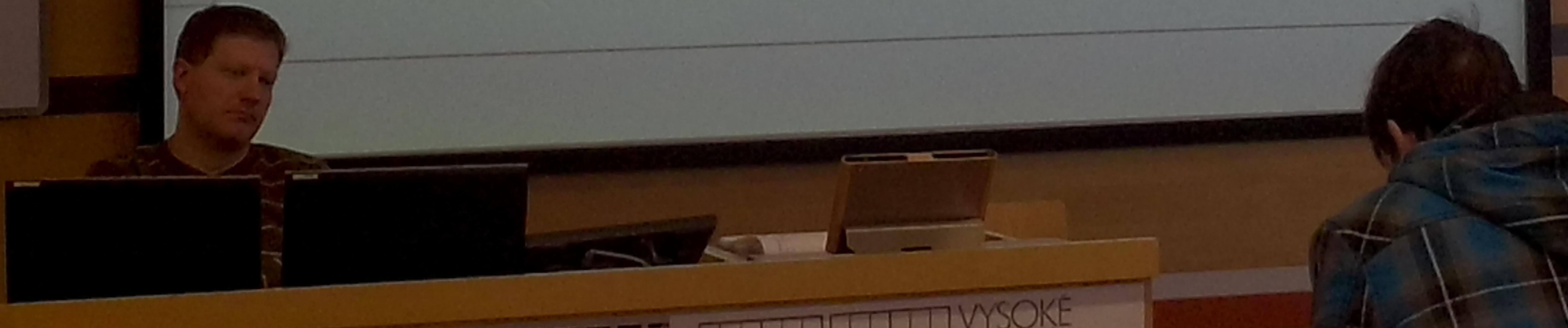


A, Příklad 1: Vysvětlete pojmy syntaxe a sémantika programovacích jazyků.

Řešení: Syntaxe: Soubor pravidel udávající přípustné konstrukce programů. Popisuje formální strukturu programu. Definuje klíčová slova, identifikátory, čísla a další programové entity a určuje zápis programu na základě syntaktických pravidel. (1b) **Sémantika:** Určuje logický význam jednotlivých výrazů jazyka. [2 body]

B, Příklad 1: Definujte pojem zarázka při sekvenčním vyhledávání v poli se zarážkou. Popište, v čem spočívá její výhoda. [2 body]

Řešení: Zarázka je **hledaná položka umístěná za konec pole**. Při průchodu polem při vyhledávání potom **není nutné opakovat testovat na konec pole** (ukončení zajišťuje zarázka).



A, Příklad 2: Co je výsledkem aplikace operátoru `sizeof`? [2 body]

Řešení:

Výsledkem aplikace operátoru `sizeof` je velikost jeho operandu v bajtech.

B, Příklad 2: Vysvětlete pojmy: „ukazatel“, „reference“ a „dereference“. Použijte operátory „reference“ a „dereference“ na vhodných operandech a výsledky uložte do proměnných vhodných typů (deklarujte proměnné vhodných typů). [3 body]

Řešení:

ukazatel = datový typ, jehož hodnota vyjadřuje umístění/adresu dat v paměti.

reference = hodnota pro nepřímý přístup k proměnné.

dereference = přístup k hodnotě proměnné přes její referenci.

použití operátoru reference: `int a = 5; int *b = &a;`

použití operátoru dereference: `int c = *b;`



A, Příklad 3: Je dána definice:

Int mat[n][n];

[3 body]

Jaký řád časové složitosti má algoritmus pro výpis všech hodnot po obvodu čtvercové matice řádu n, uložené v poli mat? Čím je dán rozsah (počet) vstupních dat? Pozn.: hodnotami po obvodu matice se myslí všechny hodnoty na prvním a posledním řádku a v prvním a posledním sloupci matice.

Řešení: řád složitosti: lineární (1b) rozsah dat je dán (vyjádřete vzorcem vzhledem k řádu matice): $4(n - 1)$ (2b)

B, Příklad 3: Je dán následující kód:

[2 body]

```
float f = 0.0f;  
for (int i=0; i<10; i++) f = f + 0.1;  
if (f == 1.0f) printf("výsledek je %6.4f", f); // XX
```

Co by se zobrazilo po jeho provedení? Odpověď zdůvodněte.

