A0B36PRI - PROGRAMOVÁNÏ

Jazyk C

Část III.

České vysoké učení technické Fakulta elektrotechnická

v1.04

Obsah

- Porovnání JAVA vs „C“
  - Pole
  - Pole a metody (předávání parametrů, návratová hodnota)
  - Vícerozměrné pole
  - String
  - Ukazatele
  - Pole a ukazatele
  - Dynamicky alokované pole
  - Ukazatel na ukazatel
  - Struktura
  - Union
  - Struktura v unionu
  - Ukazatel na funkce
  - Pole ukazatelů na funkce
  - Podmíněný překlad

Obsah

Přehled

- C - proměnné, deklarace, definice
- C - proměnné, paměťová třída
- C - proměnné, pole (array)
- C - proměnné, ukazatel (pointer)
- C - proměnné, struktura (struct)
- C - proměnné, sdílená paměť (union)
- C – funkce
- C – funkce - parametry
- C – funkce – parametr - pole
- C – funkce – parametr - pointer
- C – funkce – parametry main()
- C - příkazy preprocesoru
- C - standardní knihovny

Použité barevné značení

Rozlišení jazyka

Java

“C”

Srovnatelné vlastnosti

Rozlišení stupně znalostí

Základní znalost

Požadovaná znalost

Doporučená znalost

Pro zájemce

Pomůcka pro “C”

Přehled

C - Pole (array)

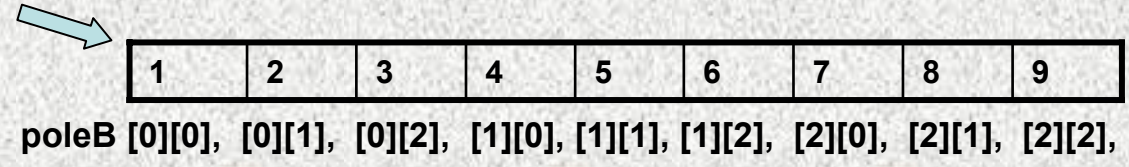
C - Pole (array):

- 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 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* prvkek takového pole je vždy *'\0' (null char)*
- C - *nekontroluje* za běhu programu, zda vypočítaný *index je platný*

C - Pole (array)

C - Deklarace pole (array):

```
př:
char poleA [10];  // jednorozmerne pole z prvku char
int poleB[3][3];  // dvourozmerne pole z prvku int
Uložení v paměti
```



C - Inicializace pole:

```
př:
double x[3]={0.1,0.5,1.3];
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:

```
př:
    ai[1][3]=15*2;
```

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[i]);
    }
    printf(" \n");
    return (EXIT_SUCCESS);
}
```

Statické pole

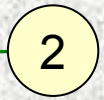
1

2

3

Ošetřené chyby

C - Pole – obrat pole /jiné/ (1)

