

# IZP - vzorové řešení

---

**A, Příklad 1: [2 body]** Je definován typ:

```
typedef enum {AUTO, VLAK, LETADLO, AUTOBUS} DOPRAVA;
```

Definujte, co je kardinalita datového typu. Podle uvedené definice  
uveďte, jaká je kardinalita typu DOPRAVA?

**Řešení:**

Definice: *Kardinalita datového typu T je počet různých hodnot  
příslušejících typu T.*

Kardinalita typu DOPRAVA je: 4

**B, Příklad 1: [2 body]** Je definován typ:

```
typedef enum {CERVENA,ZELENA,ZLUTA} BARVY;
```

Definujte, co je kardinalita datového typu. Podle uvedené definice  
uveďte, jaká je kardinalita typu BARVY?

**Řešení:**

Definice: *Kardinalita datového typu T je počet různých hodnot  
příslušejících typu T.*

Kardinalita typu BARVY je: 3

# IZP - vzorové řešení

---

**A, Příklad 2: [2 body]** Je dán následující kód:

```
signed int z, x = 7, y = 22;
```

```
z = x++ - y/5;
```

Jaká bude hodnota proměnné z po jeho provedení?

**Řešení:** Hodnota proměnné z bude rovna 3.

**B, Příklad 2: [2 body]** Je definováno inicializované pole:

```
int a [10] = {1,2,3,[8]=8,10};
```

Jakou hodnotu bude mít prvek a[10] ?

**Řešení:** Není definováno.

# IZP - vzorové řešení

---

A, Příklad 3: [2 body] Je dán následující kód:

```
#include <stdio.h>

struct osoba{ int vek; double vaha; int vyska;};
typedef struct osoba CLOVEK;

int main(void)
{ CLOVEK c4 = {.vek = 30}; CLOVEK c3 = {.vyska = 175};
  c4 = c3; printf ("%d ", c4.vek); return 0; }
```

Co se zobrazí na standardní výstup po jeho provedení?

Řešení: Zobrazí se 0.

B, Příklad 3: [2 body] Je dán následující kód:

```
unsigned int a = 1, b = 2, c;
c = a & b || a && b;
```

Jakou hodnotu bude mít proměnná c po provedení tohoto kódu?

Řešení: Proměnná c bude mít hodnotu 1.

# IZP - vzorové řešení

---

**A, Příklad 4: [4 body]** Jaká bude hodnota proměnné sum po provedení následujícího kódu?

```
int sum = 0;  
for (int i = 0; i < 10; i++) {  
    switch (i) {  
        case 1: case 4: case 7: sum++;  
        default: continue;  
        case 5: break;  
    }  
    break; }
```

**Řešení:** Proměnná sum bude mít hodnotu 2.

**B, Příklad 4: [4 body]** Jaká bude hodnota proměnné sum po provedení následujícího kódu?

```
int sum = 1;  
for (int i = 0; i < 10; i++) {  
    switch (i) {  
        case 1: case 4: case 7: sum++;  
        default: continue;  
        case 5: break;  
    }  
    break; }
```

**Řešení:** Proměnná sum bude mít hodnotu 3.

# IZP - vzorové řešení

---

A, Příklad 5: [5 bodů] Je definován typ:

```
typedef float (*Pcena)(double, int);
```

Popište slovně tento definovaný typ.

Řešení:

Pcena je ukazatel (pointer) na funkci se dvěma parametry specifikovanými pro předávání hodnotou, která vrací hodnotu typu float.

B, Příklad 5: [5 bodů] Je definován typ:

```
typedef double (*Pcena)(double, int);
```

Popište slovně tento definovaný typ.

Řešení:

Pcena je ukazatel (pointer) na funkci se dvěma parametry specifikovanými pro předávání hodnotou, která vrací hodnotu typu double.

# IZP - vzorové řešení

---

**A, Příklad 6: [4 body]** Uveděte výhody modulárního programování (alespoň 4 výhody). Každou výhodu stručně charakterizujte.

**Řešení:**

- srozumitelnost (modul se zabývá pouze jedním problémem)
- spolehlivost (jasně definovaná jednotka je lépe testovatelná)
- udržovatelnost (malé části se lépe udržují, mění, upravují)
- znovupoužitelnost (efektivní využití zdrojů)
- možnost souběžného vývoje (více modulů může být využito souběžně)

**B, Příklad 6: [4 body]** Uveděte, které programové konstrukce jsou z hlediska tvorby modulů vhodné v hlavičkovém souboru (.h) a které ve zdrojovém souboru (.c).

