Obsah

- Porovnání JAVA vs „C“
  - Ukazatele, pointery
  - Pole
  - Pole a ukazatele
  - String
  - Pole a metody (předávání parametrů, návratová hodnota)
  - Vícerozměrné pole
  - Ukazatel na ukazatel
  - Dynamicky alokované pole

C - Ukazatel (pointer)

- Jak v Javě (reference)
  - Jak je ukazatel řešen v Javě?
    - Je možný ukazatel na ukazatel?
  - Lze použít pro volání odkazem?
  - Co může být cílem odkazu?
  - Jaké aritmetické operace lze s ukazatelem (referencí) provádět v jazyku Java?
  - Lze vytvářet seznamové struktury?

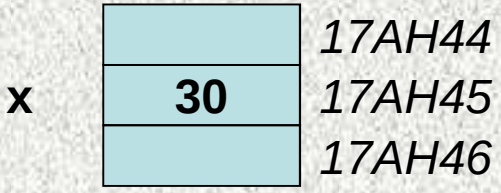
- Jak v C (pointery)
  - **Předávání parametrů odkazem**
  - **Práce s poli, řízení průchodu polem pointery**
  - **Pointer na funkci**
  - **Pole funkcí**
  - Na rozdíl od Javy je možné s ukazatelem provádět **aritmetické operace**
  - Ukazatel v C je přímo implementován v operační paměti a **je tedy možné přímo adresovat,** včetně požadavku na registry
    - Mocný nástroj pro implementaci strojově orientovaných aplikací
    - Řízené hešování

C - Ukazatel (pointer)

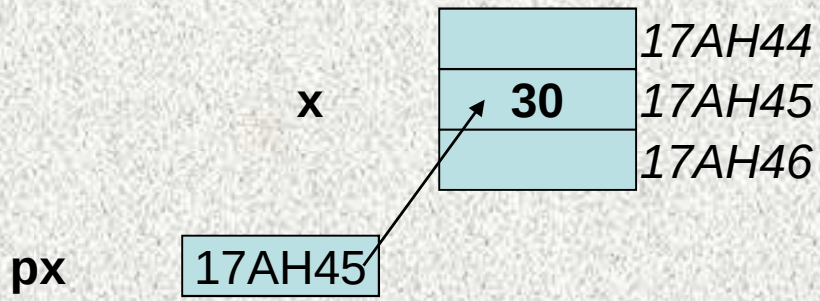
- *Ukazatel (pointer)* je proměnná jejíž hodnotou je „*ukazatel“ na jinou proměnnou* (analogie nepřímé adresy ve strojovém kódu či v assembleru)
- *Ukazatel* má též *typ* proměnné na kterou může *ukazovat*
  - ukazatel na char, int,..
  - „ukazatel na pole“
  - ukazatel na funkci
  - ukazatel na ukazatel, atd.
- Ukazatel může být též bez typu *(`void`),* pak může obsahovat *adresu libovolné proměnné*. Její velikost pak nelze z vlastností ukazatele určit
- Specialitou C je pointer na funkci!
- *„Prázdná“adresa,*ale definovaná v ukazateli má hodnotu konstanty **`NULL`**
- **C za běhu programu *nekontroluje* zda adresa v ukazateli *je platná***
- Pomocí *ukazatele* lze předávat parametry funkci *odkazem (call by reference*) – základní využití

C - Ukazatel (pointer)

- *Adresa proměnné* se zjistí adresovým operátorem **`&`** (ampersand), tzv. **referenční operátor** (*proměnná >>> adresa této proměnné*) **`int x;`**



  - *adresa proměnné* **`x`**, *pomocí operátoru* **`&, &x`** **~~~** *17AH45*
- K obsahu proměnné na kterou *ukazatel ukazuje se přistoupí* operátorem nepřímé adresy **`*`** *(hvězdička*), tzv. **dereferenční operátor** (proměnná ukazatel >> hodnota z adresy, kam ukazuje) **`int *px; px=&x;`**



  - *obsah proměnné, jejíž adresa je v proměnné* **`px`** *pomocí operátoru* **`*`** *(hvězdička)*, **`*px`** **~~~** *30*

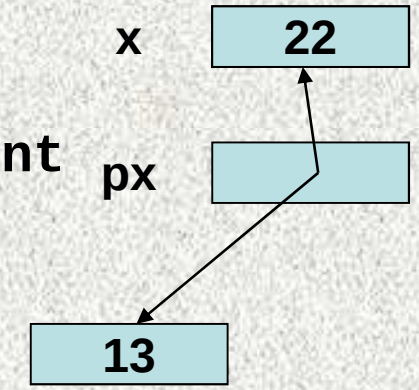
Základní znalost

C - Ukazatel (pointer)

```
int x=30;    // proměnná typu int
             // &x  - adresa proměnné x
int *px;     // *px proměnná typu int
             //  px proměnná typu pointer na int
px=&x;       // do proměnné typu pointer na int
             // se uloží adresa proměnné x
printf(" %d " "    %d  \n", x, px);
             // 30   2280564
printf(" %d " "    %d  \n", &x, *px);
             // 2280564  30
printf(" %d " "    %d  \n", *(&x), &(*px));
              // 30   2280564
```

C - Ukazatel (pointer)

```
int x; // proměnná typu int
int *px; // *px proměnná typu int
         // px proměnná typu pointer na int
x = 3; // přiřazení hodnoty int
       // do proměnné typu int
*px = 13; // přiřazení hodnoty int
       // do proměnné typu int (tj. *px)
       // určena proměnnou typu pointer (px)
// &x  - adresa proměnné x
px = &x; // do proměné px se uloží adresa
           // proměnné x
x = 22;
// *px == 22
```



C - Ukazatel (pointer)

Pozor při zápisu:

```
int x=5;
int *px = &x;  //definuje se proměnná px
               // a inicializuje se počáteční
               // hodnotou adresy proměnné x
```

je totožné s:

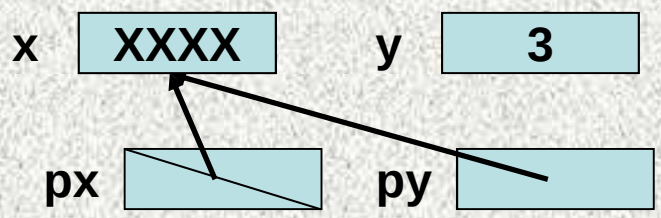
```
int x=5, *px;      //definuje se proměnná px typu
                   // pointer na int

px = &x;           // přiřazuje se hodnota adresy
                   // proměnné x
```

Kterékoli proměnné typu „pointer na…“ lze přiřadit hodnotu **NULL:**

```
#define NULL 0
px = NULL;
```

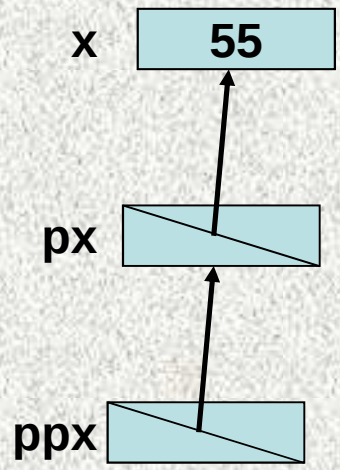
C - Ukazatel (pointer)

```
int x,y;
int *px,*py;
px=NULL;
px=&x;
x=3;
y=*px;
py=px;
*py=10;

*px =&x;
```



C - Ukazatel (pointer)

```
int x;
int *px;
int **ppx;
x=1;
px=NULL;
ppx=NULL;
px=&x;
ppx=&px;
**ppx=6;
*px=10;
x = 55;

printf(" 6 %d " "  %d " "  %d" "  \n", x, *px,  **ppx);
```



C - Ukazatel (pointer) – operace, relace, přířazení

- Povolené aritmetické operace s ukazateli:
  - pointer + integer
  - pointer - integer
  - pointer1 - pointer2 (musí být stejného typu)

- Povolené operandy relace:
  - dva ukazatele (pointers) shodného typu nebo jeden z nich NULL nebo typu void

- Přiřazení – povolený operand na pravé straně
  - pointer (stejného typu) nebo operand=NULL
  - konkrétní adresa

- *Aritmetické* operace jsou *užitečné* když ukazatel *ukazuje na pole*

C - Pole (array)

C - Pole (array):

- Pole je *množina* prvků (proměnných) *stejného typu*
- K prvkům pole se *přistupuje* pomocí pořadového čísla prvku (*indexu*)
- Index musí být *celé číslo* (konstanta, proměnná, výraz)
- Index *prvního* prvku je vždy *roven 0*
- *Prvky pole* mohou být proměnné *libovolného typu* (i strukturované)
- Pole může být *jednorozměrné* i vícerozměrné (prvky pole jsou opět pole)
- Definice pole určuje:
  - - jméno pole
  - - typ prvku pole
  - - počet prvků pole
- Prvky pole je možné *inicializovat*
- Počet prvků *statického* pole musí být znám *v době překladu*
- ***Prvky pole v „C“ zabírají v paměti souvislou oblast!***
- **Velikost pole (byte) = počet prvků pole * `sizeof (prvek pole)`**
- ***C - nemá proměnnou typu String, nahrazuje se jednorozměrným polem z prvků typu char. Poslední prvek takového pole je vždy ‘\0’ (null char)***
- ***C - nekontroluje za běhu programu, zda vypočítaný index je platný!!!***

C - Pole (array)

C - Deklarace pole (array):

```
char poleA [9]; // jednorozmerne pole z prvku char
int poleB[3][3]; // dvourozmerne pole z prvku int
Uložení v paměti
```

poleB [0][0], [0][1], [0][2], [1][0], [1][1], [1][2], [2][0], [2][1], [2][2],

C - Inicializace pole:

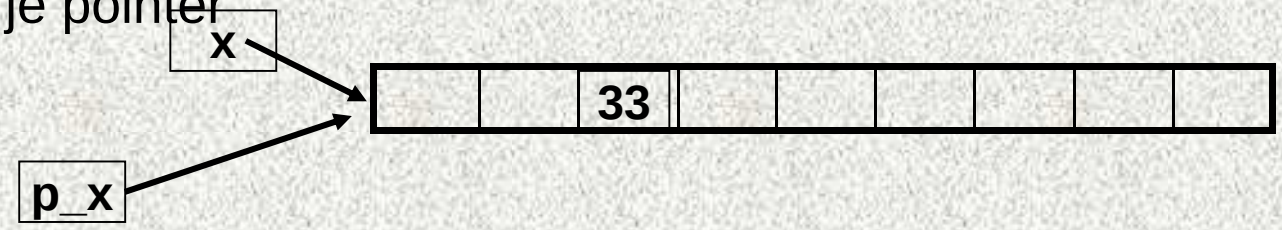
```
double x[]={0.1, 0.4, 0.5};
char s[]="abc";
char s1[]={'a','b','c','\0'}; // totez jako “abc”
int ai[3][3]={{1,2,3}, {4,5,6}, {7,8,9}};
char cmd[][10]={"Load","Save","Exit"};
                                // druhy rozmer nutny
```

C - Přístup k prvkům pole:

```
ai[1][3]=15*2;
```

C – pole a ukazatele

Identifikátor pole je pointer




```
int x[9];
x[2]=33;
x[i] ~~ obsah prvku pole i
// x ~~ pointer na počátek pole
//&x[i] ~~ adresa prvku pole - “adresa x” + i * sizeof(int)
//x[i] ~~ obsah prvku pole - *(x + i)
int *p_x;

p_x = x; // ~~ p_x = &x[0];

for (i=0;i<9;i++)x[i]=0;
for (i=0;i<9;i++)(p_x + i)=0;
```

- **Jméno pole je konstantní ukazatel na počátek pole (na prvek x[0])**

C – Pole a ukazatel (pointer)

Př:

```
int a[10],y;
                    // a[] pole typu int
int *px;
                    // px ukazatel na int
px=a;
                    // Adresa a[0] do px
px++;
                    // px ukazuje na a[1]
y=*(px+5);
                    // do y hodnotu y a[6]
px=&a[3];
                    // px ukazuje na a[3]
```

C – funkce – parametr typu pointer

C - Funkce (function) pokrač:

př:

```
void vymen(int *px, int *py); // Prototyp funkce

void main(void)
  {                              // Zacatek programu
  int a=10,b=20;
  vymen(&a,&b);
  . . . .
  }
                                 // Zamena x a y
void vymen(int *px, int *py)  // Definice funkce
  {                              // Nahrada volani odkazem
  int tmp=*px;
  *px=*py;
  *py=tmp;
 }
```

