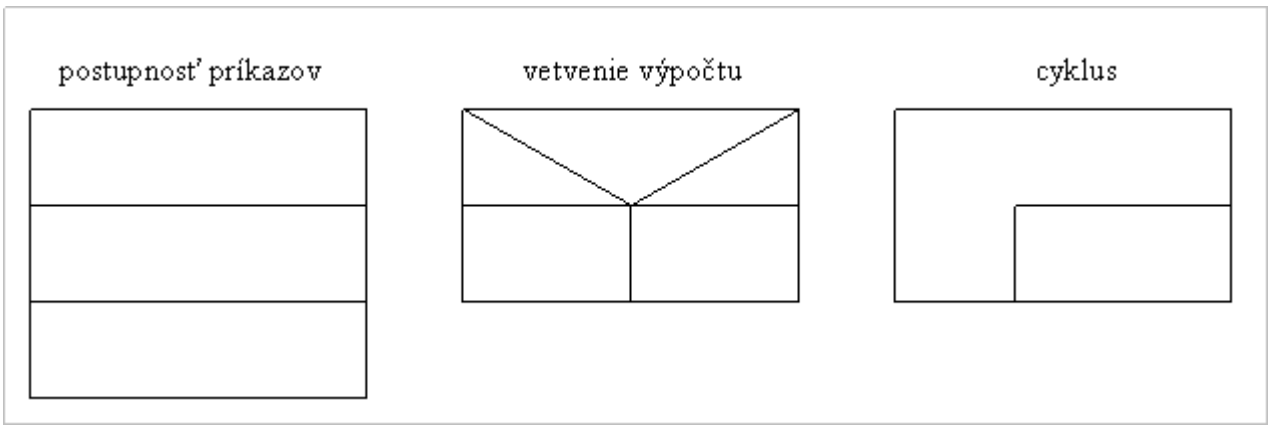


jednoduchý príkaz:

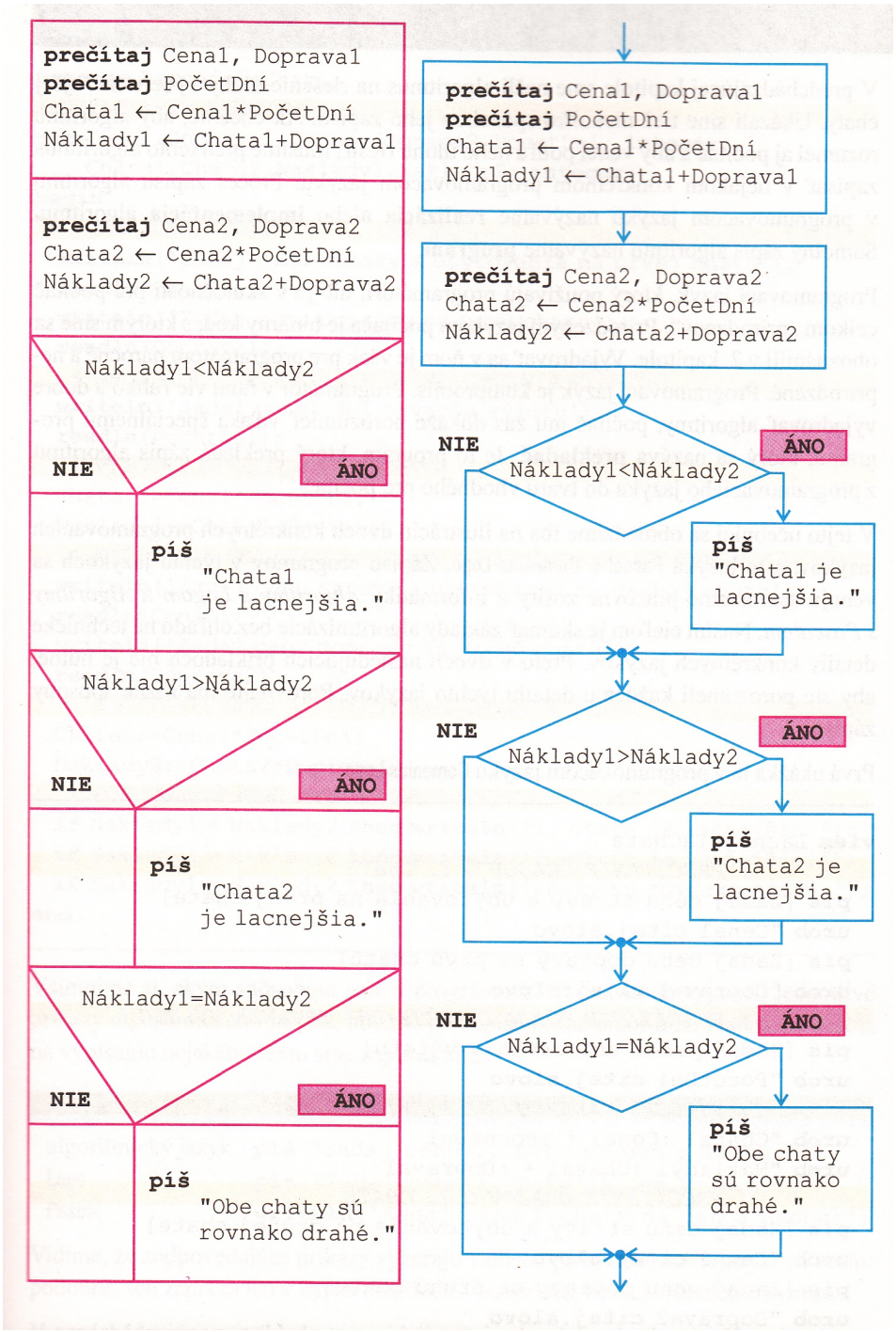
podmienený príkaz (rozhodnutie):