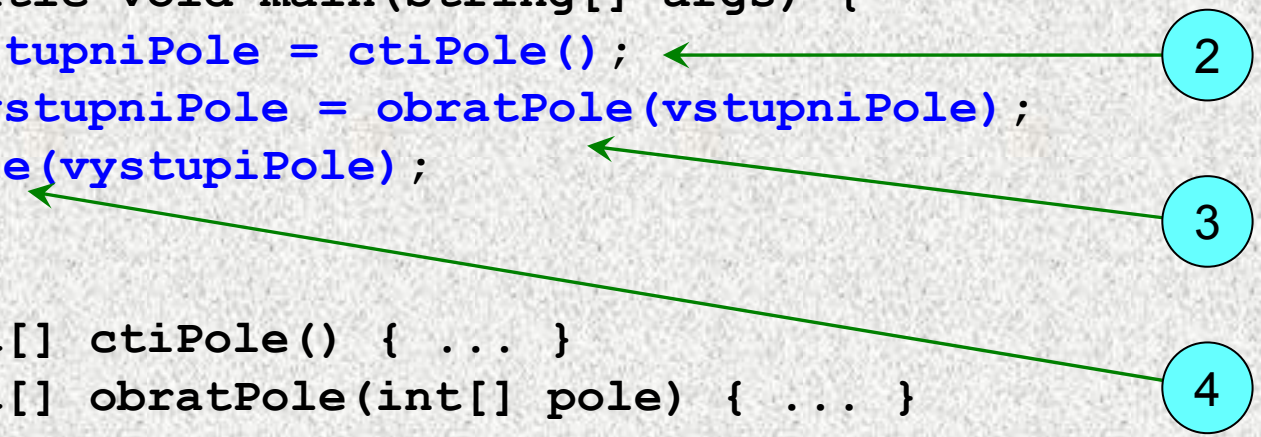



```
int nextInt(int *cislo){
    // === Bezpecne pro libovolny zadany pocet znaku ===
    // Navratova hodnota:
    // TRUE  - zadano cele cislo
    // FALSE - neplatny vstup
    enum boolean {FALSE,TRUE};
    const int BUF_SIZE = 80;
    char vstup[BUF_SIZE],smeti[BUF_SIZE];
    fgets(vstup,sizeof(vstup),stdin);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

JAVA - pole a metody /jiné/ (2)

- Odkaz na pole může být parametrem funkce i jejím výsledkem

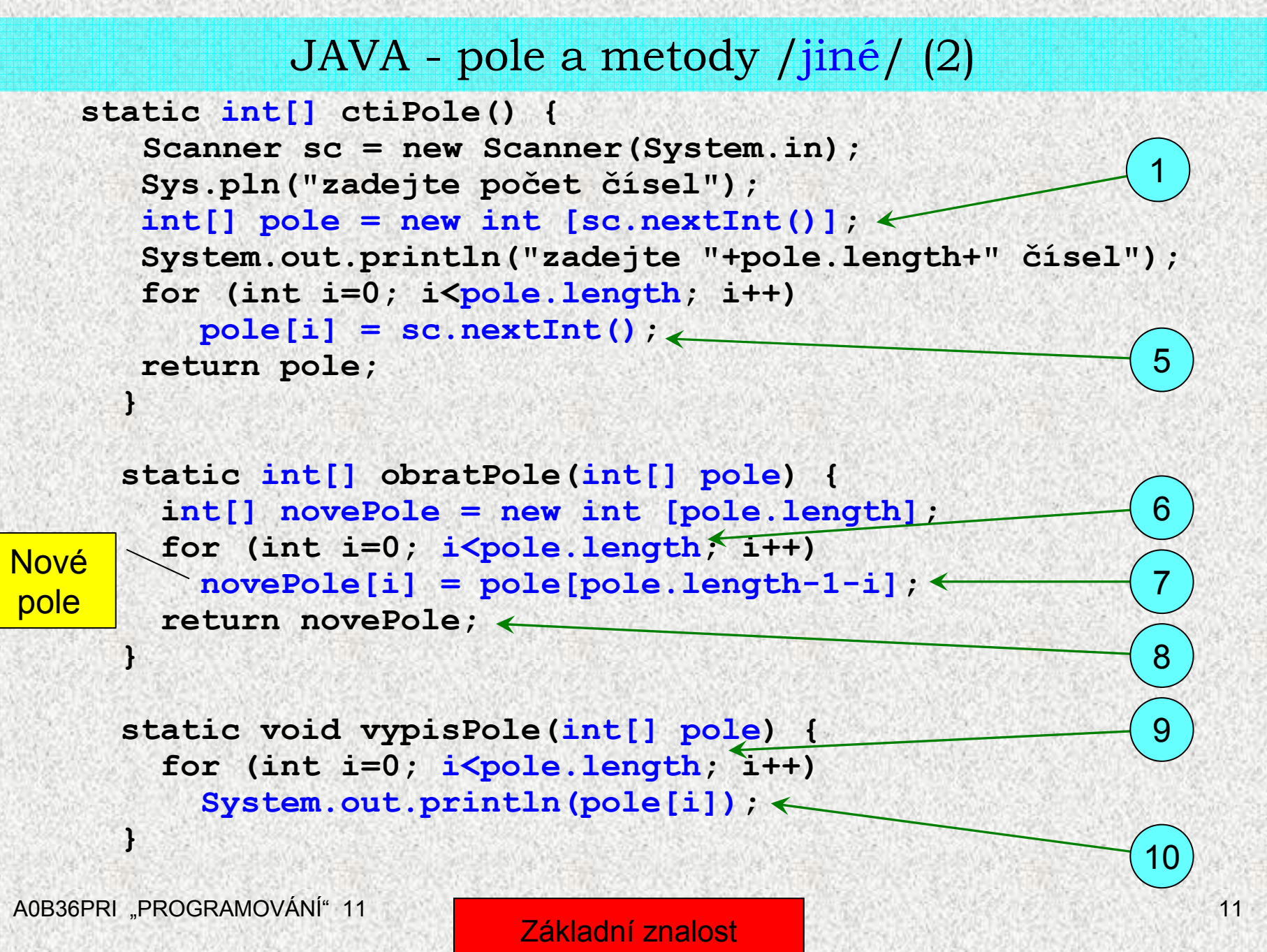
```
import java.util.*;

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 metody /jiné/ (2)