C – funkce – parametr typu pole

C - Funkce (function) pokrač:

př:

```
int secti(int a[], int n);          // Prototyp funkce

void main(void)
  {                                 // Zacatek programu
  int a[]={1,2,3,4,5,6,7,8};
  long suma;
  suma=secti(a,n);
  . . . .
  }
                                    // Secti n prvku a[]
int secti(int a[], int n)           // Definice funkce
  {
  int i;
  long suma=0;
  for(i=0;i<n;i++)
    suma += a[i];
  return(suma);
 }
```

C – Pole a ukazatel (pointer)

C - Ukazatel na pole (pointer to array) pokrač:

- V složitějších deklaracích ukazatelů mohou být nezbytné závorky

```
double a[10];
                    // Pole z prvku double
double *pa[10];
                    // Pole 10-ti ukazatelu na double
```

C - Ukazatel na char a práce s řetězci:

```
př: // Ukazatel na string
    char *s;              // Ukazatel na char
    s="Ja jsem string"; // s ukazuje na prvni znak retezce
```

C – funkce a pole jako parametr + pointer

C - Funkce (function) pokrač:

```
př:
#include <stdio.h>                    // Standardni knihovna
int porovnejString(char *s, char *p);        // Prototyp fce
char s[]="Nazdar";
char *g="Ahoj";

void main(void)
      {                               // Zacatek programu
  int vysledek;
  if((vysledek=porovnejString(s,g))==0)
    printf("Retezce se rovnaji");
  else
    printf("Retezce jsou ruzne");
      }

                                  // Porovnej retezce
int porovnejString(char *s, char *p)         // Definice funkce
                                  // Nahrada volani odkazem
{   char *ss=s;char *pp=p;
    int i; // Nahrada volani odkazem
    for (i = 0; *s++ == *p++ && *p != '\0'; );
          if (*s == '\0' && *p == '\0')return (0);
          else return (s – ss +1 );                              }
```

(if (*--s == *--p) return (0);)

JAVA - Pole – obrat pole /jiné/ (1)

```
import java.util.*; // Scanner je v knihovně java.util

public class ObratPole {
   public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("zadejte počet čísel");
    int[] pole = new int[sc.nextInt()];
    System.out.println("zadejte "+pole.length+" čísel");
    for(int i=0; i < pole.length; i++)
          pole[i] = sc.nextInt();
    System.out.println("výpis čísel v obráceném pořadí");
    for(int i=pole.length-1; i >= 0; i--)
          System.out.println(pole[i]);
   }
}
```

Dynamické pole

1

2

3

Neošetřené chyby

Základní znalost

C - Pole – obrat pole /jiné/ (1)

```
int main(int argc, char** argv) {
    #define MAX_DELKA 20
    int pole[MAX_DELKA]; int n, i;
    printf(" Zadejte pocet cisel = ");
    if (!nextInt(&n)) {
        printf(" Chyba - Zadany udaj neni cislo\n\n");
        exit(EXIT_FAILURE);
    }
    if(n > MAX_DELKA){
        printf("\n Chyba - max pocet cisel = %d \n\n",MAX_DELKA);
            exit(EXIT_FAILURE);
    }
    printf("\n Zadejte cela cisla (kazde ukoncit ENTER)\n\n");
    for (i = 0; i < n; i++) {
        if (!nextInt(&pole[i])) {
            printf("Chyba - Zadany udaj neni cislo\n\n");
            exit(EXIT_FAILURE);
        }
    }
    printf("\n Vypis cisel v obracenem poradi \n\n");
    for (i = n - 1; i >= 0; i--) {
        printf("pole[%d] = %d \n", i, pole[
    }
    printf(" \n");return
}
```

Statické pole

1

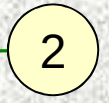
2

3

Ošetřené chyby

Základní znalost

C - Pole – obrat pole /jiné/ (1)

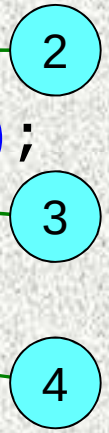



```
int nextInt(int *cislo){
// === Bezpecne pro libovolny zadany pocet znaku ===
    // Navratova hodnota:
    // TRUE  - zadano cele cislo
    // FALSE - neplatny vstup
    enum boolean {FALSE,TRUE};
    const int BUF_SIZE = 80;
    char vstup[BUF_SIZE],smeti[BUF_SIZE];
    fgets(vstup,sizeof(vstup),stdin);
    if(sscanf(vstup,"%i%[^\n]",cislo,smeti) != 1)
        return(FALSE); //špatný počet parametrů
    return(TRUE);
}
```

JAVA - pole a funkce /jiné/ (2)

- Odkaz na pole může být parametrem funkce i jejím výsledkem

```
import java.util.*;

public class ObratPole {
  public static void main(String[] args) {
    int[] vstupniPole = ctiPole();
    int[] vystupniPole = obratPole(vstupniPole);
    vypisPole(vystupiPole);
  }

  static int[] ctiPole() { ... }
  static int[] obratPole(int[] pole) { ... }
  static void vypisPole(int[] pole) { ... }

}// class ObratPole END
```



JAVA - pole a funkce /jiné/ (2)

```
static int[] ctiPole() {
   Scanner sc = new Scanner(System.in);
   Sys.pln("zadejte počet čísel");
   int[] pole = new int [sc.nextInt()];
   System.out.println("zadejte "+pole.length+"
čísel");
   for (int i=0; i<pole.length; i++)
      pole[i] = sc.nextInt();
   return pole;
}
static int[] obratPole(int[] pole) {
   int[] novePole = new int [pole.length];
   for (int i=0; i<pole.length; i++)
      novePole[i] = pole[pole.length-1-i];
   return novePole;
}
static void vypisPole(int[] pole) {
   for (int i=0; i<pole.length; i++)
      System.out.println(pole[i]);
}
```

Nové pole

1

5

6

7

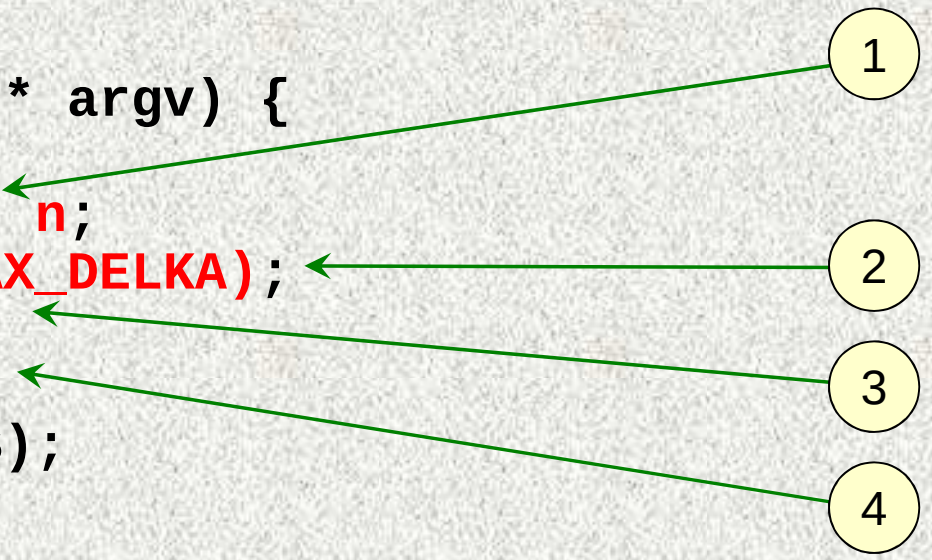
8

9

10

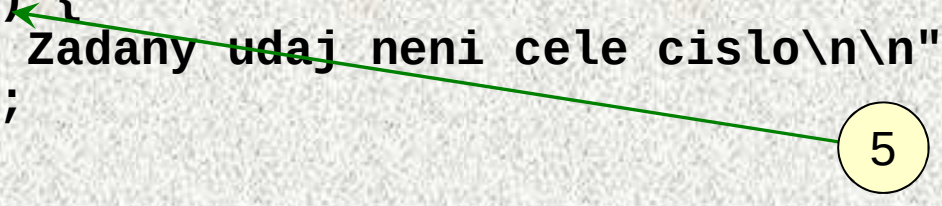
C - pole a funkce /jiné/ (2)

```
#include <stdio.h>
#include <stdlib.h>

// Function prototypes
int ctiPole (int p[], int max_delka);
void obratPole (int p[], int delka_pole);
void vypisPole (int p[], int delka_pole);
int nextInt(int *cislo);

int main(int argc, char** argv) {
    #define MAX_DELKA 20
    int pole[MAX_DELKA], n;
    n = ctiPole(pole, MAX_DELKA);
    obratPole(pole, n);
    vypisPole(pole, n);
    return (EXIT_SUCCESS);
}
```



C - pole a funkce /jiné/ (2)

```
int ctiPole (int p[], int max_delka){
    // Pole p[] se predava odkazem
    int n, i;
    printf(" Zadejte pocet cisel = ");
    if (!nextInt(&n)) {
        printf("\n Chyba - Zadany udaj neni cele cislo\n\n");
        exit(EXIT_FAILURE);
    }
    if(n <1 || n > max_delka){
        printf("\n Chyba-pocet cisel=<1,%d> \n\n",MAX_DELKA);
        exit(EXIT_FAILURE);
    }
    printf("\n Zadejte cela cisla (kazde ukoncit ENTER)\n\n");
    for (i = 0; i < n; i++) {
        if (!nextInt(&p[i])) {
         printf("\n Chyba - Zadany udaj neni cele cislo\n\n");
         exit(EXIT_FAILURE);
        }
    }
    return(n);
}
```



C - pole a funkce /jiné/ (2)

Volání odkazem

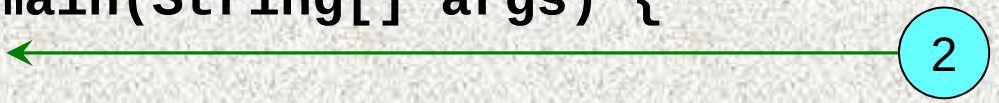
Původní pole

```
void obratPole (int p[], int delka_pole){
    // Pole p[] se predava odkazem (call by reference)
    int i, x, nPul;
    nPul= delka_pole/2;
    for(i=0; i < nPul; i++ ){
        x = p[i];
        p[i] = p[delka_pole-i-1];
        p[delka_pole-i-1] = x;
    }
}

void vypisPole (int p[], int delka_pole){
    // Pole p[] se predava odkazem (call by reference)
    int i;
    printf("\n Vypis cisel v obracenem poradi \n\n");
    for(i=0; i < delka_pole; i++){
        printf("pole[%d] = %d \n", i, p[i]);
    }
    printf(" \n");
}

int nextInt(int *cislo){. . .} // viz (1)
```

