Informatika

Algoritmizácia a vývojové diagramy

- **Problém**
    - stav, v ktorom existuje rozdiel medzi tým, čo v danom momente máme a tým, čo chceme dosiahnuť
- **Riešenie problému**
    - odstraňovanie rozdielu medzi aktualnym stavom a tým, čo chceme dosiahnuť
- **Algoritmus**
    - postup, ktorým sa pri riešení problému riadime

Algoritmus je návod (presná postupnosť krokov a inštrukcií) na vykonanie činnosti, ktorý nás od (meniteľných) vstupných údajov privedie v konečnom čase k výsledku.

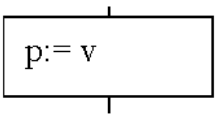
Vykonávanie činnosti na základe algoritmu označujeme ako ***výpočet.***

Vlastnosti algoritmu:

- elementárnosť
    - Postup je zložený z **jednoduchých** krokov, ktoré sú pre vykonávateľa (počítač, nemysliace zariadenie, človek) **zrozumiteľné**
- Determinovanosť
    - Postup je zostavený tak, že v každom momente jeho vykonávania je **jednoznačne určené, aká činnosť má nasledovať**, alebo či sa už postup skončil
- Rezultatívnosť
    - Realizácia dokázateľne **vedie po konečnom počte krokov k správnemu výsledku** pri riešení ľubovoľnej úlohy zo skupiny úloh, pre ktorú bol algoritmus vytvorený
- konečnosť
    - Algoritmus **musí mať vždy svoj koniec** (musí skončiť)
- Hromadnosť
    - Postup je **použiteľný na celú triedu** prípustných vstupných údajov
- Efektívnosť
    - Navrhnúť taký **postup, ktorý s použitím minimálnych prostriedkov v čo najkratšom čase vyrieši problém**

*Proces, ktorý vykonávame pri zápise algoritmu sa nazýva **algoritmizácia***

- Objekt (resp. miesto v pamäti) slúžiaci počas behu algoritmu na odkladanie údajov
- Jej hodnota sa počas činnosti algoritmu môže meniť (a zvyčajne sa aj mení)
- Nadobúda hodnoty priradením alebo načítaním
    - Načítanie je realizované prostredníctvom operácií pre vstup
    - Priraďovanie umiestní (priradí) do premennej konkrétnu hodnotu priraďovacím príkazom

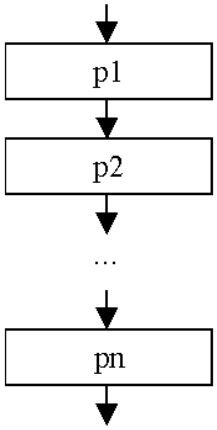


p je premenná
v je výraz, ktorého hodnotu priradením premenná *p* nadobudne

:= alebo **=** je symbol priradenia

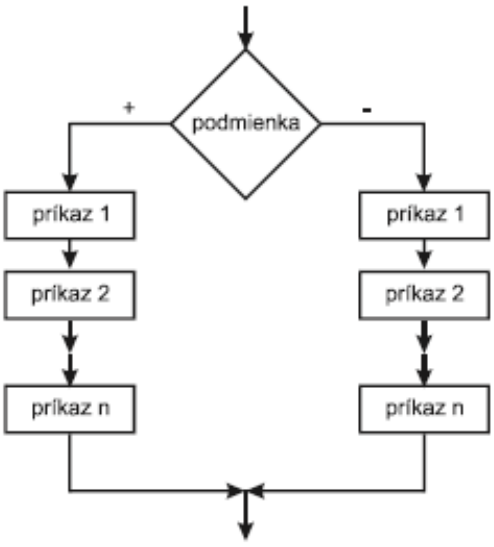
Sekvencia

- Postupnosť príkazov vykonávaná v takom poradí, v akom sú jednotlivé časti zapísané