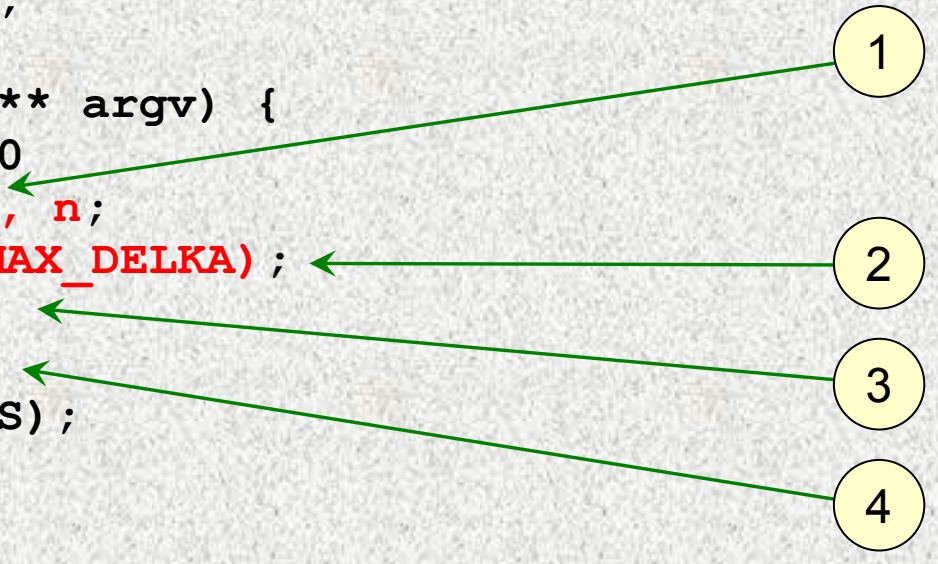
```
static int[] ctiPole() {
   Scanner sc = new Scanner(System.in);
   Sys.pln("zadejte počet čísel");
   int[] pole = new int [sc.nextInt()];
   System.out.println("zadejte "+pole.length+" čísel");
   for (int i=0; i<pole.length; i++)
      pole[i] = sc.nextInt();
   return pole;
  }

  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



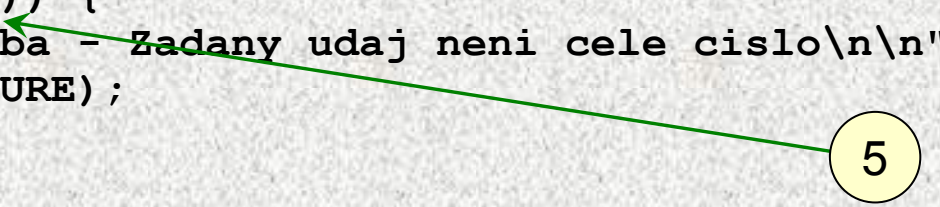
C - pole a funkce /jiné/ (2)

```
#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 (call by reference)
    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);
}
```



Základní znalost

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

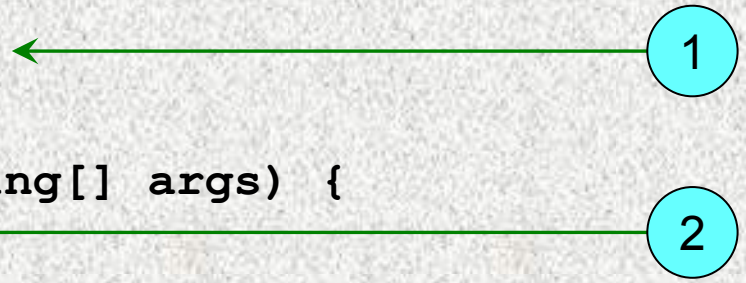
7

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]);
}
```