6 8 7 9 10

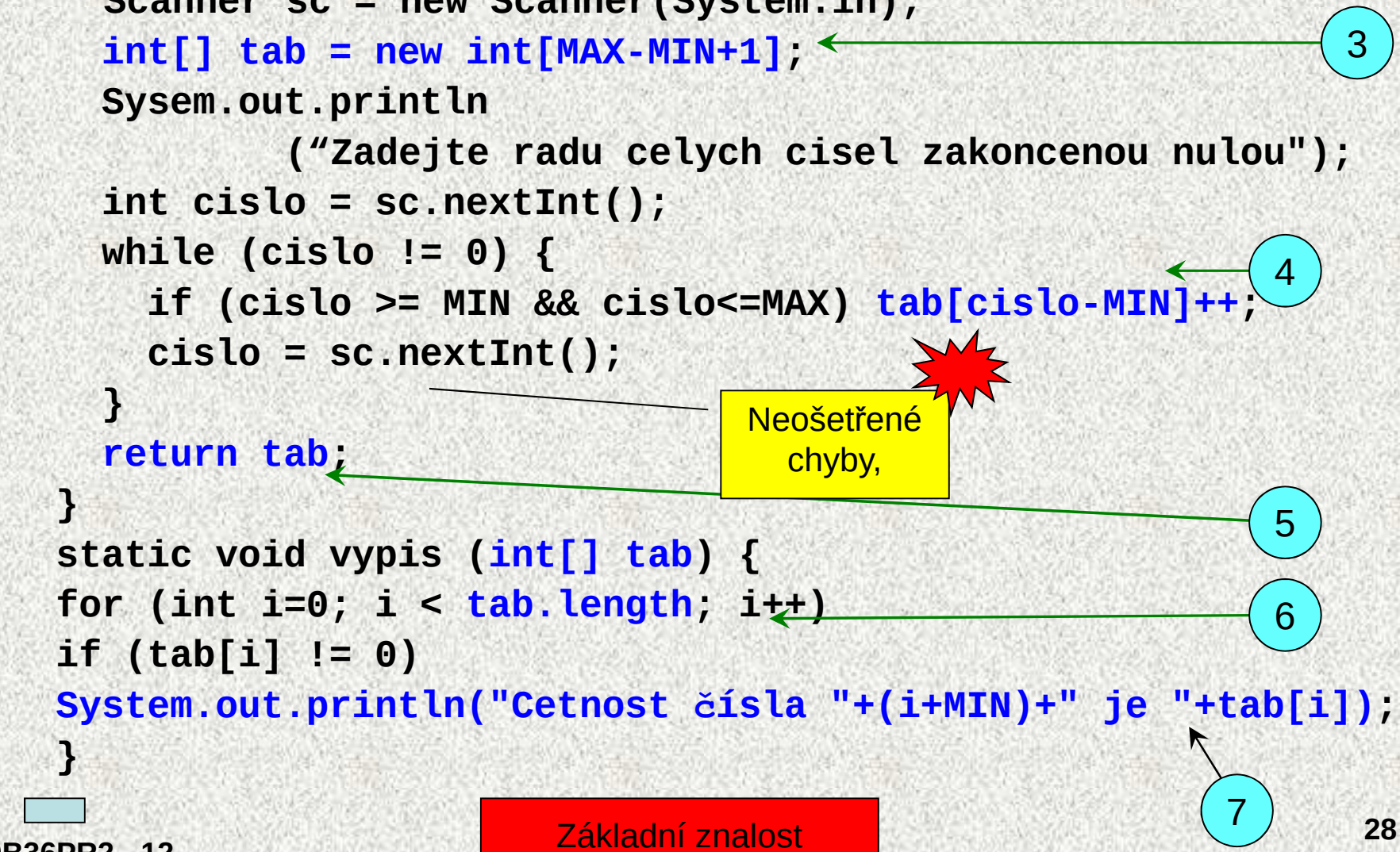
JAVA – pole – tabulka četnosti /jiné/ (3)

```
import java.util.*;

public class CetnostCisel {

  final static int MIN = 1;
  final static int MAX = 100;

  public static void main(String[] args) {
    vypis(tabulka());
  }

  static int[] tabulka() {
    ...
  }

  static void vypis(int[] tab) {
   ...
  }
}
```



JAVA – pole – tabulka četnosti /jiné/ (3)

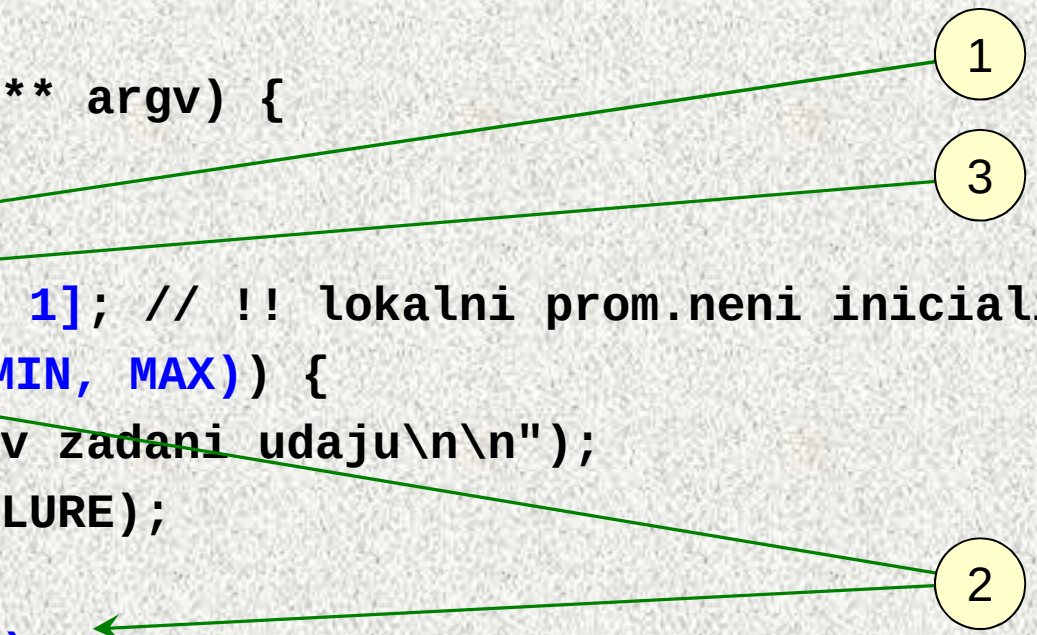
```
static int[] tabulka () {
  Scanner sc = new Scanner(System.in);
  int[] tab = new int[MAX-MIN+1];
  Sysem.out.println
          (“Zadejte radu celych cisel zakoncenou nulou");
  int cislo = sc.nextInt();
  while (cislo != 0) {
    if (cislo >= MIN && cislo<=MAX) tab[cislo-MIN]++;
    cislo = sc.nextInt();
  }
  return tab;
}
static void vypis (int[] tab) {
for (int i=0; i < tab.length; i++)
if (tab[i] != 0)
System.out.println("Cetnost čísla "+(i+MIN)+" je "+tab[i]);
}
```



Základní znalost

C – pole – tabulka četnosti /jiné/ (3)

```
#include <stdio.h>
#include <stdlib.h>
int tabulka(int t[], int min, int max);
void vypis (int t[], int min, int max);
int nextInt(int *cislo);

int main(int argc, char** argv) {
    #define MIN 1
    #define MAX 100
    int tab[MAX - MIN + 1]; // !! lokalni prom.neni inicializ.
     if (!tabulka(tab, MIN, MAX)) {
         printf(" Chyba v zadani udaju\n\n");
         return(EXIT_FAILURE);
    }
    vypis(tab, MIN, MAX);
    return (EXIT_SUCCESS);
}
```



C – pole – tabulka četnosti /jiné/ (3)

```
int tabulka (int t[], int min, int max) {
    enum boolean {FALSE,TRUE};
   int i, cislo;
    for (i = 0; i <= (max - min); i++){
        t[i] = 0;
    }
    printf(" Zadejte radu celych cisel zakoncenou nulou \n\n");
    printf(" Povoleny rozsah cisel je <%d, %d> \n\n", min, max);
    do {
        if (!nextInt(&cislo)) {
            printf("\n Chyba - Zadany udaj neni cele cislo\n");
            return (FALSE); // FALSE -> chyba
        }
        if ((cislo < MIN || cislo > MAX) && cislo != 0) {
            printf("\n Chyba - Zadane cislo je mimo rozsah\n");
            return (FALSE); // FALSE -> chyba
        }
        if (cislo == 0)break;
        t[cislo - MIN]++;
    } while (TRUE);
    return (TRUE); // Zadani bez chyby
}
```

5

4

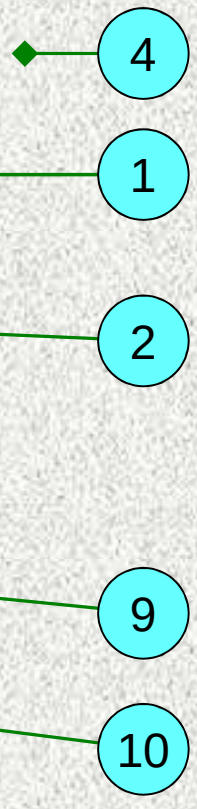
Základní znalost

C – pole – tabulka četnosti /jiné/ (3)

```
void vypis (int t[], int min, int max){
    int i;
    printf("\n");
    for(i=0; i<=(max-min+1); i++){
        if(t[i] != 0){
            printf("Cetnost cisla %3d je %3d\n",(i+MIN), t[i]);
        }
    }
    printf("\n");
}

int nextInt(int *cislo){ // viz (1)
. . .
}
```

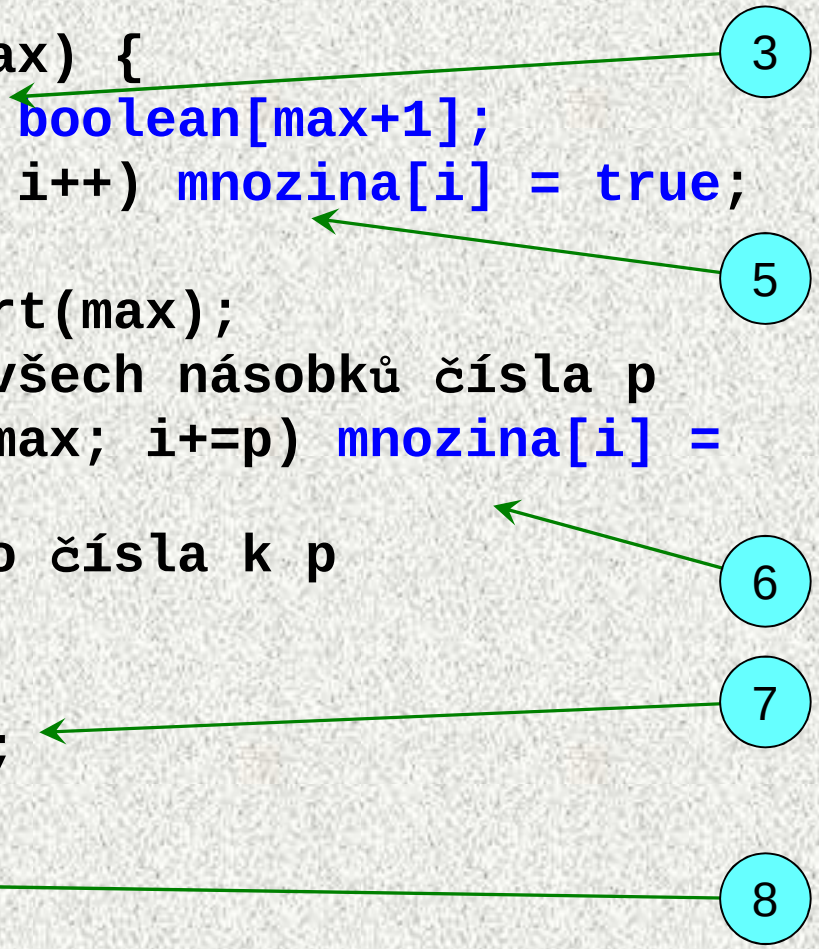
JAVA – pole - Eratosthenovo síto /jiné/ (4)

```
import java.util.*;
public class EratosthenovoSito {

    public static void main(String[] args) {
       Scanner sc = new Scanner(System.in);
       System.out.println("Zadejte max");
       int max = sc.nextInt();
       boolean[] mnozina = sito(max);
       System.out.println("Prvočísla od 2 do "+max);
       vypis(mnozina);
     }

    static void vypis(boolean[] mnozina) {
       for (int i=2; i < mnozina.length; i++)
         if (mnozina[i]) System.out.println(i);
     }

    static boolean[] sito(int max) {. . .}
}
```



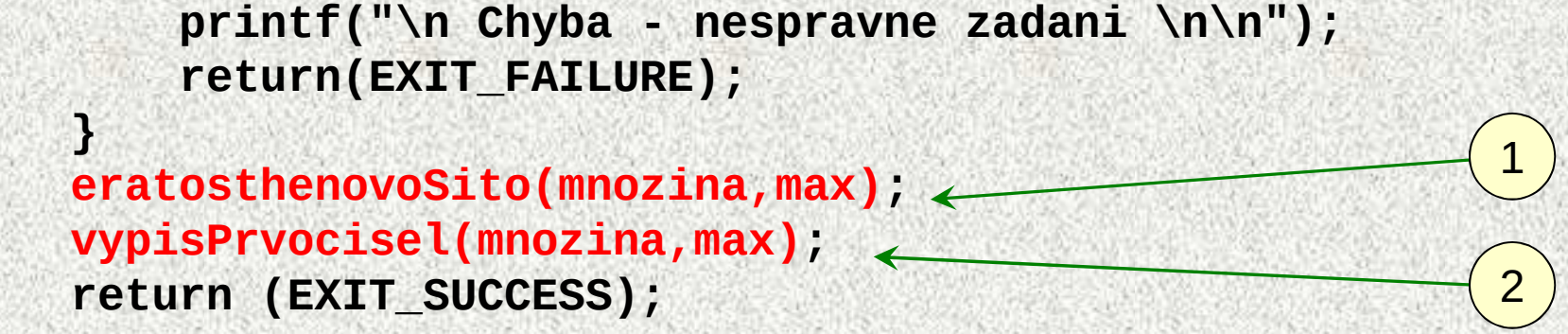
JAVA – pole - Eratosthenovo síto /jiné/ (4)

```
static boolean[] sito(int max) {
    boolean[] mnozina = new boolean[max+1];
    for (int i=2; i <= max; i++) mnozina[i] = true;
    int p = 2;
    int pmax = (int)Math.sqrt(max);
    do {      // vypuštění všech násobků čísla p
      for (int i=p+p; i <= max; i+=p) mnozina[i] =
  false;
      // hledání nejbližšího čísla k p
      do {
        p++;
      } while (!mnozina[p]);
    } while (p <= pmax);
    return mnozina;
  }
```