**Řešení:**

**hlavičkový soubor (alespoň 4 konstrukce):** prototypy funkcí, deklarace datových typů, obecná makra, dopředné deklarace globálních proměnných

**zdrojový soubor (alespoň 2 konstrukce):** definice funkcí, deklarace globálních proměnných

# IZP - vzorové řešení

A, Příklad 7: [8 bodů] Upravte následující funkci tak, aby zobrazila všechny prvky nad vedlejší diagonálou čtvercové matice řádu n (po řádcích v původním pořadí) uložené ve dvourozměrném poli, daném parametrem array.

```
void printAboveSideDiag (int n, int array[n][n])
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n-i-1; j++)
            printf ("%d ", array[i][j]); //!! tento příkaz
        neupravujte!!
        printf ("\n");
    }
    return;
}
```

# IZP - vzorové řešení

**B, Příklad 7:** [8 bodů] Upravte následující funkci tak, aby zobrazila všechny prvky pod vedlejší diagonálou čtvercové matice řádu n (po řádcích v původním pořadí, tj. v obou směrech ve směru rostoucích indexů) uložené ve dvourozměrném poli daném parametrem array.

```
void printUnderSideDiag (int n, int array[n][n])
// parametr n je řád matice
{
    for (int i = n-1; i < n; i++)
    {
        for (int j = 1; j < n; j++)
            printf ("%d ", array[i][j]); //!! tento příkaz
neupravujte !!
        printf ("\n");
    }
    return;
}
```

# IZP - vzorové řešení

**A, Příklad 8:** [10 bodů] Je dáno:

```
typedef struct { ..; int pay; } tdata;
struct item { tdata data; item *next; };
typedef struct item titem;
typedef struct { titem *head; .. } tlist;
```

Definujte funkci listFindMin, která vrací ukazatel na položku s minimální hodnotou složky pay v lineárním seznamu (daném parametrem funkce). V případě, že je seznam prázdný, funkce vrací NULL.

**Řešení:**

```
titem *listFindMin (tlist *list)
{
    titem *tmp = list->head;
    titem *minitem = tmp;
    while (tmp != NULL)
    {
        if (tmp->data.pay < minitem->data.pay)
            minitem = tmp;
        tmp = tmp->next;
    }
    return minitem;
}
```

# IZP - vzorové řešení

A, Příklad 9: [12 bodů] Co se zobrazí po provedení následujícího programu? Uveďte přesné výsledek, který se zobrazí na standardní výstup.

```
unsigned long vypocet (unsigned long a, unsigned long b) {  
    if (a * b == 0) return 0; // Místo A  
    if (a < b) {  
        a ^= b; b ^= a; a ^= b; // Místo B  
    }  
    return (a % b > 0) ? vypocet(b, a % b) : b; // Místo C  
}  
  
int main(void) {  
    unsigned long x = 35;  
    unsigned long y = 10;  
    printf("%lu", vypocet(x, y));  
    return 0;  
}
```

Řešení: (co se zobrazí?): 5

Význam funkce vypocet: Funkce vypočítá největší společný dělitel dvou přirozených čísel

Význam kódu v místě A: koncová podmínka rekurze, pokud bude některá z proměnných nulová, funkce vrátí nulu.

Význam kódu v místě B: swap, neboli výměna hodnot proměnných a a b, aby bylo před rekursivním voláním zaručeno, že  $a \geq b$ .

Význam kódu v místě C : rekursivní volání funkce vypocet.