Řešení: Typ float je pouze approximací reálných čísel. 10x provedené sečtení hodnot 0.1 se pouze přibližuje hodnotě 1.0 (1b), ostrá rovnost (1b) v podmínce tedy platit nemusí.

A, Příklad 4: Uvedte příklady (název) datových struktur. U každé struktury také uvedte definici proměnné pro uvedený typ datové struktury. [4 body]

Řešení: **homogenní:** pole, **heterogenní:** záznam, **statická:** pole, **dynamická:** lineární seznam

B, Příklad 4: Čím se liší unární, binární a ternární operátory? Ke každému druhu uvedte příklad (stačí jeden z každého druhu) a stručný popis operátoru jazyka C. [4 body]

Řešení:

Čím se liší: Rozdíl je v počtu operandů (unární 1, binární 2, ternární 3). (1b)

unární: (- negace, ~ inverze, sizeof() počet bajtů výrazu nebo typu, ...). (1b)

binární: (+ součet, == ekvivalence, = přiřazení, „čárka“ předání hodnoty, ...) (1b)

ternární: (c ?a :b - podmíněný výraz, kde výsledná hodnota (a nebo b) závisí na hodnotě c). (1b)



A, Příklad 5: Je dána definice: `unsigned int r; // inicializace r`

Napište výraz vyjadřující, zda rok uložený v proměnné `r` je přestupný (true) nebo není přestupný (false). Přestupné roky, podle Gregoriánského kalendáře, jsou roky dělitelné 4, avšak roky dělitelné 100 jsou přestupné pouze tehdy, pokud jsou dělitelné i 400 (nepouživejte ternární operátor!).

[6 bodů]

Řešení:

`((r % 4 == 0) && ((r % 100 > 0) || (r % 400 == 0)))`

nebo lépe: `(!(r % 4) && ((r % 100) || !(rok % 400)))`

B, Příklad 5: Je dáno: `int x; // inicializace x`

Napište výraz, který nabývá hodnoty true, když hodnota uložená v proměnné `x` není dělitelná žádným číslem z množiny {3,5,7} (nepouživejte ternární operátor!).

[6 bodů]

Řešení:

`((x % 3 != 0) && (x % 5 != 0) && (x % 7 != 0))`

nebo lépe: `((x % 3) && (x % 5) && (x % 7))`



A + B, Příklad 6: Je definována funkce:

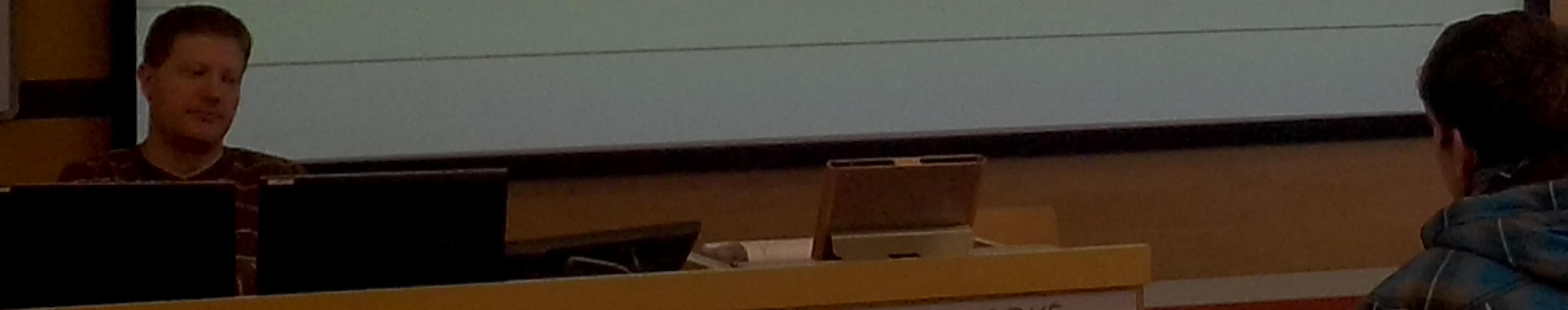
```
void test(int a, int b)
{
    if (a = b)
        printf("a je rovno b\n"); // xx
    else
        printf("a není rovno b\n");
}
```