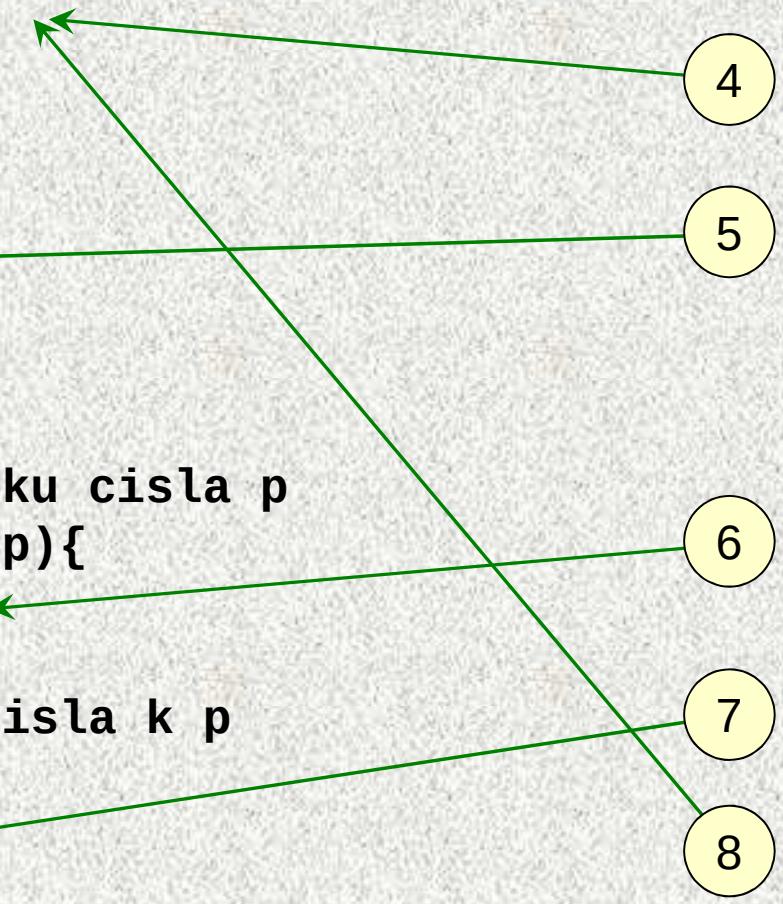
C – pole - Eratosthenovo síto /jiné/ (4)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void eratosthenovoSito (int mnozina[], int max);
void vypisPrvocisel(int mnozina[], int max);
int nextInt(int *cislo);
int main(int argc, char** argv) {
    #define MAX_LIMIT_PRO_HLEDANI_PRVOCISLA 100
    int max; int mnozina[MAX_LIMIT_PRO_HLEDANI_PRVOCISLA];
       printf("Zadejte limit pro hledani prvocisel <2, %d> =
  ", MAX_LIMIT_PRO_HLEDANI_PRVOCISLA);
    if(!nextInt(&max) || max > MAX_LIMIT_PRO_HLEDANI_PRVOCISLA
                        || max < 2){
        printf("\n Chyba - nespravne zadani \n\n");
        return(EXIT_FAILURE);
    }
    eratosthenovoSito(mnozina,max);
    vypisPrvocisel(mnozina,max);
    return (EXIT_SUCCESS);
}
```



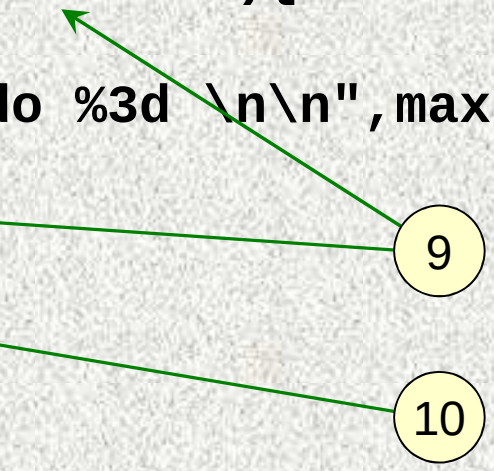
C – pole - Eratosthenovo síto /jiné/ (4)

```
void eratosthenovoSito (int mnozina[], int max){
    const int TRUE = 1;
    const int FALSE = 0;
    int i, p=2, pmax;
    for(i=2; I <= max; i++){
        mnozina[i]= TRUE;
    }
    pmax = (int)sqrt(max);
    do{
        // Vypusteni vsech nasobku cisla p
        for(i=p+p; i <= max; i+=p){
            mnozina[i] = FALSE;
        }
        // Hledani nejblizsiho cisla k p
        do{
           p++;
        }while(!mnozina[p]);
    }while(p <= pmax);
}
```



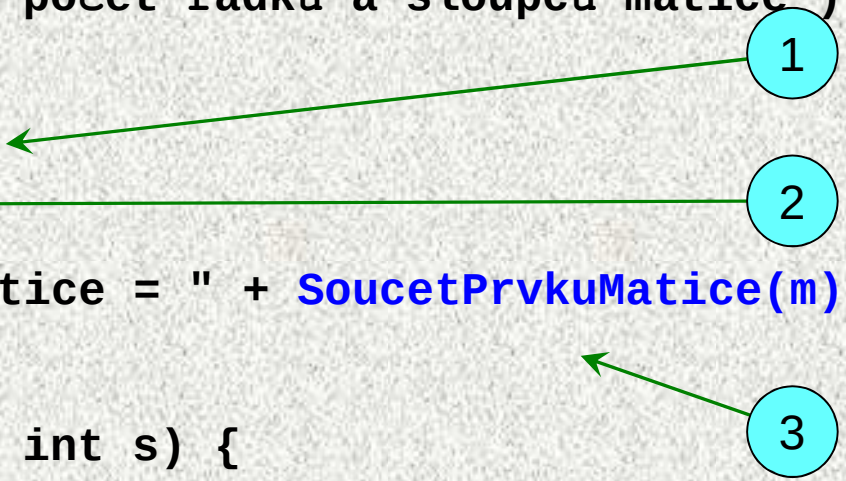
C – pole - Eratosthenovo síto /jiné/ (4)

```
void vypisPrvocisel(int mnozina[], int max){
    int i;
    printf("\n Nalezena prvocisla do %3d \n\n",max);
    for(i=2; i <= max; i++){
        if(!mnozina[i]){
            printf(" %4d \n", i);
        }
    }
    printf("\n");
}
```



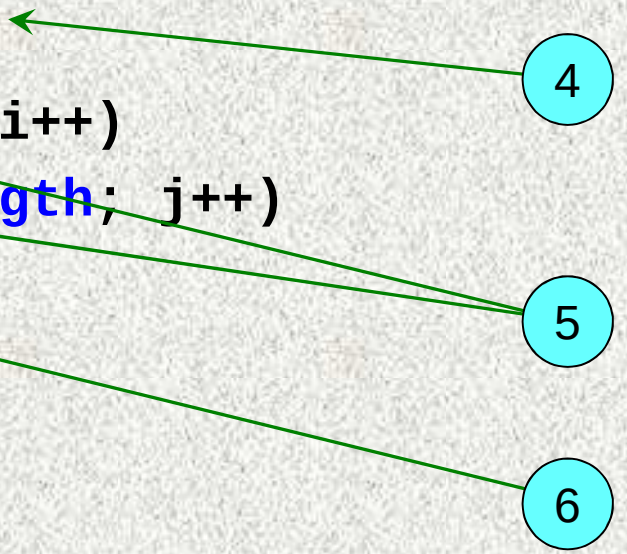
JAVA – vícerozměrné pole - součet prvků /podobné/ (5)

```
import java.util.*;
public class Matice {
   static Scanner sc = new Scanner(System.in);
  public static void main(String[] args) {
    System.out.println("zadejte počet řádků a sloupců matice");
    int r = sc.nextInt();
    int s = sc.nextInt();
    int[][] m = ctiMatici(r,s);
     VypisPrvkuMatice(m);
    System.out.println
              ("Součet prvku matice = " + SoucetPrvkuMatice(m));
  }

static int[][] ctiMatici(int r, int s) {
    int[][] m = new int[r][s];
    System.out.println("zadejte celočíslenou matici "+r+"x"+s);
    for (int i=0; i<r; i++)
      for (int j=0; j<s; j++)
        m[i][j] = sc.nextInt();
    return(m);
  }
```



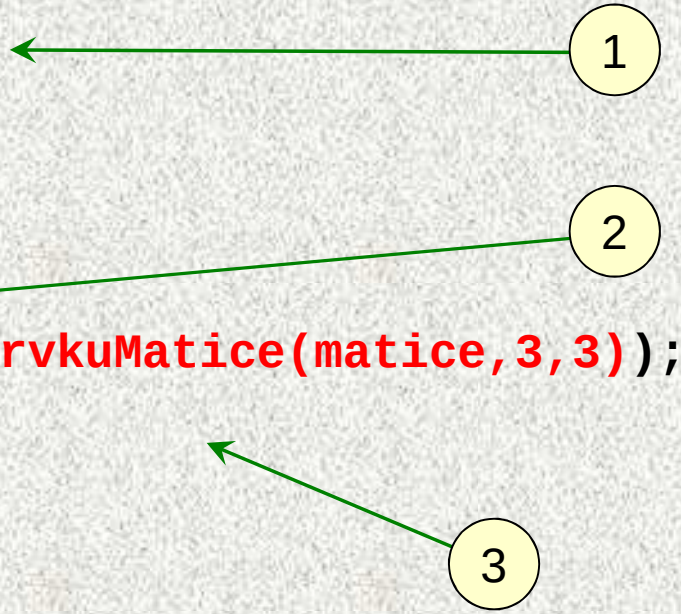
JAVA –vícerozměrné pole-součet prvků /podobné/ (5)

```
static int soucetPrvkuMatice (int[][] m) {
    int suma = 0;
    for (int i=0; i < m.length; i++)
      for (int j=0; j < m[0].length; j++)
        suma = suma + m[i][j];
    return(suma);
  }

static void vypisMatice(int[][] m) {
    for (int i=0; i < m.length; i++) {
      for (int j=0; j < m[i].length; j++)
        System.out.print(m[i][j]+" ");
      System.out.println();
    }
  }
```



C - vícerozměrné pole - součet prvků /podobné/ (5)

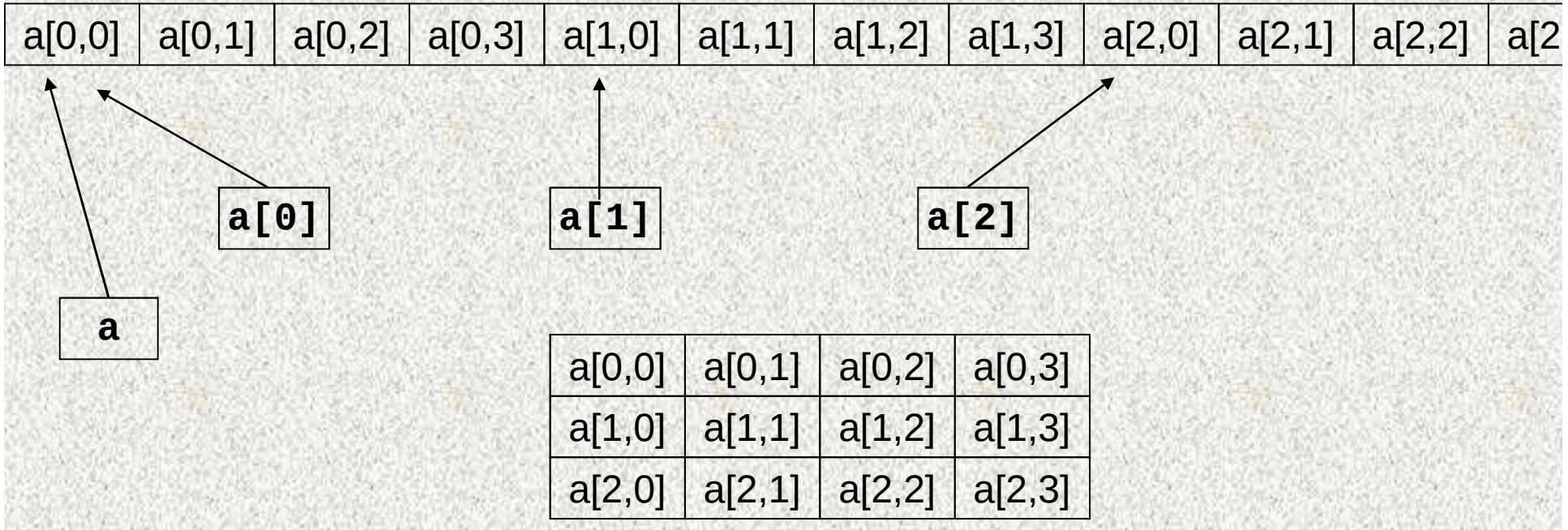
```
#include <stdio.h>
#include <stdlib.h>
#define MAX_J 3

