Štruktúry údajov a algoritmy

OBSAH

Typ údajov

- každá konštanta, premenná, výraz alebo funkcia sú určitého typu
- typ konštanty, premennej alebo funkcie sa explicitne stanovuje v rámci ich deklarácie
- základné vlastnosti pojmu, ako ich budeme chápať:
  1. Typ údajov určuje množinu hodnôt, do ktorej daná konštanta prináleží, alebo ktoré môže daná premenná či výraz nadobudnúť, alebo ktorú môžeme daným operátorom alebo funkciou vypočítať.
  2. Typ hodnoty konštanty, premennej alebo výrazu sa dá určiť na základe ich deklarácie alebo zápisu bez nevyhnutnosti uskutočniť vlastný výpočtový proces programu.
  3. Každý operátor alebo funkcia predpokladá argumenty presne definovaných typov a poskytuje výsledok tiež presne definovaného typu. V prípade, že operátor pripúšťa argumenty rôznych typov (napr. + pre celé alebo reálne čísla), dá sa typ výsledku určiť podľa špecifických pravidiel príslušného programovacieho jazyka.

Primitívne typy údajov

- **Enumeračný** typ – malý počet hodnôt, reprezentované celými číslami, nad nimi sa nevykonávajú žiadne numerické operácie

```
enum {BIELA, MODRA, CERVENA} farba;
          typ           premenná
```

- **Celočíselný** typ – podmnožina množiny celých čísel, ktorej veľkosť je rôzna pre rôzne typy počítačov. Operácie nad nimi spĺňajú bežné aritmetické pravidlá.

```
int a, b;
```

- **Reálny** typ – podmnožina reálnych čísel

```
float x;    double y;
```

- **Znakový** typ – množina znakov vytlačiteľných na danom počítači (ASCII, UNICODE)

```
char c;
```

- Typ pre **logické pravdivostné hodnoty** – obsahuje dve hodnoty, zvyčajne sa označujú ako `true` a `false`, + logické operátory (jazyk C nepodporuje)

Štruktúrované typy údajov

- Základné metódy štruktúrovania: **pole** (array), **záznam** (struct, record), **množina** (set), **postupnosť** (file - súbor)
- Zložitejšie štruktúry sa zvyčajne nedefinujú ako statické typy, ale v priebehu výpočtu programu sa vytvárajú dynamicky (viď neskôr spojkové zoznamy, stromy), pričom sa môže meniť ich veľkosť a tvar
- **Pole** – homogénna štruktúra, ktorá pozostáva z prvkov jediného typu (tzv. bázový alebo základný typ).
  - Štruktúra s náhodným prístupom – všetky prvky môžu byť vybraté náhodne a sú rovnako sprítupniteľné
  - Referencia individuálneho prvku – meno + index
  - (kardinalita(bázový typ))^{n} ... n je kardinalita typu indexu poľa

```
typ_prvkov   identifikátor[počet_prvkov]
```

```
int a[3];        //a je pole troch prvkov typu int
int x;           //celočíselná premenná x
x = a[0];        //priradenie prvého prvku poľa a do
                 //premennej x
```

Štruktúrované typy (2)

- **Záznam** – slúži na spájanie ľubovoľných prvkov (zložiek) – môžu byť aj štruktúrované, do zložených typov.
  - v matematike sa takýto zložený typ nazýva karteziánsky súčin zložkových typov
  - v oblasti spracovania údajov sa takéto zložené typy nazývajú záznamy
  - kardinalita(typ_1)* kardinalita(typ_2)*...* kardinalita(typ_n)

```
struct názov {   typ1   id_prvku1;
                 . . .
                 typn   id_prvkun;
} zoznam_ premenných;
struct bod {     //bod je menovka
   int   x;      //x je členská premenná štruktúry
   int   y;      //y je členská premenná štruktúry
} p, q;          //p, q sú premenné typu záznam bod
p.x = 1; p.y = 2;      //p predstavuje bod [1,2]
```


Algoritmus

Definícia:

Algoritmus je presný popis definujúci výpočtový proces, ktorý vedie od meniteľných vstupných údajov až k požadovaným výsledkom.