3

4

Neošetřené chyby,

5

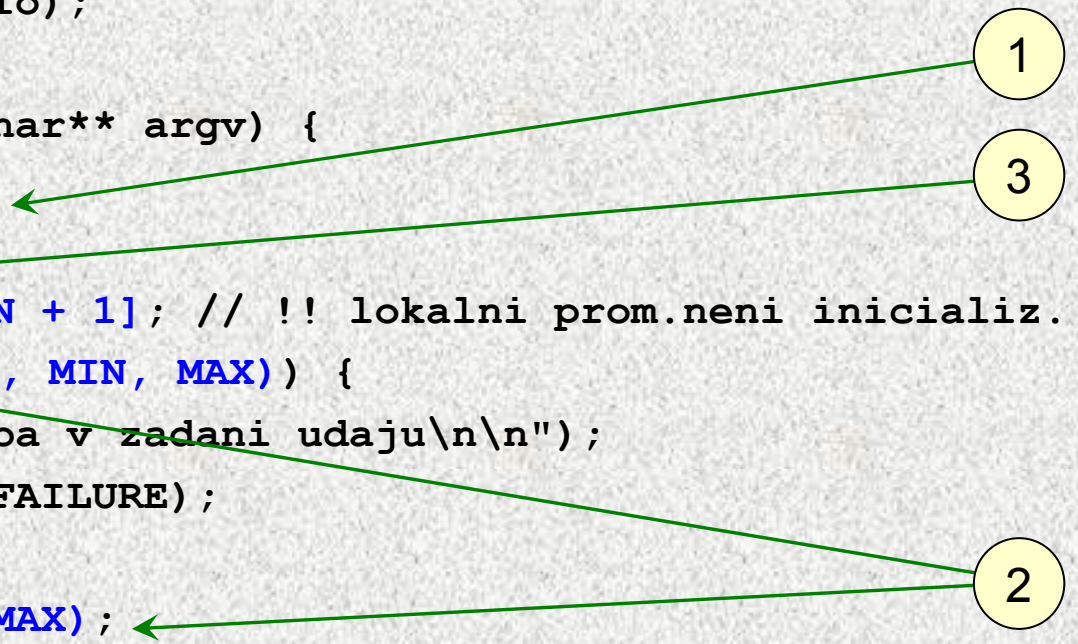
6

7

Základní znalost

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

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

int tabulka(int t[], int min, int max); // Function prototypes
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);
}
```



Základní znalost

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
}
```

