

IZP – vzorové řešení

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

typedef enum {AUTO, VLAK, LETADLO, AUTOBUS, MOTORKA} 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 . (1b)

Kardinalita typu Doprava je: 5 (1b)

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

typedef enum {CERVENA, MODRA, 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 . (1b)

Kardinalita typu Barvy je: 4 (1b)

IZP – vzorové řešení

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

```
signed int z, x = 9, y = 22;
```

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

Jaká bude hodnota proměnné z po jeho provedení?

Řešení: Hodnota proměnné z bude rovna 5. (2b)

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

```
int a [11] = {1,2,3,[7]=10,9,8};
```

Jakou hodnotu bude mít prvek a[9] ?

Řešení: Prvek a[9] bude mít hodnotu 8. (2b)

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 = 25, .vyska = 165}; Clovek c3 = { .vek = 35};
  c4 = c3; printf ("%d ", c4.vyska); return 0; }
```

Co se zobrazí na standardní výstup po jeho provedení?

Řešení: Zobrazí se 0. (2b)

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

```
unsigned int a = 3, 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. (2b)

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 3: case 6: sum++;  
        default: continue;  
        case 4: break;  
    }  
    break;  
}
```

Rešení: 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 3: case 4: case 7: sum++;  
        default: continue;  
        case 5: break;  
    }  
    break;  
}
```

Rešení: Proměnná sum bude mít hodnotu 4.

IZP – vzorové řešení

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

typedef float (*Pmzda)(double, int);

Popište slovně tento definovaný typ.

Řešení:

Pmzda je ukazatel (pointer)(1b) na funkci (1b) se dvěma parametry specifikovanými pro předávání hodnotou (1b), která vrací hodnotu typu float. (1b)

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

typedef double (*Pmohutnost)(double, int);

Popište slovně tento definovaný typ.

Řešení:

→ Pmohutnost je ukazatel (pointer)(1b) na funkci (1b) se dvěma parametry specifikovanými pro předávání hodnotou (1b), která vrací hodnotu typu double. (1b)

IZP – vzorové řešení

A, Příklad 6: [4 body] Uvedte výhody a nevýhody modulárního programování. Každou položku stručně charakterizujte.

Výhody (alespoň 4) :

- srozumitelnost (moduly jsou jednoúčelné),
- spolehlivost (lépe testovatelné jednotky),
- udržovatelnost (menší části se lépe udržují),
- opětovné použití (absence chyb z kopírování kódu),
- možnost použití souběžného vývoje (více modulů může být vyvíjeno souběžně).

Nevýhody (alespoň 2):

- problémy vyplývající z integrace modulů (testování komunikace modulů),
- konkretizace modulů (možné chyby vyplývající z neznalosti implementačních detailů).

B, Příklad 6: [4 body] Definujte export a import modulu:

Export modulu: Množina funkcí, datových typů, identifikátorů, které modul poskytuje. (2 body)

Import modulu: Množina funkcí, datových typů, identifikátorů, které modul vyžaduje. (2 body)

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, tj. v obou směrech ve směru rostoucích indexů) uložené ve dvourozměrném poli, daném parametrem array. Výsledný algoritmus musí vykonat minimální možný počet iterací.

```
void printAbovesideDiag (int n, int array[n][n])
{
    for (int i = 0; i < n-1; i++)
    {
        for (int j = 0; j < n-i-1; j++)
            printf ("%d ", array[i][j]);
        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. Výsledný algoritmus musí vykonat minimální možný počet iterací.

```
void printUndersideDiag (int n, int array[n][n])
{
    for (int i = 1; i < n; i++)
    {
        for (int j = n - i; j < n; j++)
            printf ("%d ", array[i][j]);
        printf ("\n");
    }
}
```

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

B, Příklad 8: [10 bodů] Je dáno:

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

```
typedef struct { item *head; ... } tlist;
```

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

Řešení:

```
item *listFindMax (tlist *list)
{
    item *tmp = list->head;
    item *maxitem = tmp;
    while (tmp != NULL)
    {
        if (tmp->data.pay > maxitem->data.pay)
            maxitem = tmp;
        tmp = tmp->next;
    }
    return maxitem;
}
```

IZP – vzorové řešení

A, Příklad 9: [12 bodů] Co se zobrazí po provedení následujícího programu? Uveďte přesně 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 rekurrencie, 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 rekurzivním voláním zaručeno, že $a \geq b$.

Význam kódu v místě **C** : rekurzivní volání funkce **vypocet**.

IZP – vzorové řešení

B, Příklad 9: [12 bodů] Co se zobrazí po provedení následujícího programu? Uveďte přesně 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 = 56;  
    unsigned long y = 24;  
    printf("%lu", vypocet(x, y));  
    return 0;  
}
```

Řešení: (co se zobrazí?): 8

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 rekurzivním voláním zaručeno, že $a \geq b$.

Význam kódu v místě C: rekurzivní volání funkce vypocet.