# IZP - vzorové řešení

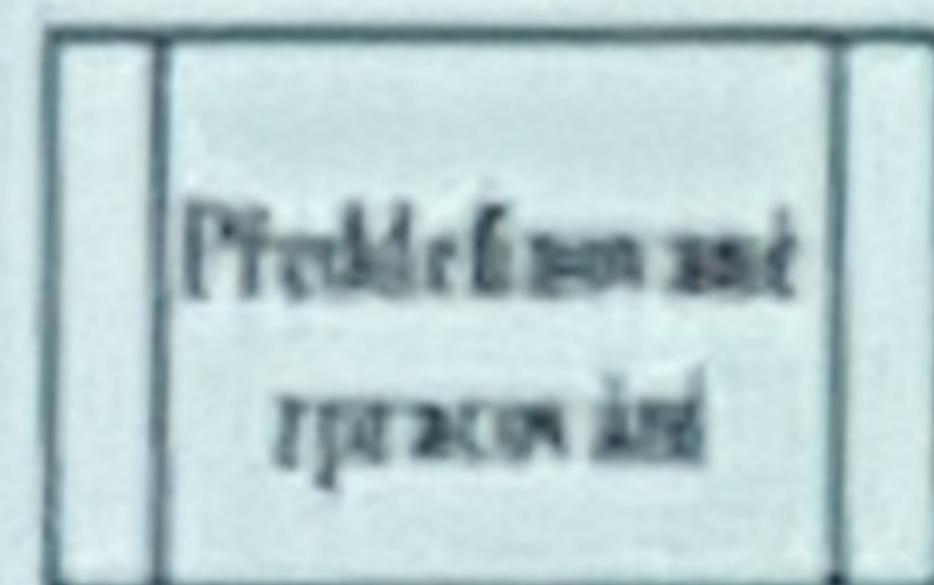
A, Příklad 10: [6 bodů] Uveďte základní řídící struktury v programování (3 struktury).

Každou strukturu stručně charakterizujte. Pro každou strukturu uveďte minimálně 2 příklady (příkazy), kterými lze danou strukturu v programu realizovat a jeden příklad symbolu, kterým lze daný příkaz struktury vyjádřit ve vývojovém diagramu.

Řešení:

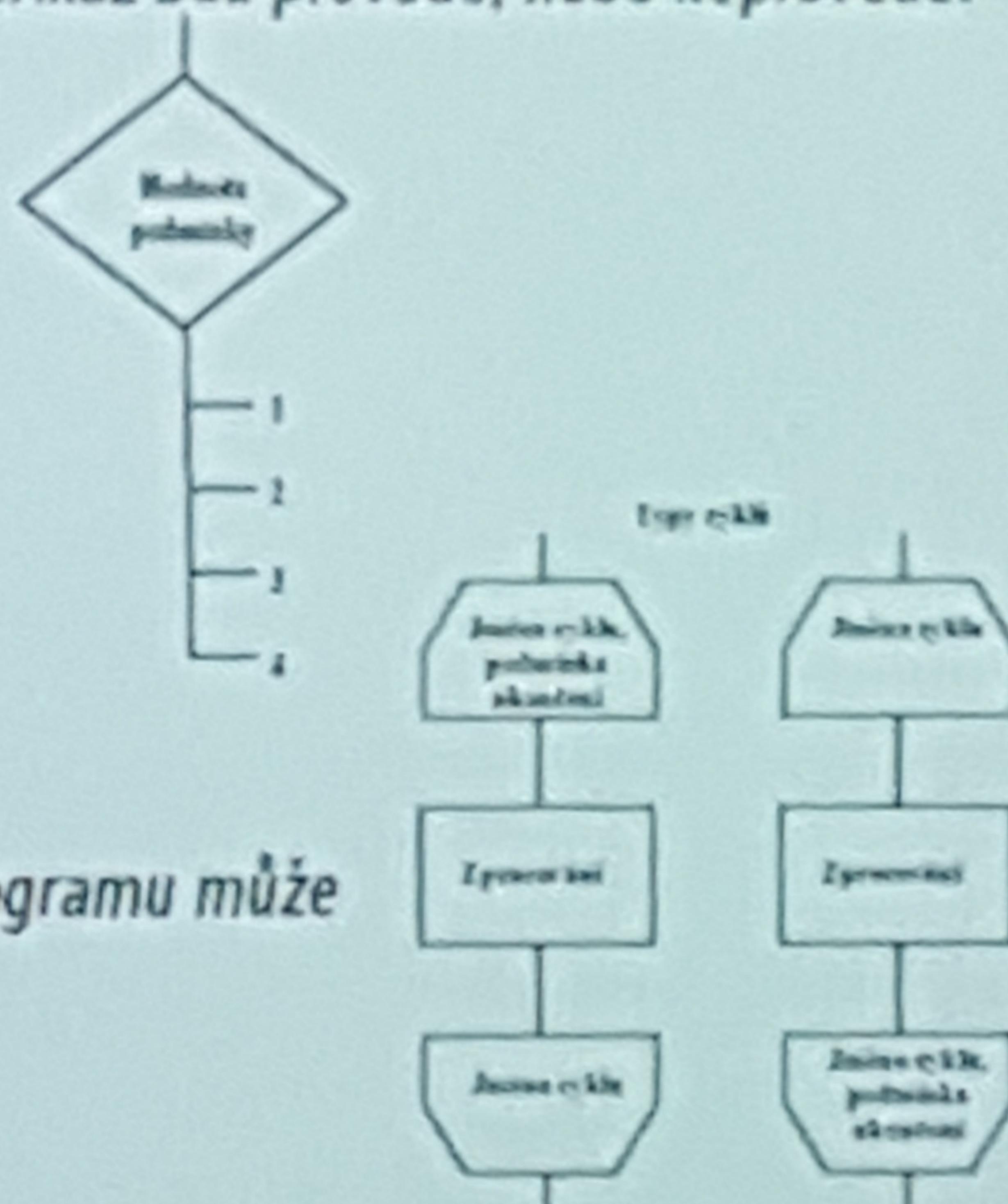
Sekvence: všechny příkazy se postupně provedou jeden po druhém v daném pořadí.