Jaká by musela platit vstupní podmínka (pro parametry) této funkce, aby se **vždy** provedl příkaz na řádku označeném XX? [4 body]

Řešení: $b \neq 0$

B, Příklad 7: Upravte následující funkci tak, aby zobrazila všechny prvky **na hlavní a na vedlejší diagonále** čtvercové matice řádu n v pořadí zleva doprava, shora dolů uložené ve dvojrozměrném poli dané parametrem array. V případě, že některý prvek matice leží současně na hlavní i na vedlejší diagonále, zobrazí se v odpovídajícím řádku pouze jednou. [8 bodů]

```
void printMainSideDiag (int n, int array[n][n])
// parametr n je řád matice
{
    0      n
for (int i = 1; i < n-1; i++)
    if (i == n-i-1
        printf ("%d \n", array[i][i]);
    else
        printf ("%d %d \n", array[i][i<n/2 ? i:n-i-1],
                array[i][i<n/2 ? n-i-1:i]);
}
```



A, Příklad 8: Je dáno:

```
typedef struct { ...; int pay; } tdata;
struct item { tdata data; titem *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. [10 bodů]

Řešení:

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

A, Příklad 9: Co se zobrazí po provedení následujícího programu?
Uveďte přesně výsledek, který se zobrazí na standardní výstup.

[10 bodů]

```
#include <stdio.h>
void myPrint (int n) {
    printf ("%d", n/2);
    if (n > 0) // Místo A
        myPrint (n - 1); // Místo B
    printf ("%d", n);
}
int main (void) { int count = 4; myPrint (count); return 0; }
```

Popište, jaký význam má kód v místech označených jako A a B.

Řešení:

Zobrazí se: 2110001234 (6b)

Místo A: podmínka ukončení rekurze. (2b)

Místo B: rekurzívní volání funkce myPrint. (2b)

B, Příklad 9: Co se zobrazí po provedení následujícího programu?
Uveďte přesně výsledek, který se zobrazí na standardní výstup.

[10 bodů]

```
#include <stdio.h>
void myPrint (int n) {
    printf ("%d", n % 2);
    if (n > 0) // Místo A
        myPrint (n - 1); // Místo B
    printf ("%d", n / 2);
}
```