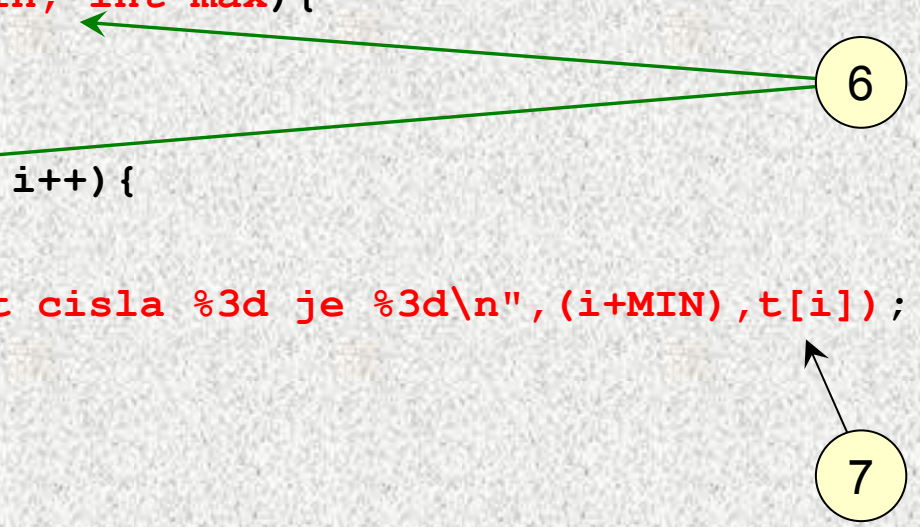
Ošetřené chyby

5

4

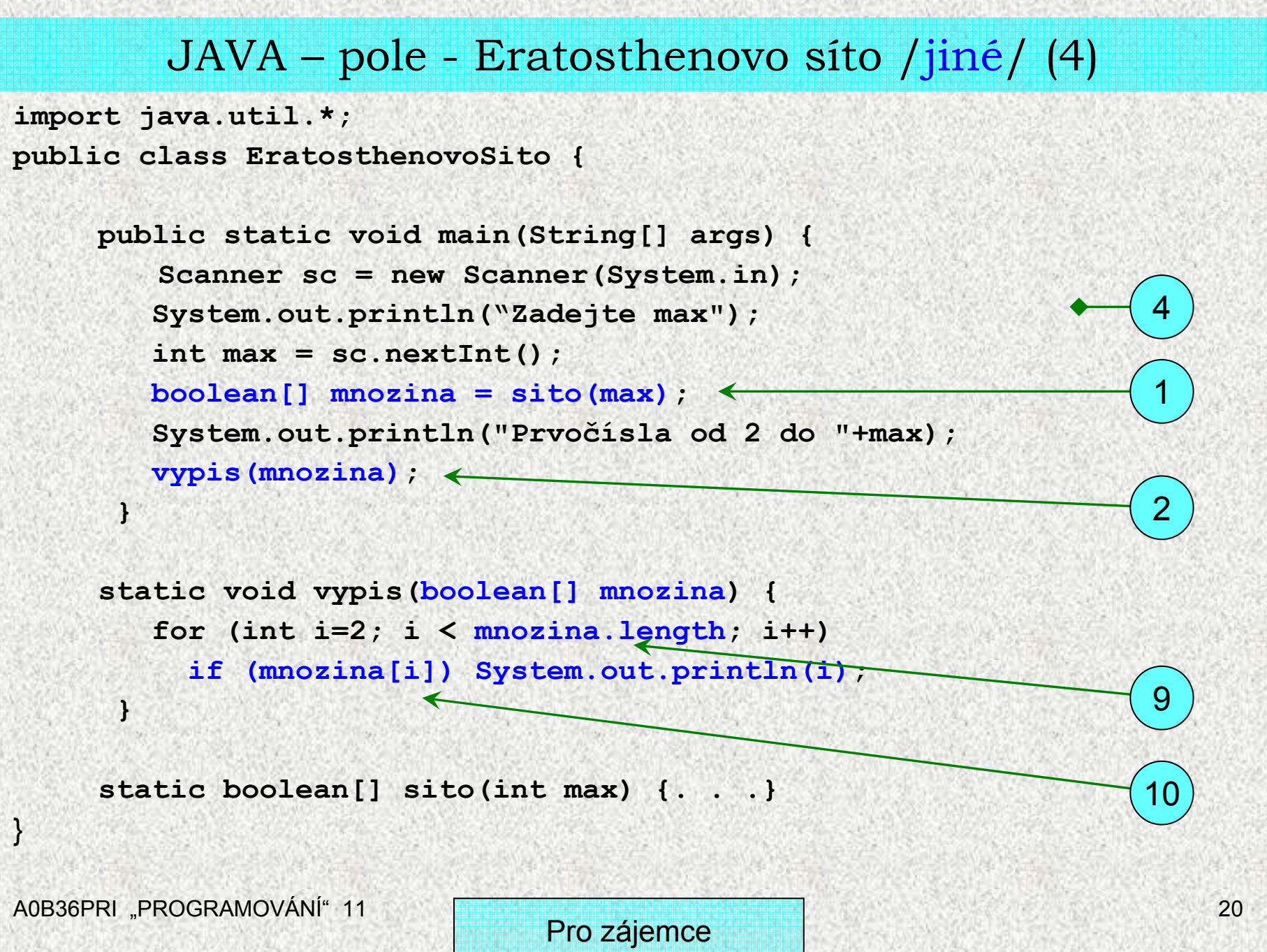
C – pole – tabulka četnosti /jiné/ (3)

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

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



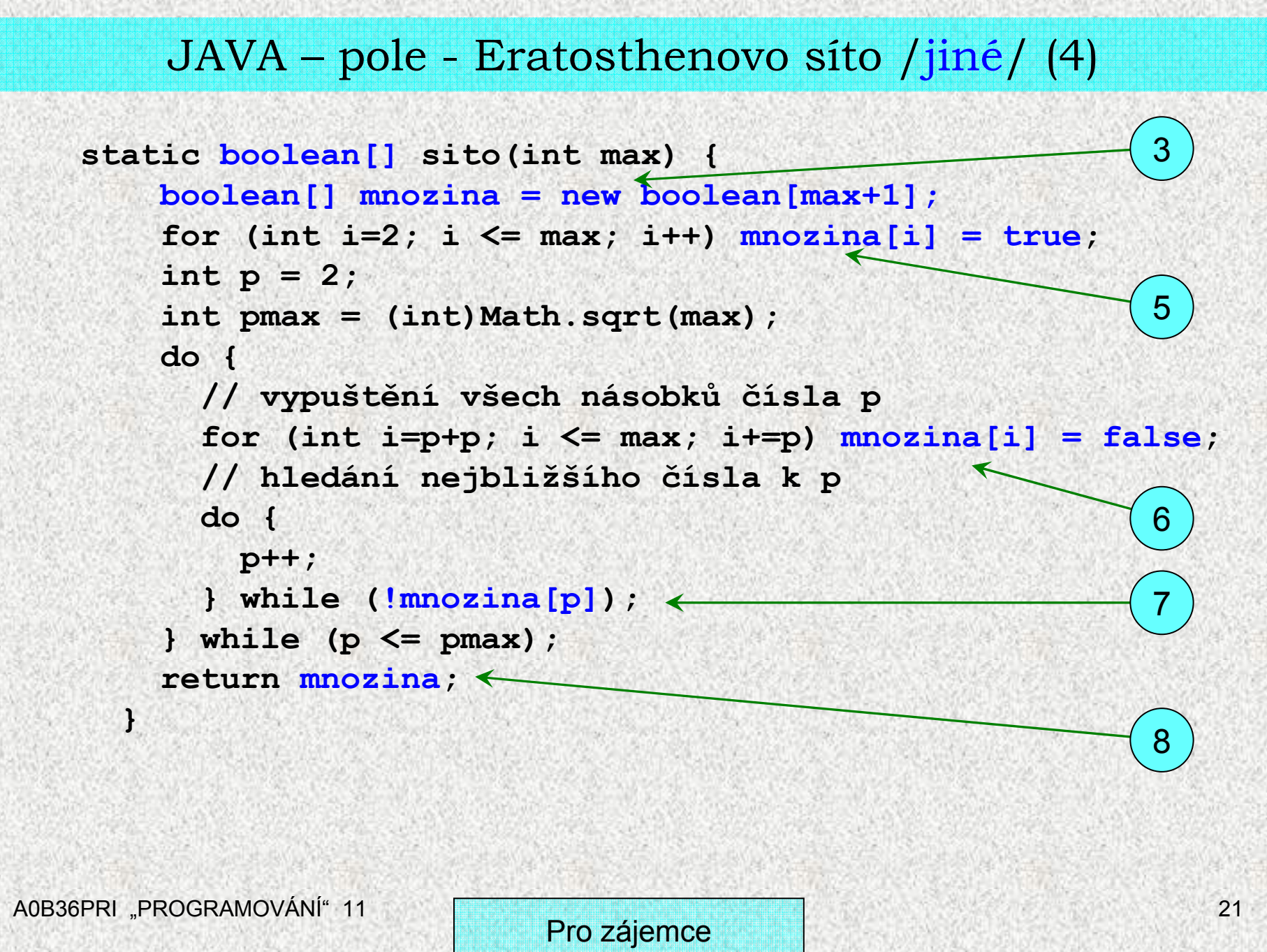
JAVA – pole - Eratosthenovo síto /jiné/ (4)

```
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) {. . .}
}
```