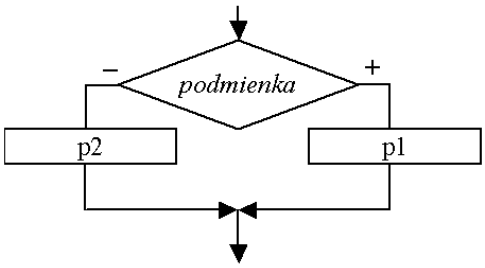


Vetvenie

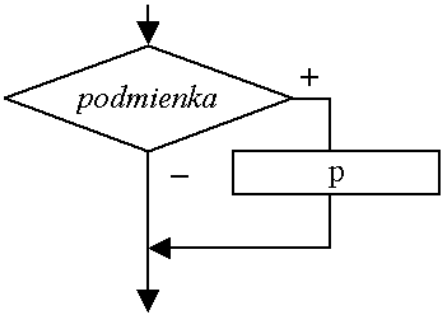
- Poskytuje **možnosť rozhodnúť sa** podľa pravdivosti skúmaného znaku.
- Skladá sa z **podmienky** uvedenej za slovíčkom ***ak*** a z príkazov, ktoré sa vykonajú v prípade kladného a záporného výsledku. Týmto dvom častiam hovoríme **vetvy**.
- Ak je podmienka splnená –> vetva „+“
- Ak nie je podmienka splnená –> vetva „-“



- **Úplné vetvenie**
  - obsahuje príkazy v oboch vetvách
  - ak je podmienka splnená vykoná sa príkaz p1
  - ak nie je splnená vykoná sa príkaz p2



- **Neúplné vetvenie**
  - Chýba jedna vetva (+ alebo -)
  - Ak je podmienka splnená vykoná sa príkaz p
  - Inak je vetvenie bez účinku

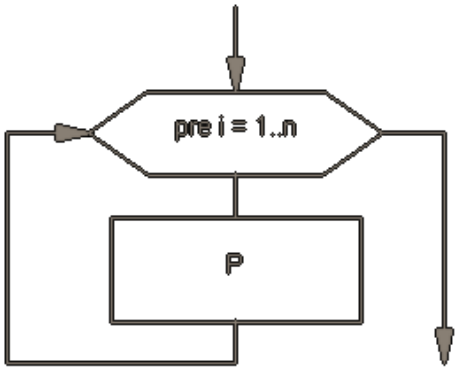


Cyklus

- Súčasťou riadiacej zložky algoritmického jazyka okrem sekvencie a vetvenia je aj cyklus
- **Umožňuje opakovať ľubovoľnú činnosť alebo činnosti**
- Pri cykle musíme určiť:
  - **Čo sa má opakovať?**

  V prípade robotického vysávača je to postupnosť príkazov *posun 50cm* a *vysaj*
  - **Dokedy alebo koľkokrát sa má činnosť opakovať?**

  V našom prípade 2m pásu je to 4-krát
- Zápis vo vývojovom diagrame ako šesťuholník
- V šesťuholníku zadefinujeme koľkokrát sa majú príkazy vykonať
- Premenná ***i*** sa označuje ako riadiaca premenná
  - „Pamätá si“ koľkokrát cyklus prebehol
  - V zápise cyklu určíme jej dolnú a hornú hranicu (napr. od 1 po *n*)
  - Slúži ako počítadlo, ktoré cyklus po každom prebehnutí príkazov zvýši (o 1)
  - Následne hodnotu *i* porovná so stanovenou koncovou hodnotou – ak sa už rovnajú, tak cyklus skončí