Příklad: přiřazení, volání funkci



Selekce: v závislosti na splnění podmínky se určitý příkaz bude provede, nebo neprovede.

Příklad: větvení if-then, if-then-else, switch



Iterace: v závislosti na splnění podmínky se část programu může vykonat vícekrát.

Příklad: cykly for, while, do-while

# IZP - vzorové řešení

**B, Příklad 10:** Stručně charakterizujte základní vlastnosti algoritmů (1-2 věty pro každou vlastnost). Uveděte příklad algoritmu, který není konečný a příklad algoritmu, který není hromadný. [6 bodů]

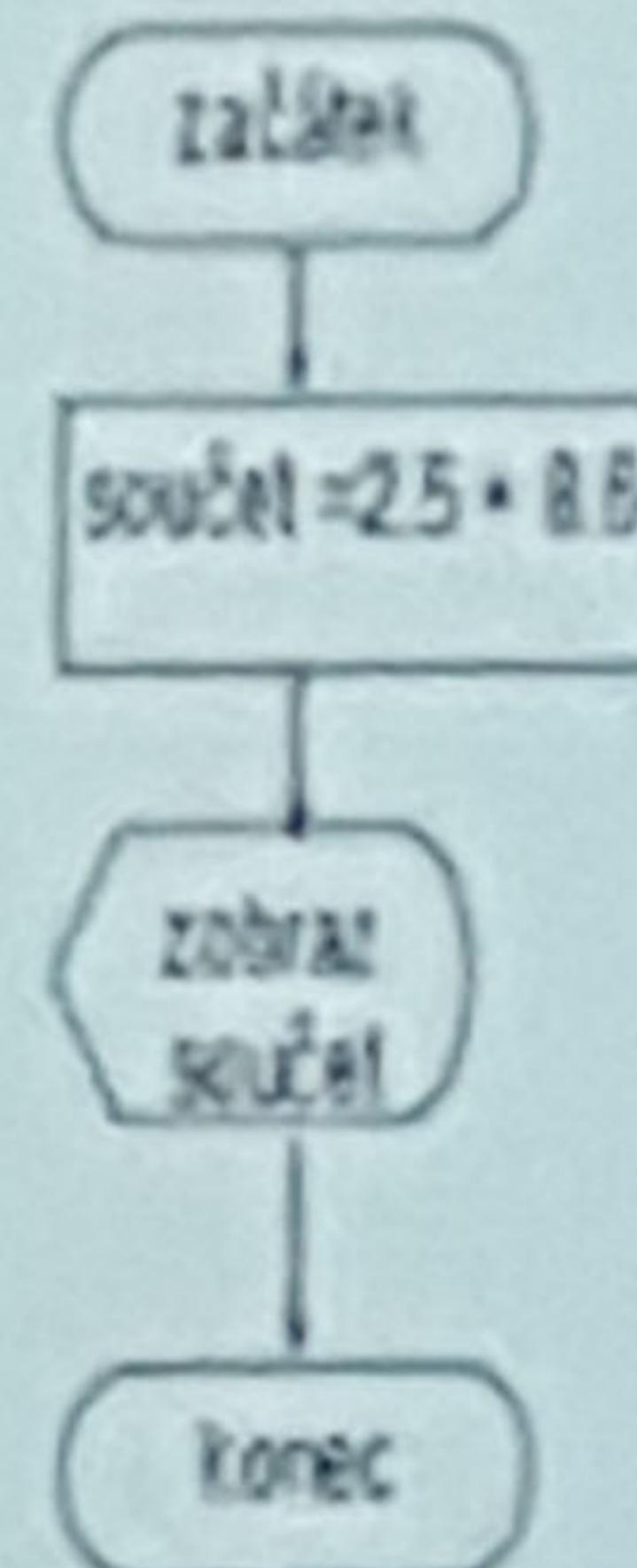