Vývojové diagramy boli v minulosti veľmi populárne. Aj dnes sa v jednoduchých prípadoch používajú na znázornenie výpočtov alebo akcií v rôznych oblastiach ľudskej činnosti. **Súčasné programovacie jazyky** sú však založené na konštrukciách, ktoré vývojové diagramy nemajú. Preto dávame prednosť grafickému vyjadreniu algoritmu pomocou **štruktúrogramov**. Tieto sú na prvý pohľad možno komplikovanejšie ako vývojové diagramy. Na rozdiel od nich však lepšie vystihujú jednotlivé konštrukcie algoritmu.



Algoritmus o chatách zapísaný vývojovým diagramom a štruktúrogramom:

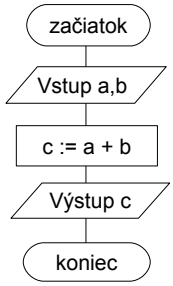


Príklady niektorých jednoduchých algoritmov:

Algoritmus na sčítanie dvoch čísel:

Vstup: ***a, b***
Výstup: ***c***

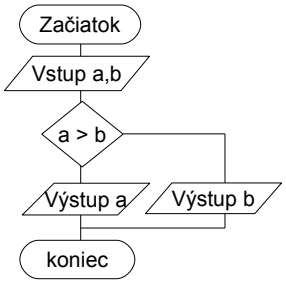
1. Začiatok
*2. Vstup **a, b***
*3. **c := a + b**(do premennej c vloží súčet premenných a + b)*
*4. Výstup **c***
5. Koniec



Algoritmus na zistenie väčšieho z dvoch rôznych čísel a, b

Vstup: ***a, b***
Výstup: ***a*** alebo ***b***

1. Začiatok
*2. Vstup **a, b***
*3. **ak a > b, tak** vypíš **a** a choď na bod 5.*
*4. Vypíš **b***
5. Koniec



Algoritmus zabíjanie klincov

Formulácia problému

Zabi klinec do dosky.

Analýza úlohy

Vstupné údaje: kladivo, klinec, doska
Výstupné údaje: klinec zabitý do dosky
Analýza: zabíjať tak dlouho, pokial nie je klinec zabitý až po hlavičku

Zostavenie algoritmu

Slovný popis:

1.Vezmi kladivo a klinec

2.Prilož klinec k doske

3.Buchni kladivom po hlavičke

4.Je hlavička klinca zabitá v doske?
+ANO - pokračuj bodom 5
- NIE - vráť sa na bod 3

5.Ukonči činnosť a odlož kladivo