void vypisMatice (int mat[][], int m, int n);
int soucetPrvkuMatice (int mat[][MAX_J], int m, int n);

int main(int argc, char** argv) {
int matice[][MAX_J]={{1, 2, 3},
                     {3, 4, 5},
                     {6, 7, 8},
                     {9,10,11}};
printf(" Matice \n")
vypisMatice(matice,3,3);
printf("Ma soucet prvku=%d\n",soucetPrvkuMatice(matice,3,3));
return (EXIT_SUCCESS);
  }
```



C- vícerozměrné pole

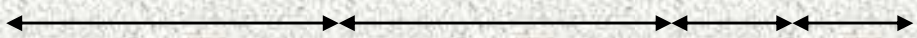
Podobně jako v jazyku Java se s vícerozměrnými poli pracuje jako s poli, jejichž prvky jsou opět pole

```
int a[3][4];
```



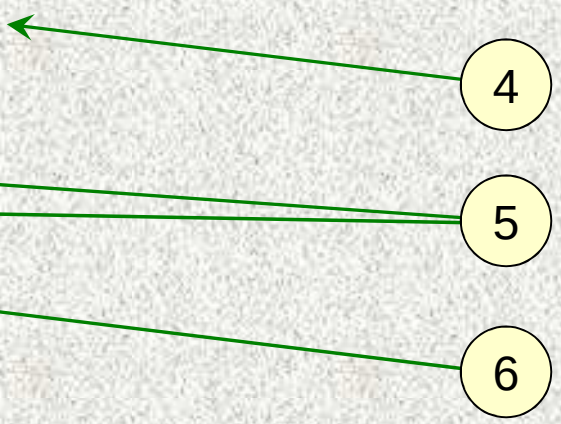
a[0,0]	a[0,1]	a[0,2]	a[0,3]
a[1,0]	a[1,1]	a[1,2]	a[1,3]
a[2,0]	a[2,1]	a[2,2]	a[2,3]

```
int a[2][1];
```



C - vícerozměrné pole - součet prvků /podobné/ (5)

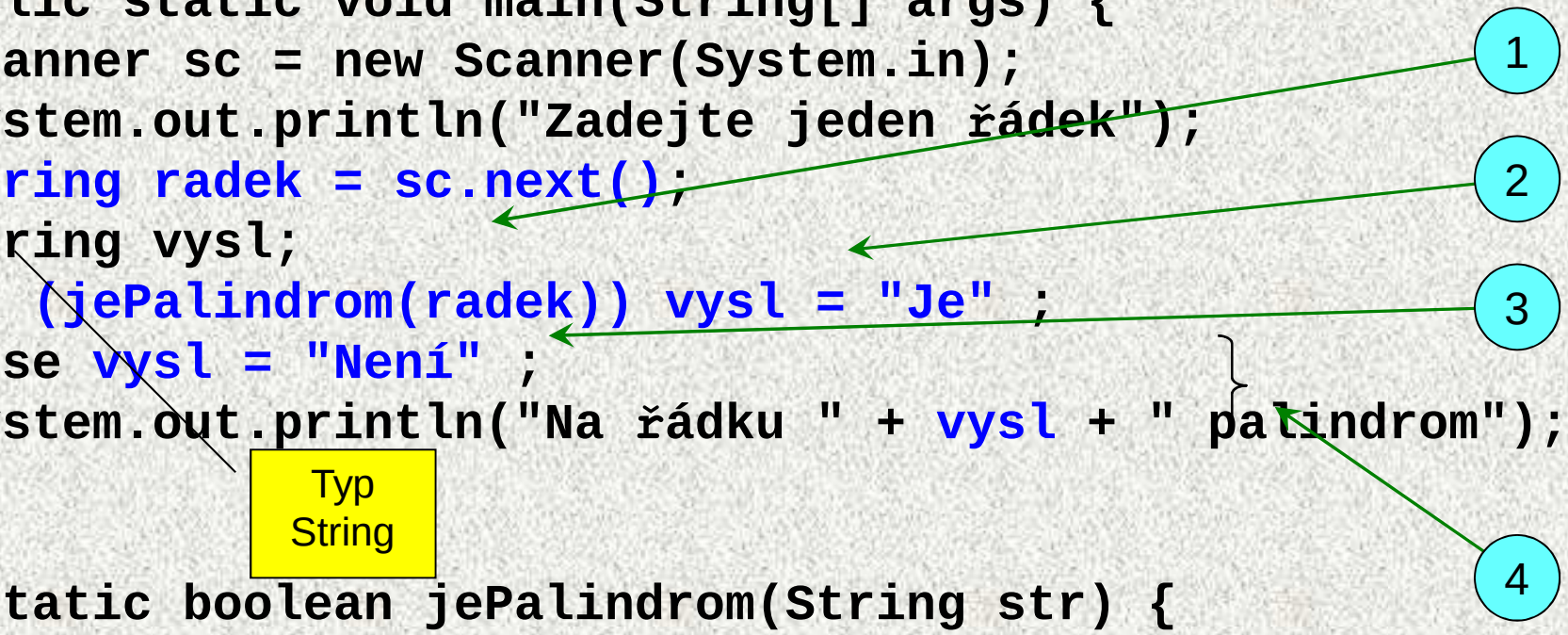
```
int soucetPrvkuMatice (int mat[][MAX_J], int m, int n){
    int suma = 0;
    int i,j;
    for(i=0; i < m; i++){
        for(j=0; j < n; j++){
            suma = suma + mat[i][j];
        }
    }
    return(suma);
}
void vypisMatice (int mat[][MAX_J], int m, int n){
    int i, j;
       for(i=0; i < m; i++){
        for(j=0; j < n; j++){
            printf(" m[%d,%d] =%3d   ",i,j,mat[i][j]);
        }
     }
    printf("\n");
}
```



Požadovaná znalost

JAVA – string – palindrom /jiné/ (6)

```
import java.util.*;
public class Palindrom {
public static void main(String[] args) {
 Scanner sc = new Scanner(System.in);
 System.out.println("Zadejte jeden řádek");
 String radek = sc.next();
 String vysl;
 if (jePalindrom(radek)) vysl = "Je" ;
 else vysl = "Není" ;
 System.out.println("Na řádku " + vysl + " palindrom");
  }

  static boolean jePalindrom(String str) {
    ...
  }
}
```



JAVA – string – palindrom /jiné/ (6)

```
static boolean jePalindrom(String str) {
    int i = 0, j = str.length()-1;
    while (i < j) {
      while (str.charAt(i) == ' ') i++;
      while (str.charAt(j) == ' ') j--;
      if (str.charAt(i) != str.charAt(j))return false;
      i++; j--;
    }
    return true;
  }
```

Délka Stringu

Znak ve Stringu

5

6

7

8

C - string – palindrom /jiné/ (6)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
int jePalindrom(char str[]);
int nextLine(int str[]);

int main(int argc, char** argv) {
    char string[80];
    printf(" Zadejte jeden radek textu \n\n");
    printf(" String = ");
    if (!nextLine(string)) {
        printf("Chybne zadani\n\n");
        return (EXIT_FAILURE);
    }
    if (jePalindrom(string)) {
        printf("\n String <%s> je palindrom \n\n", string);
    } else {
        printf("\n String <%s> neni palindrom \n\n", string);
    }
    return (EXIT_SUCCESS);
}
```

Pole znaků

1

2

3

4

Základní znalost

C - string – palindrom /jiné/ (6)

```
int jePalindrom(char str[]) {
    enum boolean {FALSE,TRUE};
    int i = 0, j = strlen(str) - 1;
    while (i < j) {
        while (str[i] == ' ')i++;
        while (str[j] == ' ')j--;
        if (toupper(str[i]) != toupper(str[j]))
            return (FALSE);
        i++;
        j--;
    }
    return (TRUE);
}

int nextLine(int str[]){
      gets(str);
    return(TRUE);
}
```

Délka stringu

Znak ve stringu

5

6

7

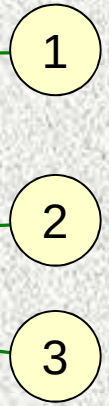
8

2

Základní znalost

C – pole a ukazatele /jiné/ (7)

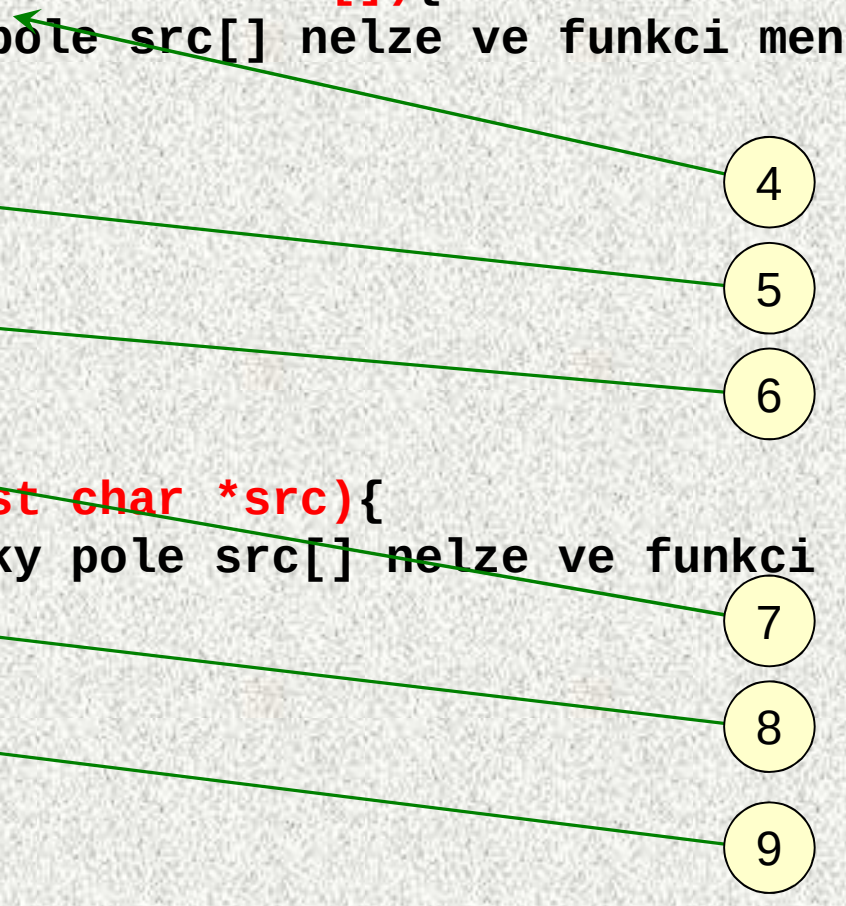
```
#include <stdio.h>
#include <stdlib.h>
void copyArray1(char dst[], const char src[]);
void copyArray2(char *dst, const char *src);
void copyArray3(char *dst, const char *src);
void copyArray4(char dst[], const char src[]);
int main(int argc, char** argv) {
    char a1[] = " Test 1"; char a2[] = " Test 2";
    char a3[] = " Test 3"; char a4[] = " Test 4";
    char b[80];
   printf("a[i] je ekvivaletni *(a_ptr+i) \n");
   printf("%s \n\n", a1);
    copyArray1(b,a1); printf("%s \n\n",b);
    printf("%s \n\n", a2);
    copyArray2(b,a2); printf("%s \n\n",b);
    printf("%s \n\n", a3);
    copyArray3(b,a3); printf("%s \n\n",b);
    printf("%s \n\n", a4);
    copyArray4(b,a4); printf("%s \n\n",b);
    return (EXIT_SUCCESS);
}
```