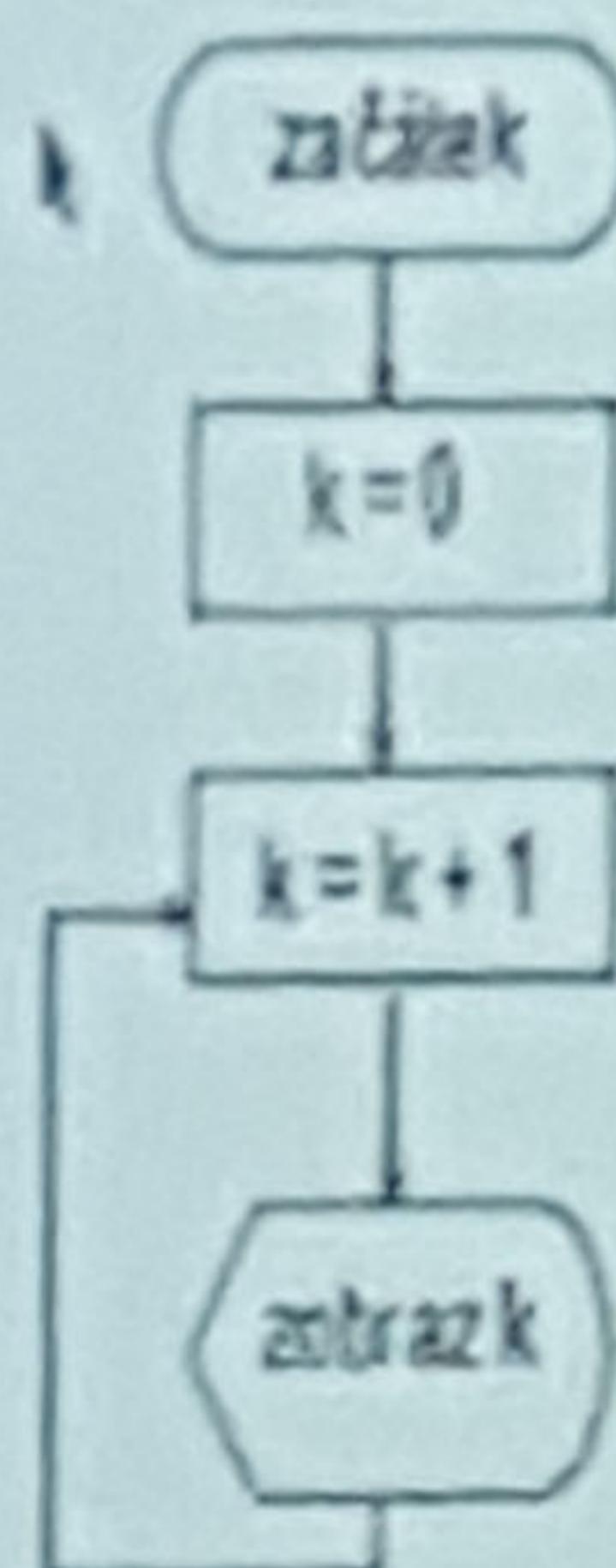
**Řešení:**

**Konečnost (rezultativnost):** má zaručit vyřešení úlohy po konečném počtu kroků.

**Hromadnost:** jedním algoritmem lze řešit celou třídu úloh stejného druhu.

**Příklad (není konečný):**

**Příklad (není hromadný):**



**Determinovanost:** algoritmus je zadáný ve formě konečného počtu jednoznačných pravidel.

**Efektivnost:** na správný průběh programu nemá žádný vliv, zajišťuje pouze to, aby program trval co nejkratší dobu.