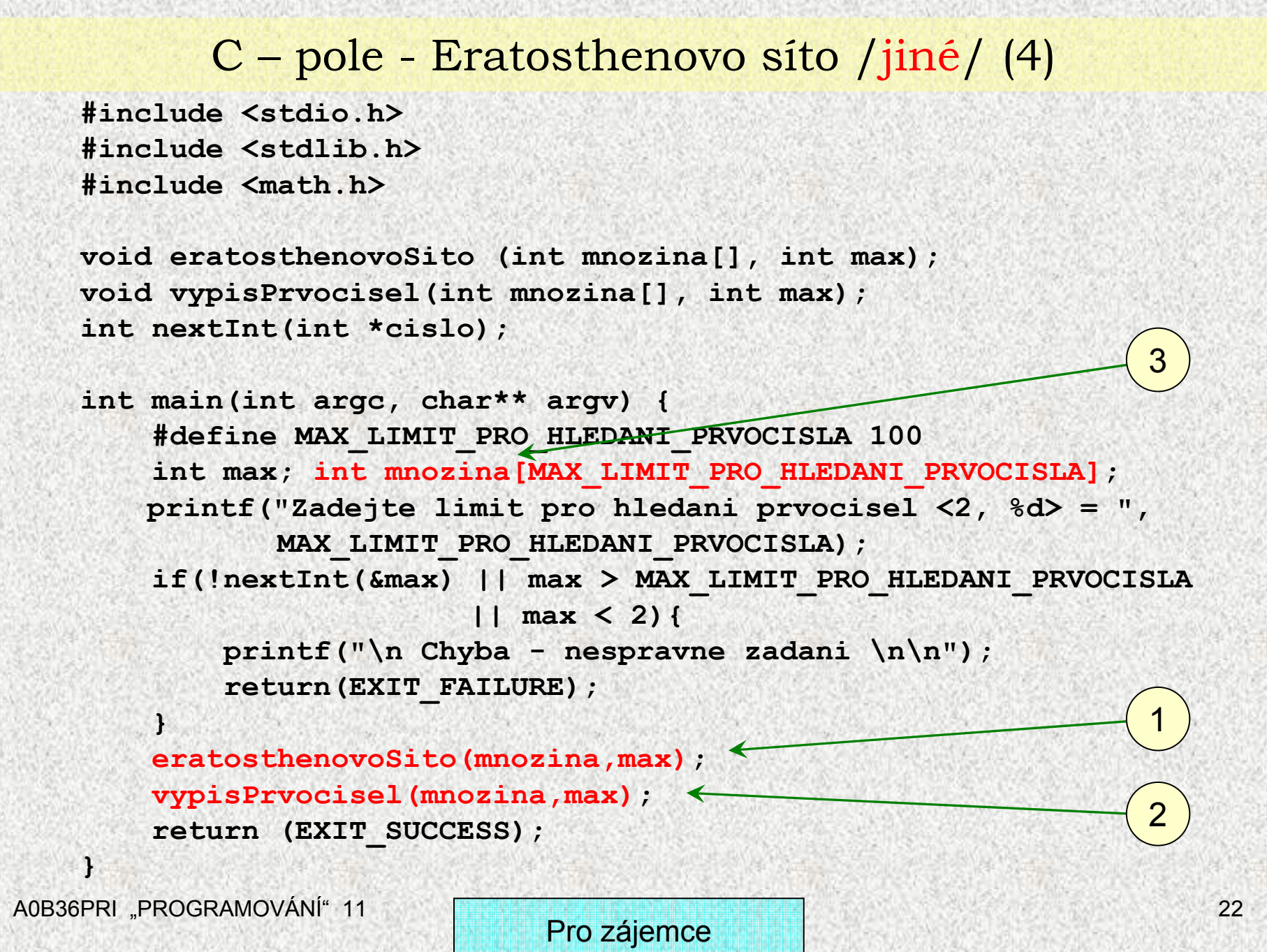
JAVA – pole - Eratosthenovo síto /jiné/ (4)

```
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);
}
```