Požadovaná znalost

C – pole a ukazatele /jiné/ (7)

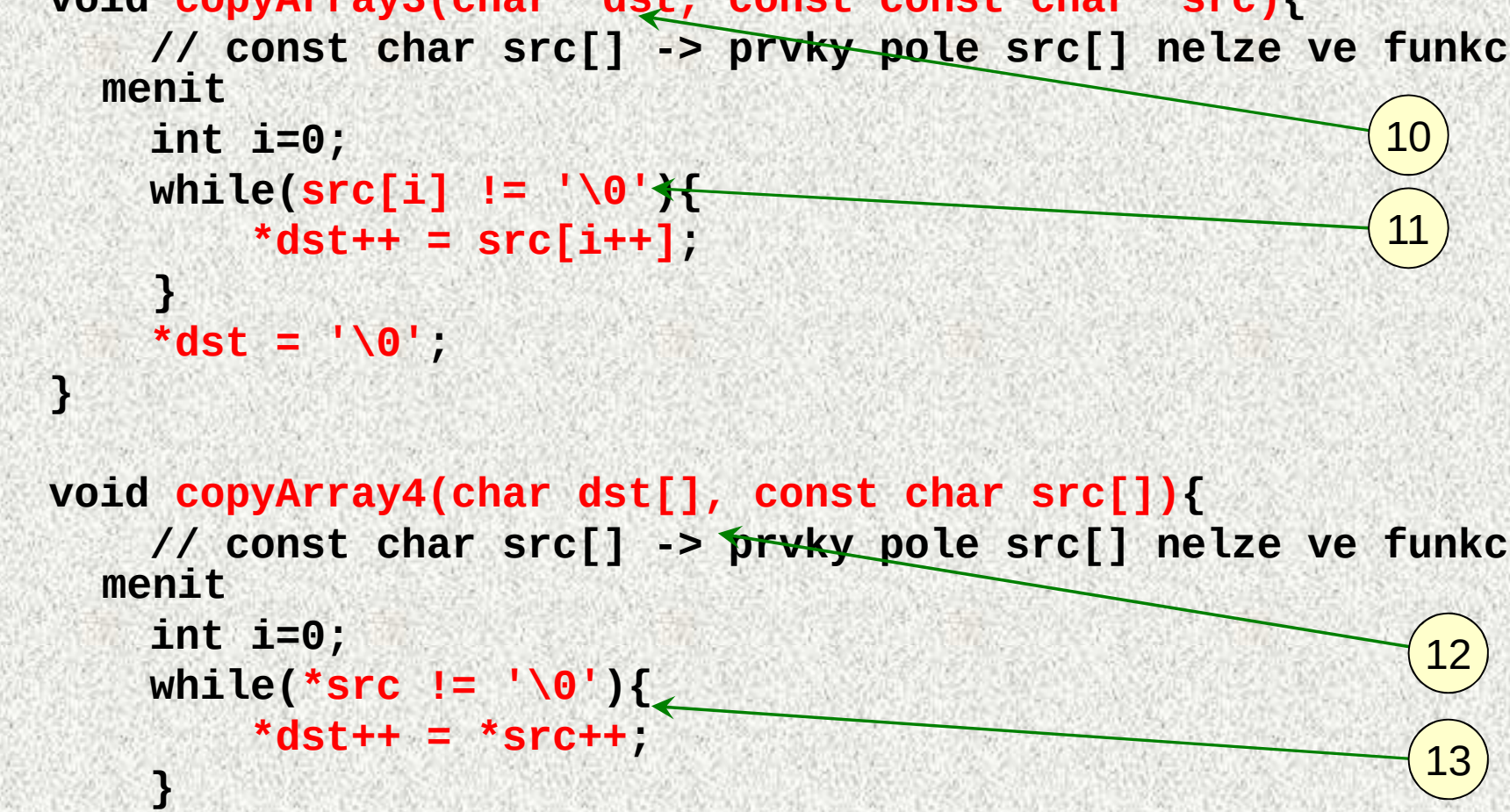
```
void copyArray1(char dst[], const char src[]){
 // const char src[] -> prvky pole src[] nelze ve funkci menit
    int i=0;
    while(src[i] != '\0'){
        dst[i] = src[i++];
    }
    dst[i] = '\0';
}

void copyArray2(char *dst, const char *src){
    // const char src[] -> prvky pole src[] nelze ve funkci
   menit
    while(*src != '\0'){
        *dst++ = *src++;
    }
    *dst = '\0';
}
```



Požadovaná znalost

C – pole a ukazatele /jiné/ (7)

```
void copyArray3(char *dst, const const char *src){
    // const char src[] -> prvky pole src[] nelze ve funkci
  menit
    int i=0;
    while(src[i] != '\0'){
        *dst++ = src[i++];
    }
    *dst = '\0';
}

void copyArray4(char dst[], const char src[]){
    // const char src[] -> prvky pole src[] nelze ve funkci
  menit
    int i=0;
    while(*src != '\0'){
        *dst++ = *src++;
    }
    *dst = '\0';
}
```



Požadovaná znalost

C – použití pointerů u funkcí

Pointer se používá

- jako způsob předávání parametrů odkazem
- jako způsob pro genericitu typů – typu **`void`** – jako pointer na různé typy
- ukazatel na funkci
- ukazatel umožní, aby funkce byla parametrem funkce

Příklad na pointer na funkci

```
double funkce1(double x);     // Prototyp funkce
double (*pFnc)(double x);
            // Ukazatel na funkci double  s parametrem
            // double
int max=200;
pFnc=funkce1;
            // Adresa funkce1 do ukazatele pFnc
(*pFnc)(max);
            // Volani funkce1 pomoci ukazatele



double funkce1(double x)
{
return (sin(x) * x + 8);
}
```

Příklad na pointer na funkci

```
double funkceprovypis (double x,
                                double(*funkce)(double n))
{
    return ((*funkce)(x));
}
 main…
  p[0]=pol1;
  p[1]=pol2;
  p[2]=sin;
  p[3]=cos;

printf(" funkce %4.2f \n",
                    funkceprovypis(0.4,(*(p + 2))));
```

C – funkce a ukazatele /jiné/

```
#define DOLNI  (-1)
#define HORNI  1
#define KROK   0.2
double pol1(double x){return (x * x + 8);}
double pol2(double x){return (x * x * x - 3);}
int main(void){
  int i;double x;
  double (*p[4])();
         /* pole pointerů na funkce vracející double */
  p[0]=pol1;
  p[1]=pol2;
  p[2]=sin;
  p[3]=cos;
  for (i = 0;  i < 4;  i++) {
   printf("%d\n", i);
    for (x = DOLNI;  x <= HORNI;  x += KROK)
      printf("%5.1f %8.3f\n", x, p[i](x));
    putchar('\n');           /* odradkovani */ }
    return (EXIT_SUCCESS);
}
```

C – dynamická alokace pole

Jak se definuje pole v Javě?

Existuje pojem statického a dynamického pole v Javě?

Jak se v Javě uvolňuje prostor dynamického pole ?

Statické pole

```
int a[10];    //  statické pole, a je konstantní pointer na začátek pole
              // nedá se měnit
int *pa;      // pointer, alokuje pamět jen pro sebe, nealokuje pole
pa=a;         // pointer na statické pole
```

C – dynamická alokace pole

Jiná možnost – dynamické pole – analogie generování pole pomocí **new** v Javě

Přidělení bloku paměti při výpočtu funkcí z heapu a vrací pointer na počátek

```
void* malloc(počet bytes);
```

Obvykle se okamžitě přetypuje na typ pole, pro které se paměť vyhrazuje

```
int *pa;
pa=(int*)malloc(10*sizeof(int));
```

Práce s dynamickým polem

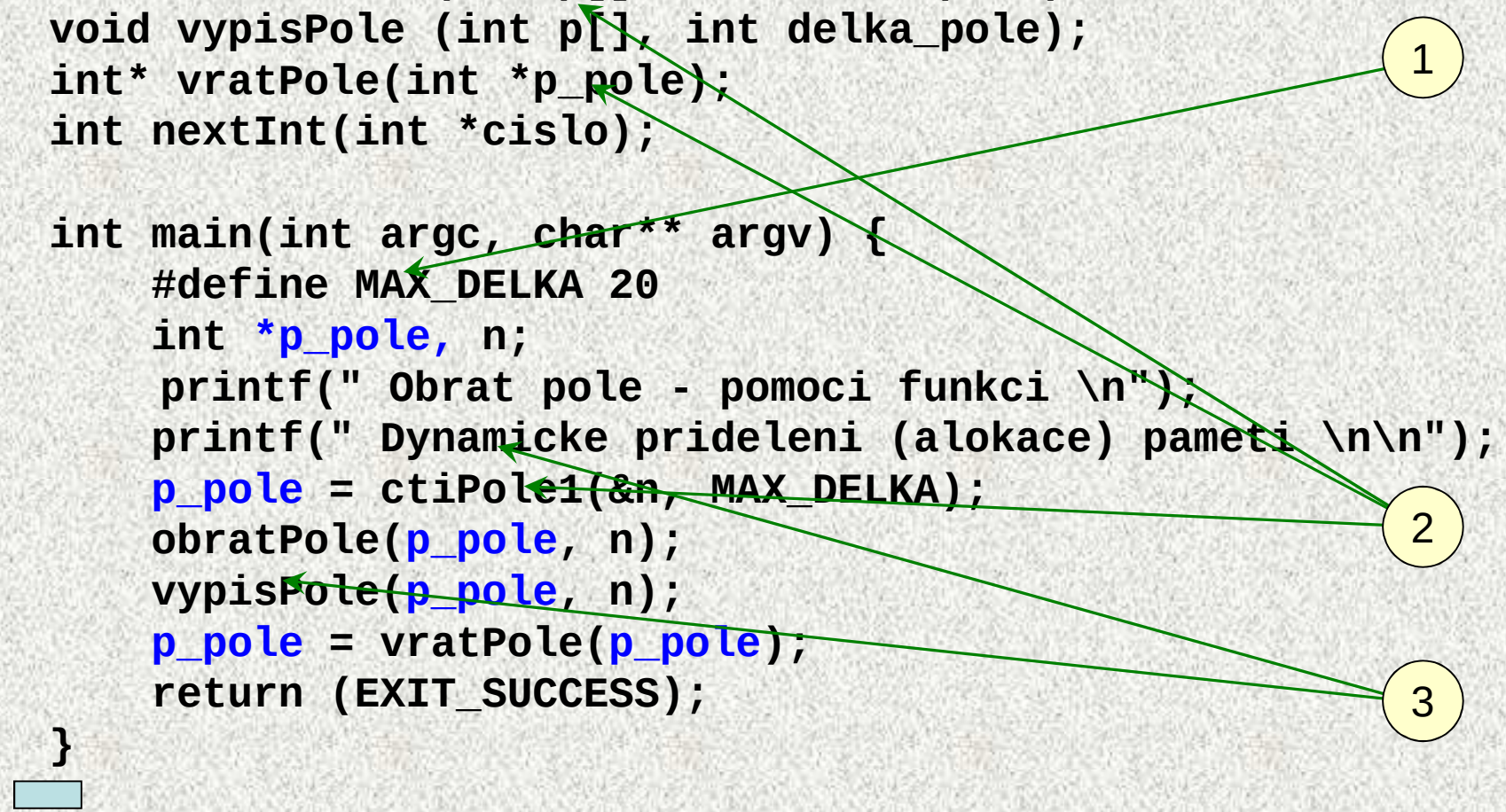
- Totožná jak se statickým
- Konvence přístup do statického pole **a[i],** dynamického ***(pa + i)**

Uvolnění paměti do heapu:

```
void free(pointer_na_paměť);
free(pa);
pa = NULL;
```