Vlastnosti:

- **Determinovanosť – presnosť a zrozumiteľnosť**
  Po každom kroku je presne určené, aký bude krok nasledujúci.
- **Rezultatívnosť – zameranie na získanie hľadaných výsledkov**
  Proces, ktorý vedie vždy k výsledkom.
- **Konečnosť – poskytnutie výsledkov za konečný počet krokov**
  Konečnosť v čase – rezultatívnosť, konečnosť v priestore – reprezentovateľnosť konečným počtom príkazov.
- **Hromadnosť – meniteľné vstupné údaje**

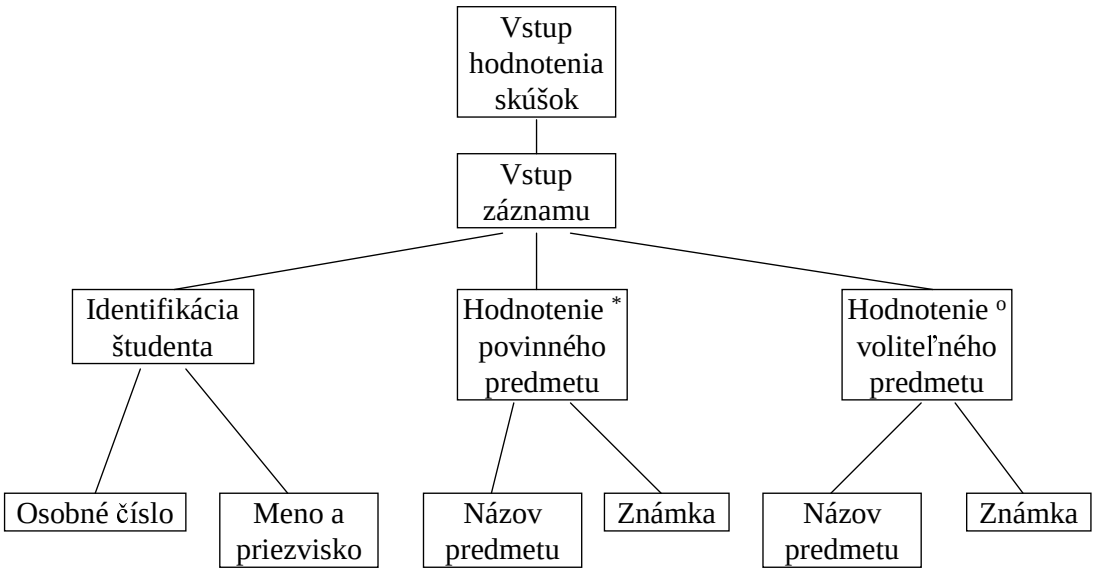
Vyjadrenie algoritmov

Textový popis – slovné vyjadrenie algoritmu – zvyčajne je to pre nás jednoduché, ale dovoľuje nejednoznačnosť výkladu alebo je rozsiahly.

Grafické vyjadrenie:

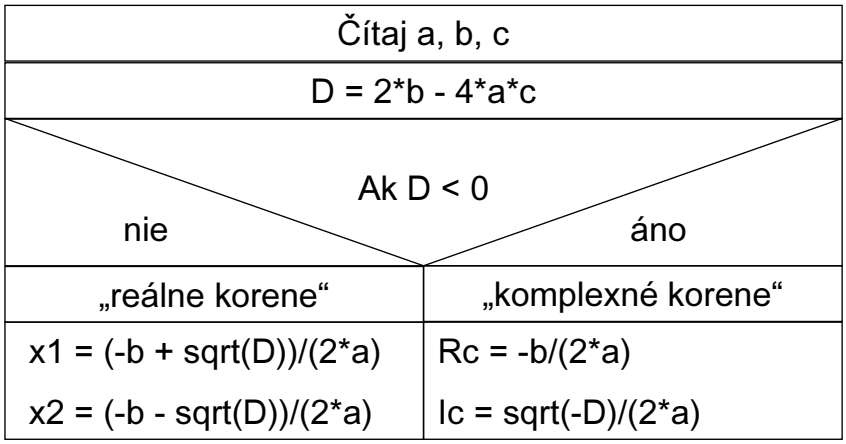
- Vývojové diagramy – súbor štandardizovaných značiek, pomocou ktorých zobrazujeme postupnosť riešenia úlohy.
- Štruktúrogramy – stromový diagram, ktorý popisuje štruktúru, tok riadenia, prípadne aj tok dát v programe alebo v module programu. Uzly diagramu tvoria bloky zodpovedajúce volaniu podprogramov alebo bloky preddefinovaných činností.
- Nassi-Schneidermanovej diagramy (NS-diagramy) – popisujú štruktúru a tok riadenia štruktúrovaným spôsobom. Na rozdiel od štruktúrogramov nie je použitý strom, ale bloky špeciálnych tvarov.