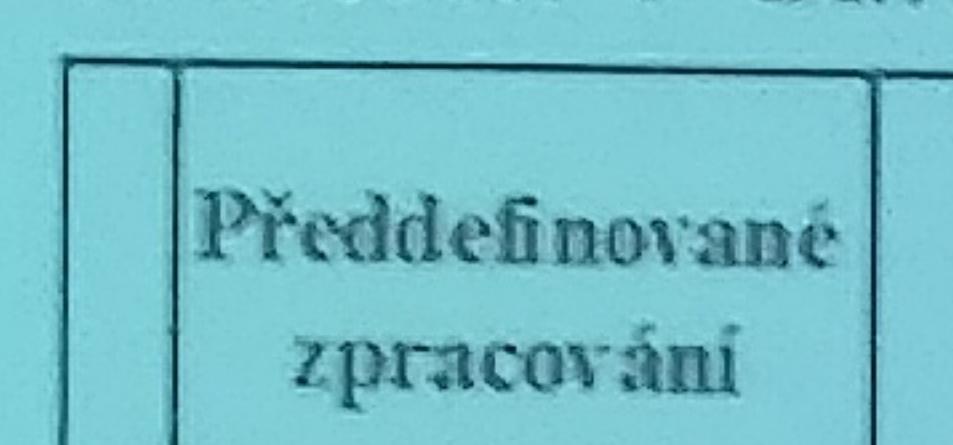
IZP – vzorové řešení

A, Příklad 10: [6 bodů] Uveďte základní řídicí 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:

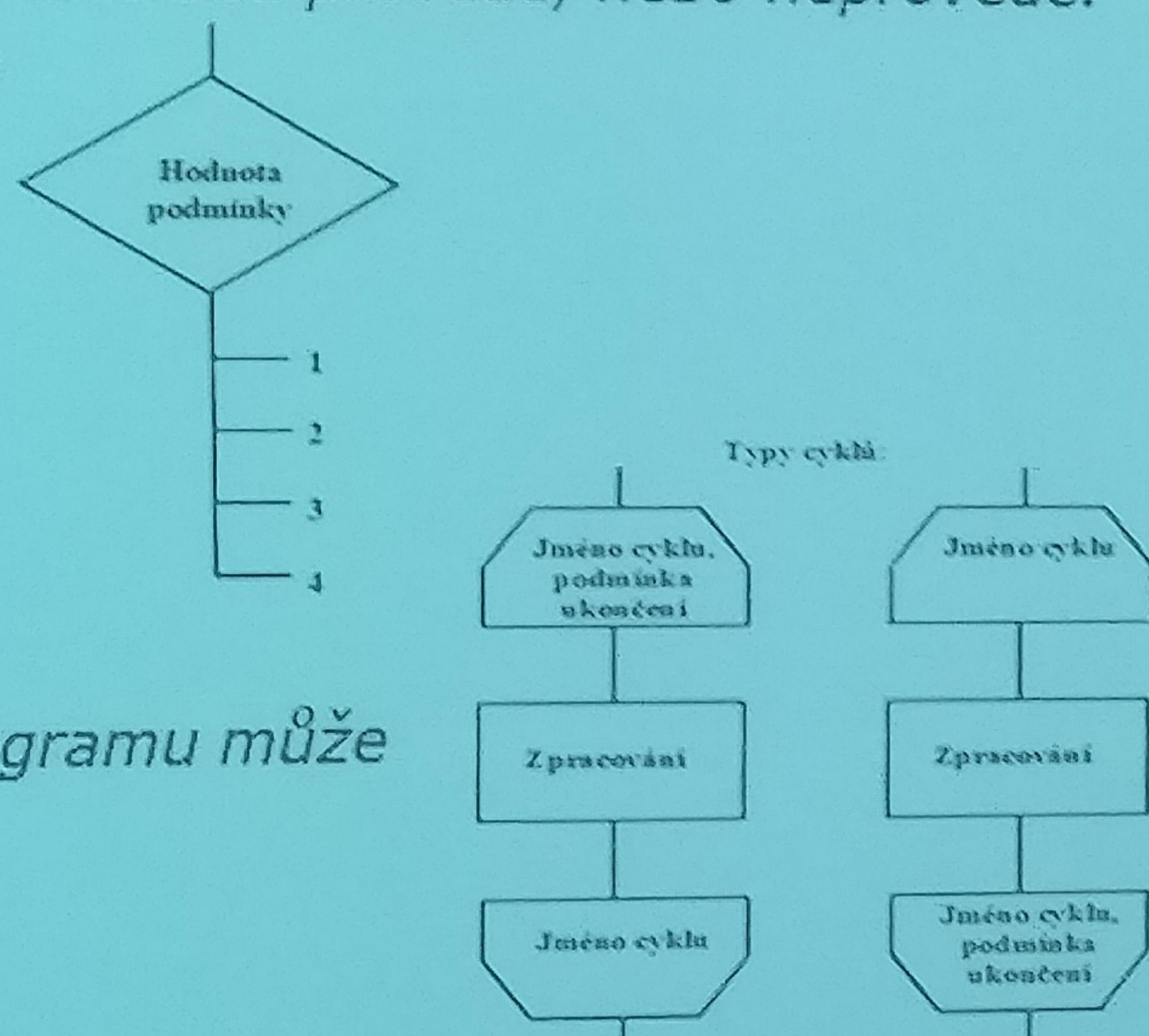
Sekvence: všechny příkazy se postupně provedou jeden po druhém v daném pořadí.

Příklad: přiřazení, volání funkcí



Selekce: v závislosti na splnění podmínky se určitý příkaz bud' 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