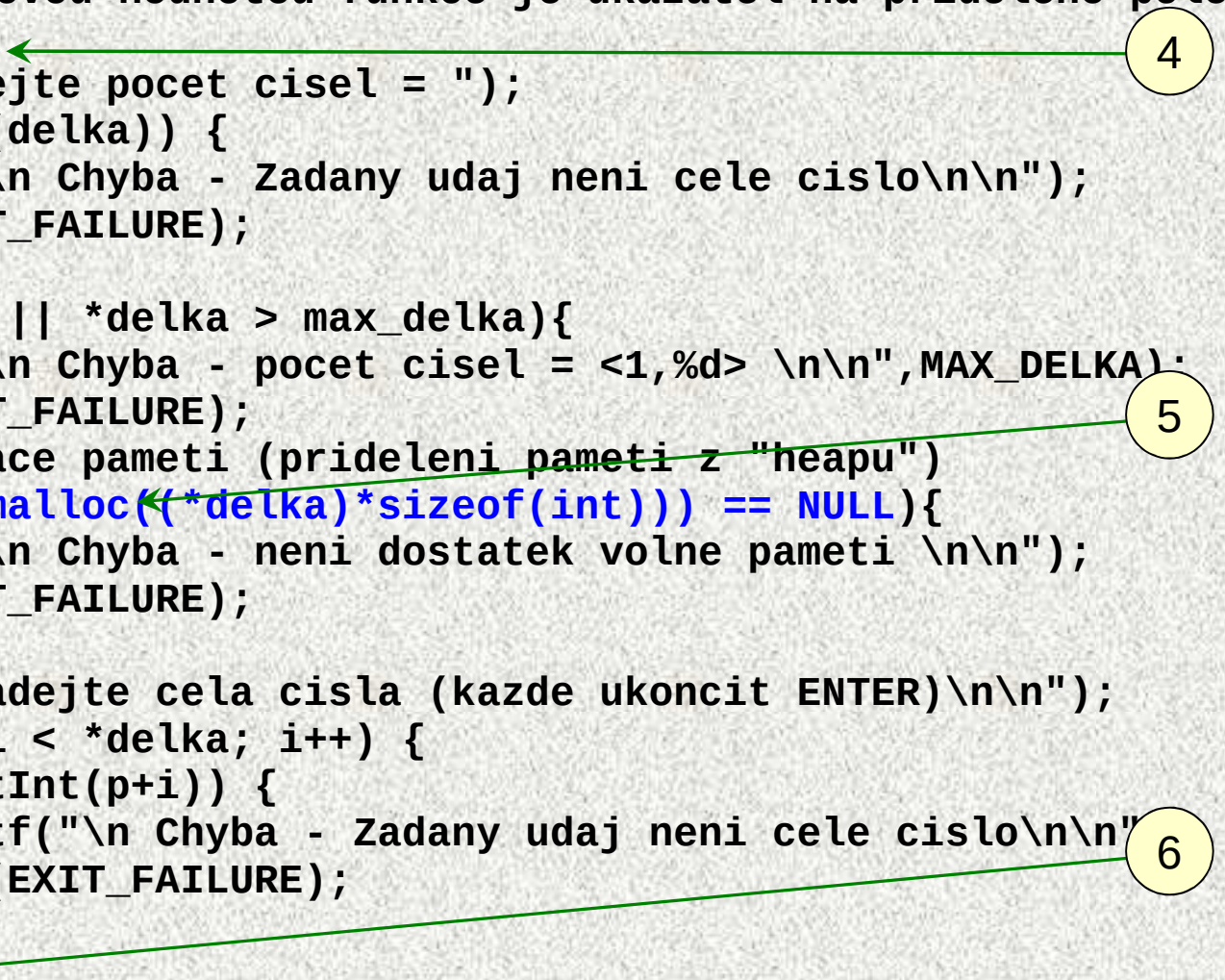
C – dynamická alokace pole a ukazatele /jiné/ (8)

```
#include <stdio.h>
#include <stdlib.h>

int* ctiPole1 (int *delka, int max_delka);
void obratPole (int p[], int delka_pole);
void vypisPole (int p[], int delka_pole);
int* vratPole(int *p_pole);
int nextInt(int *cislo);

int main(int argc, char** argv) {
    #define MAX_DELKA 20
    int *p_pole, n;
     printf(" Obrat pole - pomoci funkci \n");
    printf(" Dynamicke prideleni (alokace) pameti \n\n");
    p_pole = ctiPole1(&n, MAX_DELKA);
    obratPole(p_pole, n);
    vypisPole(p_pole, n);
    p_pole = vratPole(p_pole);
    return (EXIT_SUCCESS);
}
```



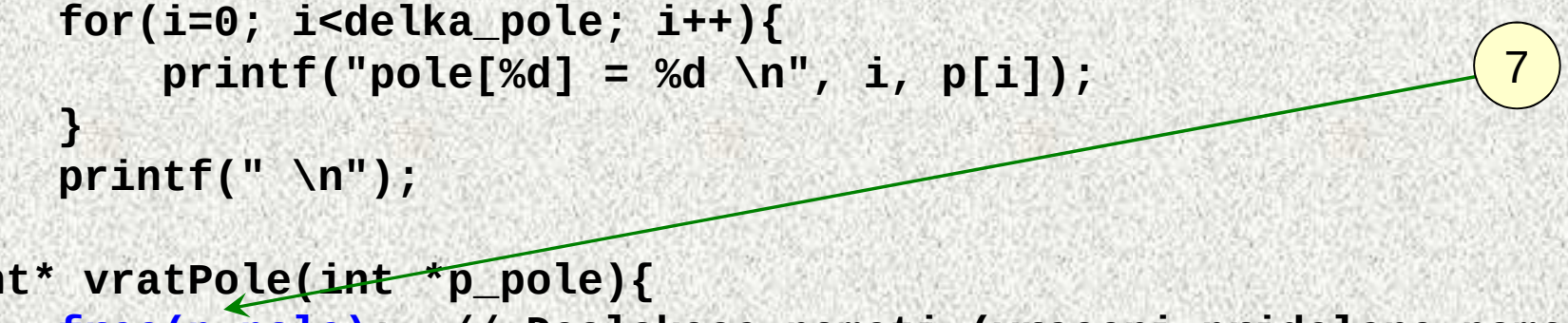
C – dynamická alokace pole a ukazatele /jiné/ (8)

```
int* ctiPole1 (int *delka, int max_delka){
       // Navratovou hodnotou funkce je ukazatel na pridelene pole
    int i, *p;
    printf(" Zadejte pocet cisel = ");
    if (!nextInt(delka)) {
        printf("\n Chyba - Zadany udaj neni cele cislo\n\n");
        exit(EXIT_FAILURE);
    }
    if(*delka <1 || *delka > max_delka){
        printf("\n Chyba - pocet cisel = <1,%d> \n\n",MAX_DELKA);
        exit(EXIT_FAILURE);
    }    // Alokace pameti (prideleni pameti z "heapu")
    if((p=(int*)malloc((*delka)*sizeof(int))) == NULL){
        printf("\n Chyba - neni dostatek volne pameti \n\n");
        exit(EXIT_FAILURE);
    }
    printf("\n Zadejte cela cisla (kazde ukoncit ENTER)\n\n");
    for (i = 0; i < *delka; i++) {
        if (!nextInt(p+i)) {
            printf("\n Chyba - Zadany udaj neni cele cislo\n\n");
            exit(EXIT_FAILURE);
        }
    }
    return(p); // p - ukazatel na pridelene a naplnene pole
}
```



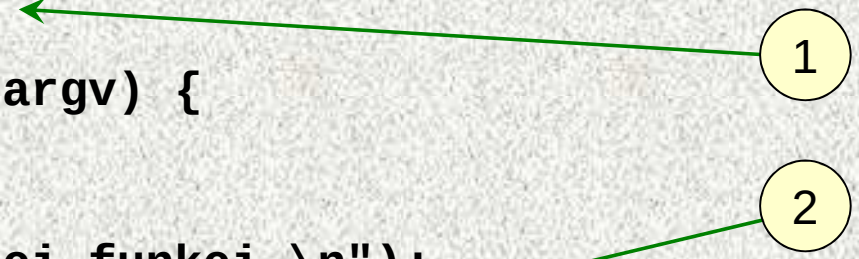
C – dynamická alokace pole a ukazatele /jiné/ (8)

```
void obratPole (int p[], int delka_pole){
       int i, x, n_pul;
    n_pul= delka_pole/2;
    for(i=0; i<n_pul; i++ ){
        x = p[i];
        p[i] = p[delka_pole-i-1];
        p[delka_pole-i-1] = x;
    }
}
void vypisPole (int p[], int delka_pole){
       int i;
    printf("\n Vypis cisel v obracenem poradi \n\n");
    for(i=0; i<delka_pole; i++){
        printf("pole[%d] = %d \n", i, p[i]);
    }
    printf(" \n");
}
int* vratPole(int *p_pole){
    free(p_pole);  // Dealokace pameti (vraceni pridelene pameti)
    p_pole = NULL;  // Bezpecnostni opatreni
    return(p_pole);
}
```



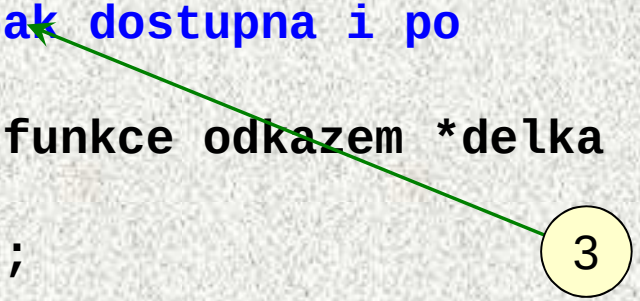
C – dyn. alokace pole a ukazatel na ukazatel /jiné/ (9)

```
#include <stdio.h>
#include <stdlib.h>
// Otoc pole
void ctiPole2 (int **p_pole, int *delka, int max_delka);
void obratPole (int p[], int delka_pole);
void vypisPole (int p[], int delka_pole);
void vratPole1(int **p_pole);
int nextInt(int *cislo);
int main(int argc, char** argv) {
    #define MAX_DELKA 20
    int *p_pole, n;
printf(" Obrat pole - pomoci funkci \n");
printf(" Dynamicke prideleni (alokace) pameti \n");
printf("Predani ukazatele odkazem(ukazatel na ukazatel)");
    ctiPole2(&p_pole, &n, MAX_DELKA);
    obratPole(p_pole, n);
    vypisPole(p_pole, n);
    vratPole1(&p_pole);
    return (EXIT_SUCCESS);
}
```



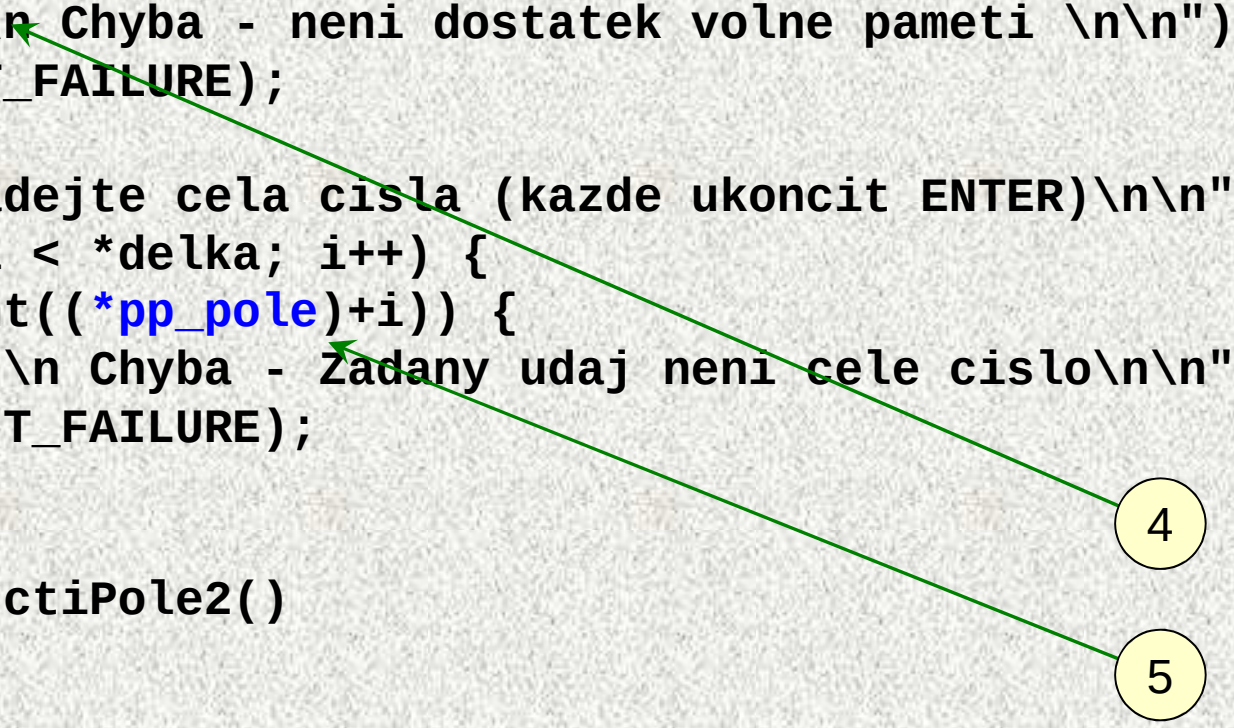
C – dyn. alokace pole a ukazatel na ukazatel /jiné/ (9)

```
void ctiPole2 (int **pp_pole, int *delka, int max_delka){
    // Pole je prideleno (alokovano dynamicky - malloc()
    // Ukazatel se bezne predava hodnotou, proto kdyz je treba
    // predat ukazatel odkazem (call by reference) pouzijeme
    // ukazatel na ukazatel (tj. **p_pole). Adresa pole
    // pridelena funkci malloc() je pak dostupna i po
    // ukonceni funkce ctiPole2()
    // Pocet prvku pole se predava z funkce odkazem *delka
    int i;
    printf(" Zadejte pocet cisel = ");
    if (!nextInt(delka)) {
        printf("\n Chyba - Zadany udaj neni cele cislo\n\n");
        exit(EXIT_FAILURE);
    }
    if(*delka <1 || *delka > max_delka){
     printf("\n Chyba - pocet cisel = <1,%d> \n\n",MAX_DELKA);
     exit(EXIT_FAILURE);
    }
       // Pokracovani funkce ctiPole2 na dalsi strance
```