Príklad štruktúrogramu



Radenie blokov zľava doprava – sekvencia, príznak iterácie (*) – cyklus, príznak voliteľnosti (o) - vetvenie

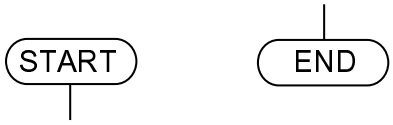
Príklad NS-diagramu



Radenie za sebou definuje sekvenciu, vetvenie je reprezentované trojuholníkovým blokom s definovaním podmienky a cyklus má špeciálny tvar

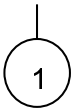
Vývojové diagramy (1)

Počiatočná a koncová značka



V počiatočnej značke môže byť aj názov algoritmu alebo podprogramu. Koncových značiek môže byť viac

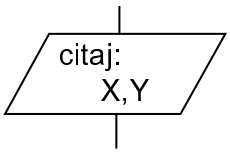
Návestie

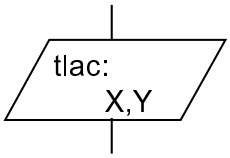


Touto značkou sa označuje prerušená spojnica v diagrame, napr.ak diagram má viac strán v dokumentácii

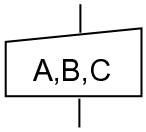
Vývojové diagramy (2)

Vstup/výstup dát – týmto symbolom sa označujú všetky vstupy a výstupy periférnych zariadení

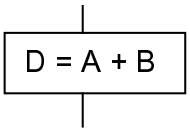




Ručný vstup dát – vstup údajov z klávesnice



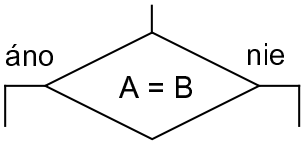
Spracovanie – výpočet, prenos hodnôt, rušenie záznamu,...



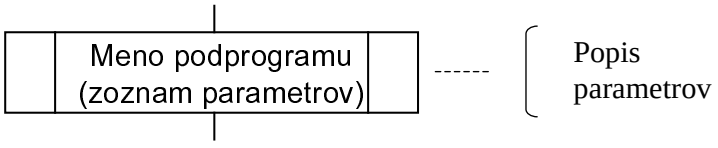
Vývojové diagramy (3)

Vetvenie (selekcia) – postup spracovania programu sa mení v dôsledku logickej podmienky

2-cestné vetvenie
3-cestné vetvenie
viac-cestné vetvenie - prepínač



Samostatne definovaná činnosť (podprogram) – samostatný úsek programu, ktorý možno vyvolať z ľubovoľného miesta programu. Jeho definícia je na inom mieste



Základné prvky algoritmov

Údajové objekty:

- atribúty – typ, hodnota, umiestnenie (adresa),
- klasifikácia – jednoduché (skalárne, neštruktúrované) a zložené (štruktúrované – napr. pole, záznam, množina, sekvenčný súbor),
- spôsoby reprezentácie v pamäti – výskyt objektu v programe ako konštanta alebo premenná,
- spôsoby odvolávania sa na objekty – konštanty označujeme ich hodnotou a na označenie premenných sa používajú identifikátory,
- spôsoby ukladania štruktúrovaných objektov v pamäti a triedy pamäte – pre údajové objekty sa vyhradzuje súvislá oblasť pamäte (možná požiadavka zarovnať objekt na určitú hranicu); statické (existujú počas celého behu programu) a dynamické/automatické údajové objekty (existujú len počas behu určitej časti programu)

Základné prvky algoritmov (2)

- viditeľnosť objektov:
  - lokálne - viditeľné len v rámci určitých častí programu (zvyčajne podprogramov),
  - globálne – viditeľné v rámci celého programu.
- Definícia údajového objektu – špecifikácia konkrétneho objektu daného typu. Priamo súvisí s procesom pridelenia pamäťového miesta.
- Deklarácia údajového objektu – popis typu tohto objektu. Deklaráciou určujeme, ako sa má s údajovým objektom v programe zaobchádzať. Spravidla sa používa v podprogramoch, do ktorých sa odovzdávajú údajové objekty.