```
int main (void) { int count = 4; myPrint (count); return 0; }
```

Popište, jaký význam má kód v místech označených jako A a B.

→ **Řešení:**

← **Zobrazí se:** 0101000112 (6b)

↑ **Místo A:** podmínka ukončení rekurze. (2b)

↓ **Místo B:** rekurzívní volání funkce myPrint. (2b)

A, Příklad 10: Stručně charakterizujte základní vlastnosti metod řazení (1-2 věty pro každou vlastnost). [6 bodů]

Řešení:

Sekvenčnost: Je vlastnost, která vyjadřuje, že řadící algoritmus pracuje se vstupními údaji i s datovými meziprodukty v tom pořadí v jakém jsou lineárně uspořádány v datové struktuře. (1b)

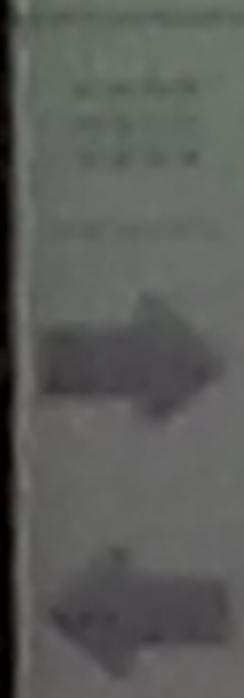
Časová složitost: Označuje míru času potřebnou k realizaci daného algoritmu. U řadicích algoritmů ji označujeme jako funkci počtu n řazených prvků. (1b)

Prostorová složitost: Označuje míru prostoru potřebnou k realizaci daného algoritmu. U řadicích algoritmů ji označujeme jako funkci počtu n řazených prvků. (1b)

Přirozenost: Je vlastnost algoritmu, která vyjadřuje, že doba potřebná k řazení již seřazené množiny údajů je menší než doba pro seřazení náhodně uspořádané množiny a ta je menší než doba pro seřazení opačně seřazené množiny údajů. (1b)