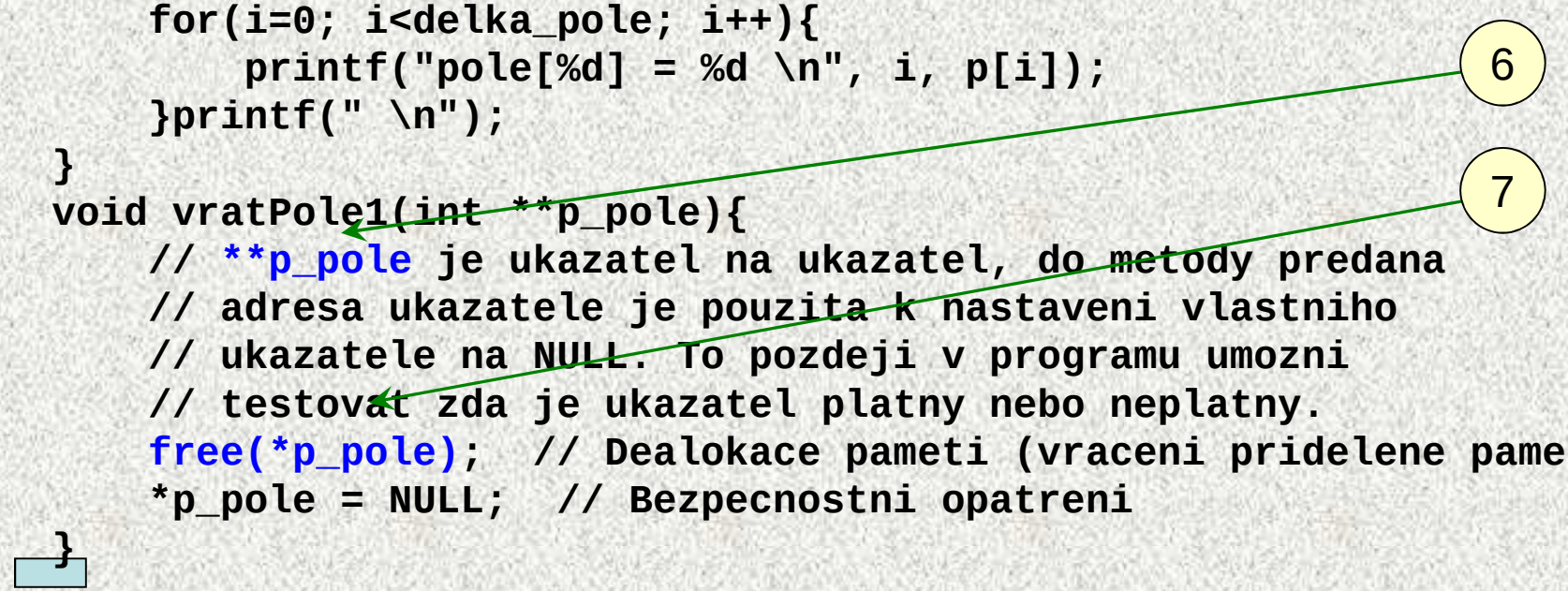
C – dyn. alokace pole a ukazatel na ukazatel /jiné/ (9)

```
// Pokracovani funkce ctiPole2 z predchozi stranky

// Alokace pameti (prideleni pameti z "heapu")
  if((*pp_pole=(int*)malloc((*delka)*sizeof(int))) == NULL){
      printf("\n Chyba - neni dostatek volne pameti \n\n");
      exit(EXIT_FAILURE);
  }
  printf("\n Zadejte cela cisla (kazde ukoncit ENTER)\n\n");
  for (i = 0; i < *delka; i++) {
    if (!nextInt((*pp_pole)+i)) {
      printf("\n Chyba - Zadany udaj neni cele cislo\n\n");
      exit(EXIT_FAILURE);
     }
  }
}// konec funkce ctiPole2()
```



C – dyn. alokace pole a ukazatel na ukazatel /jiné/ (9)

```
void obratPole (int p[], int delka_pole){
    int i, x, n_pul;n_pul= delka_pole/2;
    for(i=0; i<n_pul; i++ ){
        x = p[i];
        p[i] = p[delka_pole-i-1];
        p[delka_pole-i-1] = x;
    }
}
void vypisPole (int p[], int delka_pole){
    int i;
    printf("\n Vypis cisel v obracenem poradi \n\n");
    for(i=0; i<delka_pole; i++){
        printf("pole[%d] = %d \n", i, p[i]);
    }printf(" \n");
}
void vratPole1(int **p_pole){
    // **p_pole je ukazatel na ukazatel, do metody predana
    // adresa ukazatele je pouzita k nastaveni vlastniho
    // ukazatele na NULL. To pozdeji v programu umozni
    // testovat zda je ukazatel platny nebo neplatny.
    free(*p_pole);  // Dealokace pameti (vraceni pridelene pameti)
    *p_pole = NULL;  // Bezpecnostni opatreni
}
```

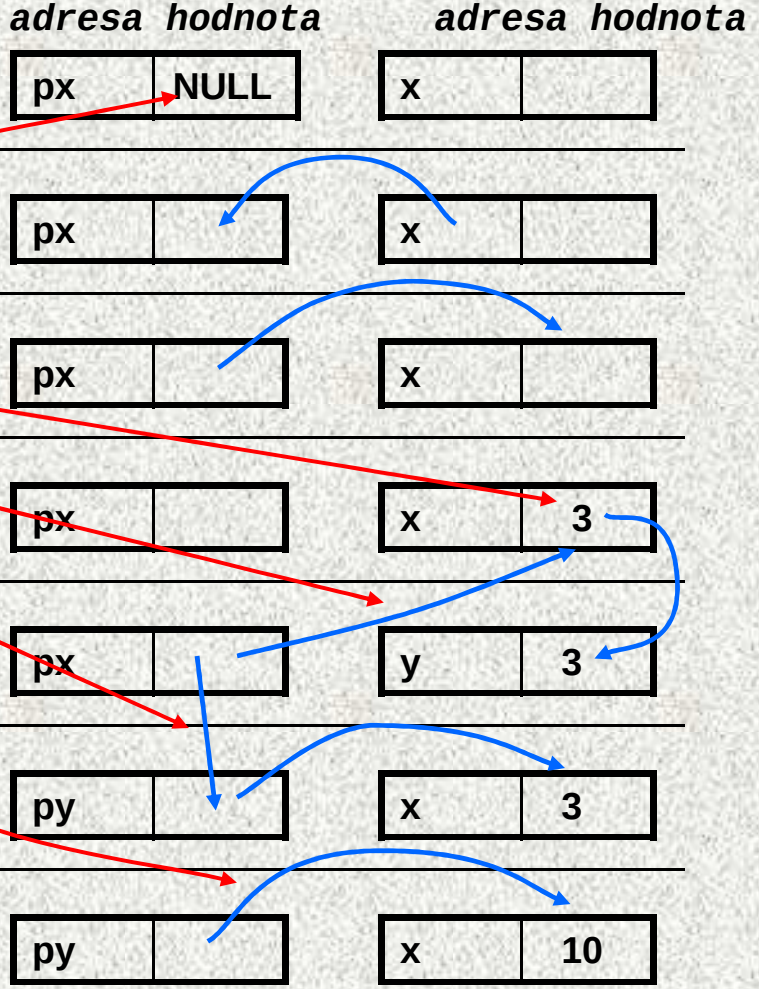


C - Ukazatel (pointer)

C - Ukazatel (pointer) pokrač:

př:

```
int x,y;
int *px,*py;
px=NULL;
px=&x;
x=3;
y=*px;
py=px;
*py=10;
```

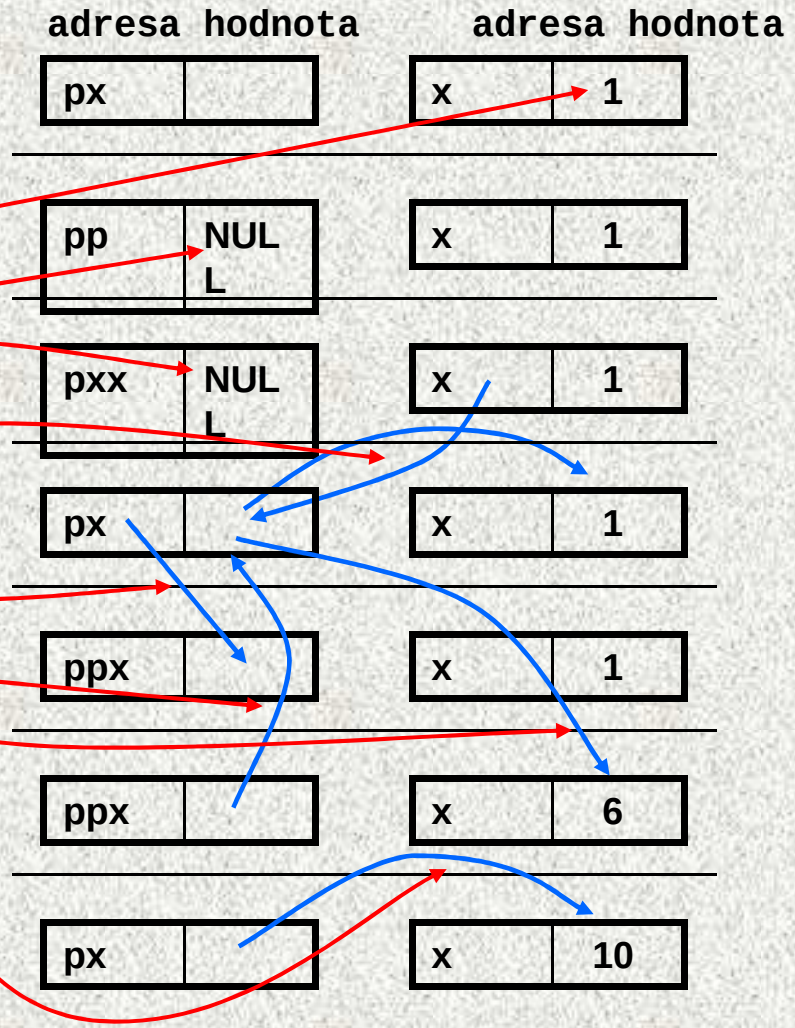


C - Ukazatel (pointer)

C - Ukazatel (pointer) pokrač:

př:

```
int x;
int *px
int **ppx;
x=1;
px=NULL;
ppx=NULL;
px=&x;
ppx=&px;
**ppx=6;
*px=10;
```



C - Ukazatel (pointer)

```
int i = 7;
int j;
int *p_i = &i;
int *p_j;
i = 9;
*p_i = 10;
*p_j = &j;
j= 111;
j = 19;
*p_j = 20;
p_j = &j;
p_i = p_j;
*p_j = 30;
*p_j = *p_i;
```

C – Pole a ukazatel (pointer)

C - Ukazatel na pole (pointer to array):

- Aritmetické operace s ukazatelem na pole zajistí *správný výpočet*, *z typu* ukazatele se zjistí velikost prvku pole a ukazatel se posune o *(pocet_prvku*velikost_prvku*)