Pro zájemce

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);
}
```

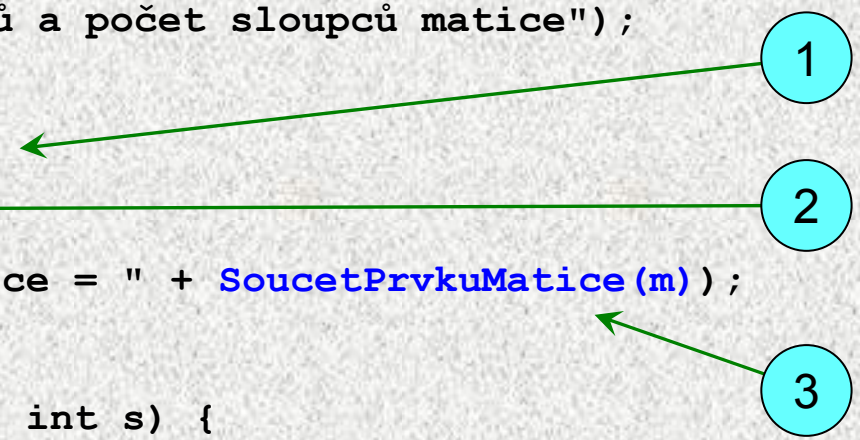
C – pole - Eratosthenovo síto /jiné/ (4)

```
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");
}
```

Pro zájemce

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

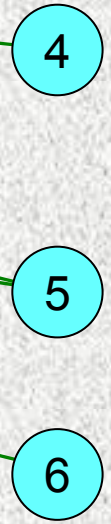
```
import java.util.*;
public class Matice {
   static Scanner sc = new Scanner(System.in);

  public static void main(String[] args) {
    System.out.println(
           "zadejte počet řádků a počet 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);
  }
```



Základní znalost

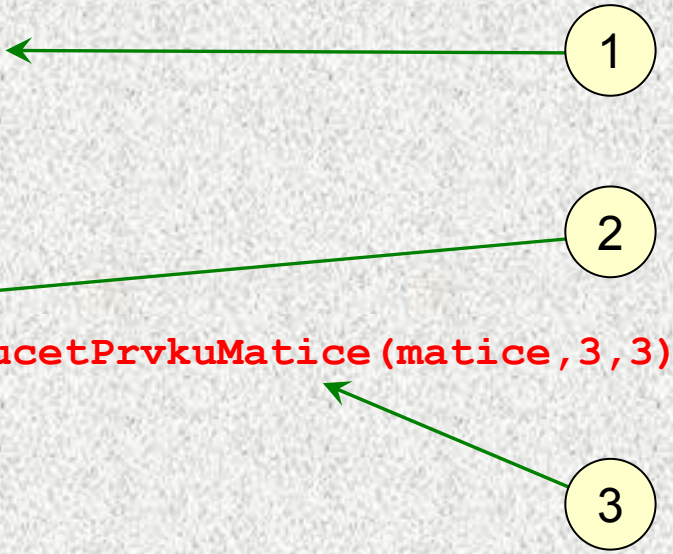
JAVA – vícerozměrné pole - součet prvků /podobné/ (5)

```
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)