Stabilita: Je vlastnost řazení, která vyjadřuje, že algoritmus zachovává vzájemné pořadí údajů se shodnými klíči. (1b)

In situ: Metoda pracuje *in situ* znamená, že metoda nepožaduje výrazně větší prostor pro řazení, než zabírá původní řazená struktura. (1b)



B, Příklad 10: Stručně charakterizujte základní vlastnosti algoritmů (1-2 věty pro každou vlastnost). Uvedte příklad algoritmu, který není konečný a příklad algoritmu, který není hromadný.

[6 bodů]

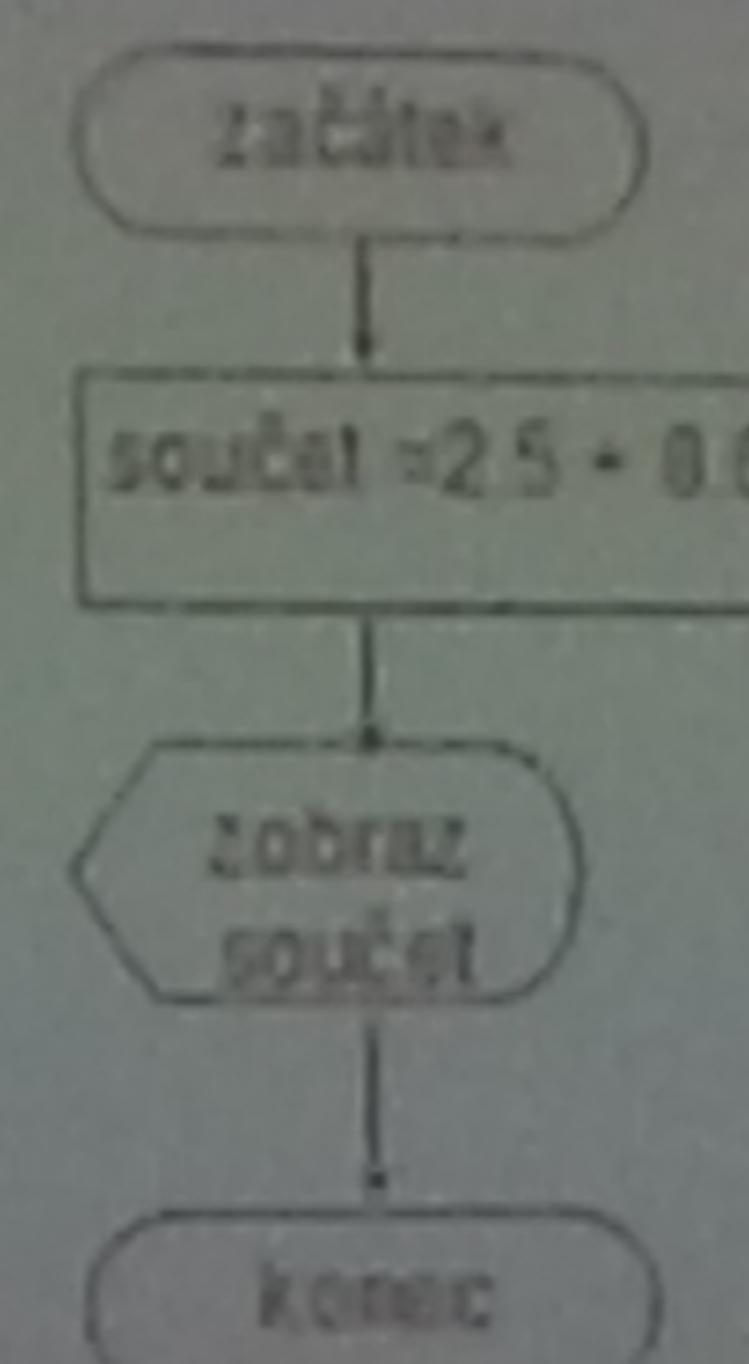
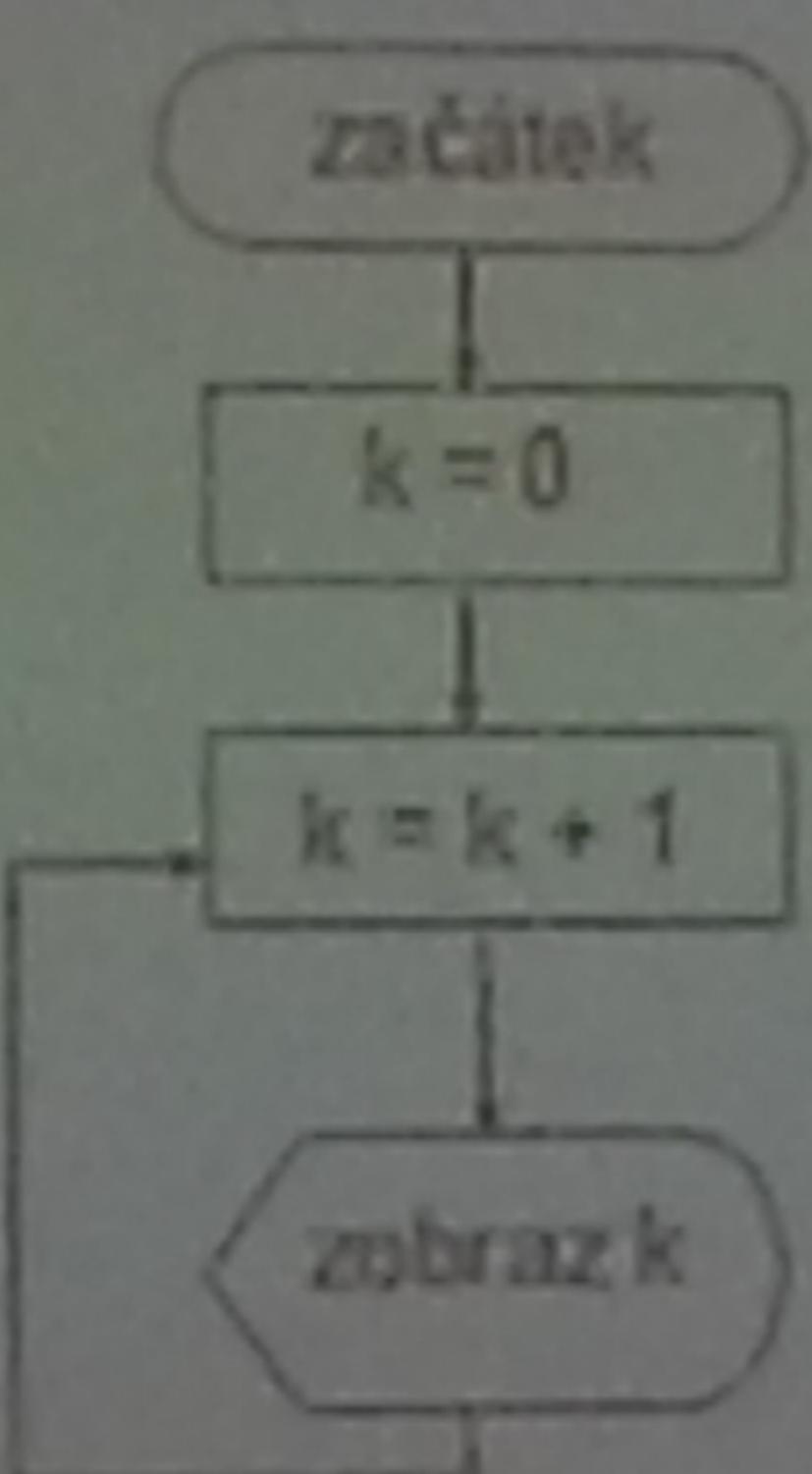
Řešení:

Konečnost (rezultativnost): má zaručit vyřešení úlohy po konečném počtu kroků. (1b)

Hromadnost: jedním algoritmem lze řešit celou třídu úloh stejného druhu. (1b)

Příklad (není konečný):

Příklad (není hromadný):



Determinovanost: algoritmus je zadáný ve formě konečného počtu jednoznačných pravidel. (1b)

Efektivnost: na správný průběh programu nemá žádný vliv, zajišťuje pouze to, aby program trval co nejkratší dobu. (1b)