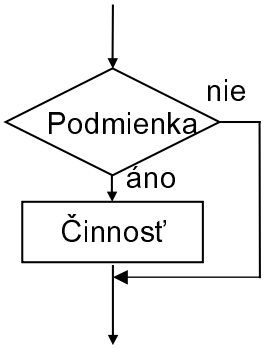
Poznámka: *Definíciu objektu môžeme v programe vykonať iba raz, kým deklarácií objektu sa v rámci zdrojového textu môže vyskytnúť ľubovoľný počet.*

Základné riadiace štruktúry

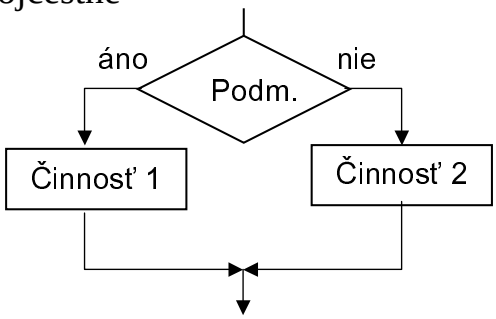
Sekvencia: príkazy sa vykonajú v takom poradí, ako sú uvedené (default tok zhora-nadol)

Vetvenie: výber z viacerých (aspoň dvoch) možností

- Preskok



Dvojcestné



- Viaccestné – každá vetva je určená pre množinu hodnôt
- Prepínač – každá vetva je určená pre konkrétnu hodnotu výrazu uvedeného v podmienke

Základné riadiace štruktúry (2)

Cyklus: špecifikovanie opakujúcich sa činností; pozostáva z:

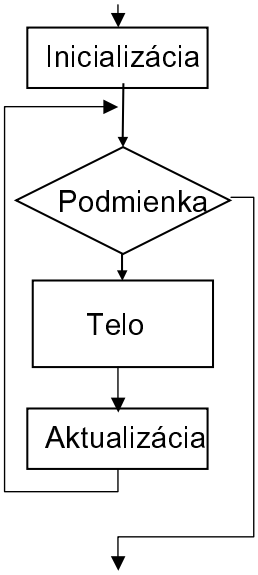
1. Inicializácie
2. Vykonania tela cyklu
3. Testu podmienky ukončenia cyklu
4. Prípravy ďalšieho kroku cyklu – aktualizácie.

Delíme ich na:

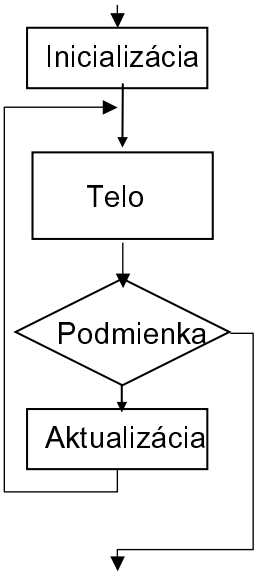
- Aritmetické (vo fáze inicializácie poznáme presný počet opakovaní cyklu)
- Logické
  - prefixné – test podmienky je pred telom cyklu
  - postfixné – test podmienky za telom cyklu

Základné riadiace štruktúry (3)

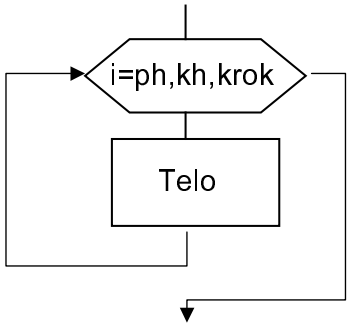
prefixný



postfixný



aritmetický



ph – počiatočná hodnota

kh – konečná hodnota

Príklad: Najväčší spol. deliteľ

Formulácia úlohy:

Pre dve načítané celé čísla z klávesnice zistite a na štandardný výstup vypíšte najväčší spoločný deliteľ.

Analýza úlohy:

Pre výpočet najväčšieho spoločného deliteľa (NSD) použijeme euklidovský algoritmus pre výpočet NSD dvoch kladných celých čísel. Vychádza z faktu, že NSD dvoch kladných celých čísel `x` a `y` je zároveň aj NSD čísel `y` a zvyšku po delení `x/y`.

Navrh údajových štruktúr:

V programe budeme potrebovať tri celočíselné premenné pre uloženie dvoch vstupných čísel (`x, y`) a pomocnú premennú uchovávajúcu zvyšok po celočíselnom delení (`z`).

Príklad: NSD

```
#include <stdio.h>
main()
{
   int x, y, zvysok;
   printf(“zadaj x: ”);
   scanf(“%d”, &x);
   printf(“zadaj y: ”);
   scanf(“%d”, &y);
   zvysok = x%y;
   while(zvysok != 0) {
       x = y;
       y = zvysok;
       zvysok = x%y;
   }
  printf(“Najvacsi spolocny delitel je: %d\n”, y);
}
```