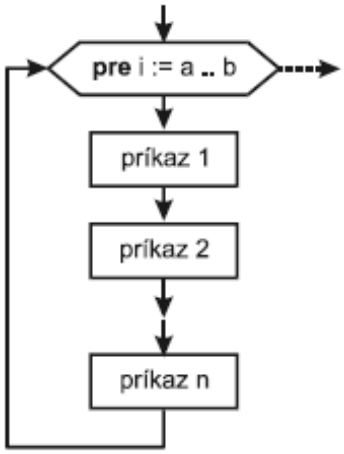
- Do *i* sa nastaví na začiatku hodnota 1, pri každom vykonávaní cyklu sa vykoná príkaz/y *P*, zvýši sa hodnota *i* (o 1) až kým sa *i* nerovná *n*

Typy cyklov

- V závislosti od vzťahu medzi telom a podmienkou cyklu môžeme cykly rozdeliť na:
    - Cyklus so známym počtom opakovaní
    - Cyklus s neznámym počtom opakovaní
        - Cyklus s podmienkou na začiatku
        - Cyklus s podmienkou na konci

Cyklus so známym počtom opakovaní

- Predpokladom využitia takéhoto cyklu je, že počet opakovaní sa dá vyjadriť pred jeho odštartovaním a operácie v tele cyklu naň nemajú vplyv
- Vo všeobecnosti ho možno zapísať takto:

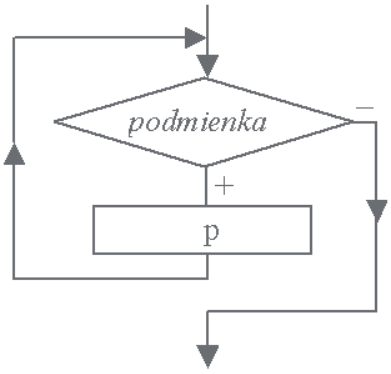


Cykly s neznámym počtom opakovaní

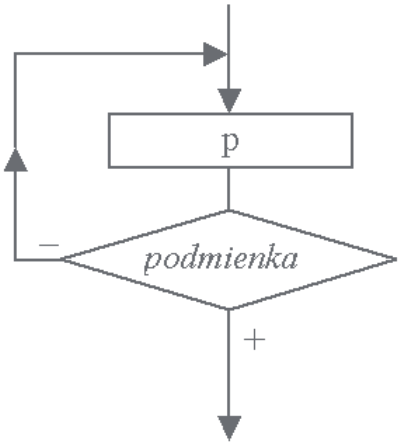
- Využívame ich vtedy, keď nám počet opakovaní **nie je známy v momente vstupu do cyklu** a/alebo **ukončenie cyklu ovplyvňujú operácie v jeho tele**
- Kontrolu ukončenia cyklu môžeme realizovať:
    - Pred vykonaním tela cyklu – **cyklus s podmienkou na začiatku**
    - Po vykonaní tela cyklu – **cyklus s podmienkou na konci**

Cyklus s podmienkou na začiatku

- Podmienka, ktorá sa stará o ukončenie cyklu, je umiestnená pred telom cyklu
- Ak je podmienka splnená, vykoná sa telo cyklu a opäť sa otestuje
- Ak vstupná podmienka nie je splnená, príkazy v tele cyklu sa nevykonajú
    - Cyklus sa nemusí vykonať ani raz – ak podmienka nie je splnená už pri prvom vstupe do cyklu
    - „opatrný cyklus“
- Využitie: pomerne univerzálne a časté



Cyklus s podmienkou na konci



- Najskôr sa vykoná telo cyklu a až potom sa zisťuje splnenie podmienky
- Ak je podmienka cyklu splnená, vykonávanie cyklu sa ukončí
- V opačnom prípade (ak podmienka nie je splnená) sa pokračuje opätovným vykonávaním tela cyklu
- -> Cyklus vždy prebehne minimálne raz
- Využitie napr.: kontrola vkladaných hodnôt
- „Neopatrný“ cyklus – najskôr sa niečo vykoná a potom sa rozhoduje,
  či to bolo dobre
- V prípade cyklu s podmienkou na začiatku – najskôr sa otestuje platnosť podmienky a až potom sa vykonáva – „opatrný“ cyklus