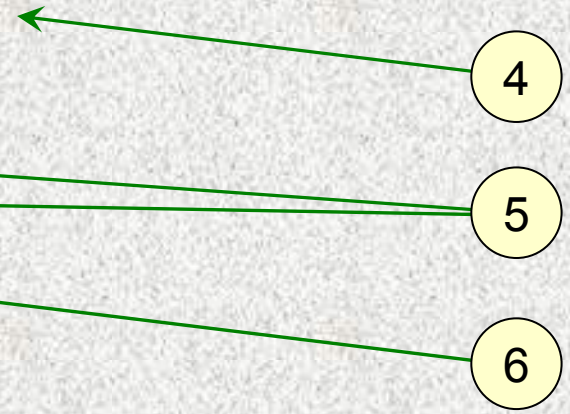
```
#include <stdio.h>
#include <stdlib.h>

#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);
}
```



Požadovaná znalost

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) {
    ...
  }
}
```

Typ String

1

2

3

4

Základní znalost

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 z char

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 - Ukazatel (pointer)

C - Ukazatel (pointer):

- *Ukazatel* je proměnná obsahující *adresu jiné proměnné* (nepřímá adresa)
- *Ukazatel* má též *typ* proměnné na kterou může *ukazovat*
  - ukazatel na char
  - ukazatel na int atd.
- Ukazatel může být též bez typu *(void*), pak může obsahovat *adresu libovolné proměnné*. Její velikost pak nelze z vlastností ukazatele určit
- *Neplatná adresa* v ukazateli má hodnotu konstanty *NULL*
- C za běhu programu *nekontroluje* zda adresa v ukazateli *je platná*

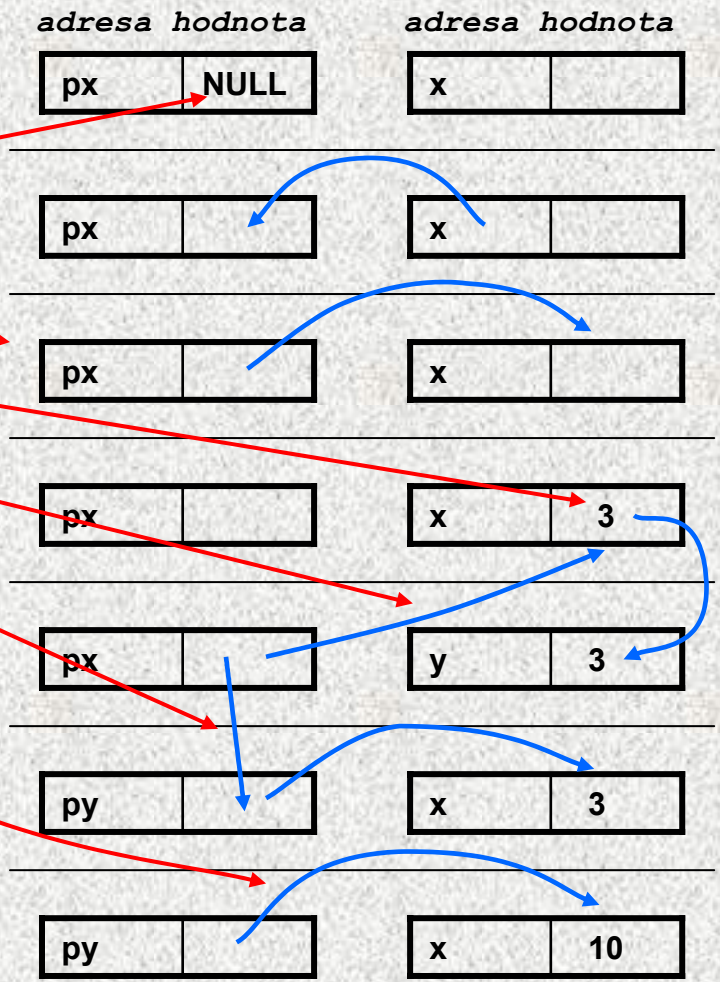
- *Adresa proměnné* se zjistí adresovým operátorem ***&***
- K proměnné na kterou *ukazatel ukazuje se přistoupí* operátorem nepřímé adresy ******* *(hvězdička*)

- Ukazatel může obsahovat též *adresu funkce* (ukazatel na funkci)
- Pomocí *ukazatele na funkci* lze *funkci volat*, včetně předání parametrů
- Pomocí *ukazatele* lze předávat parametry funkci *odkazem (call by reference*) (*standardní metoda v C* pro ostatní případy je *volání hodnotou (call by value))*

C - Ukazatel (pointer) pokrač:

př:

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

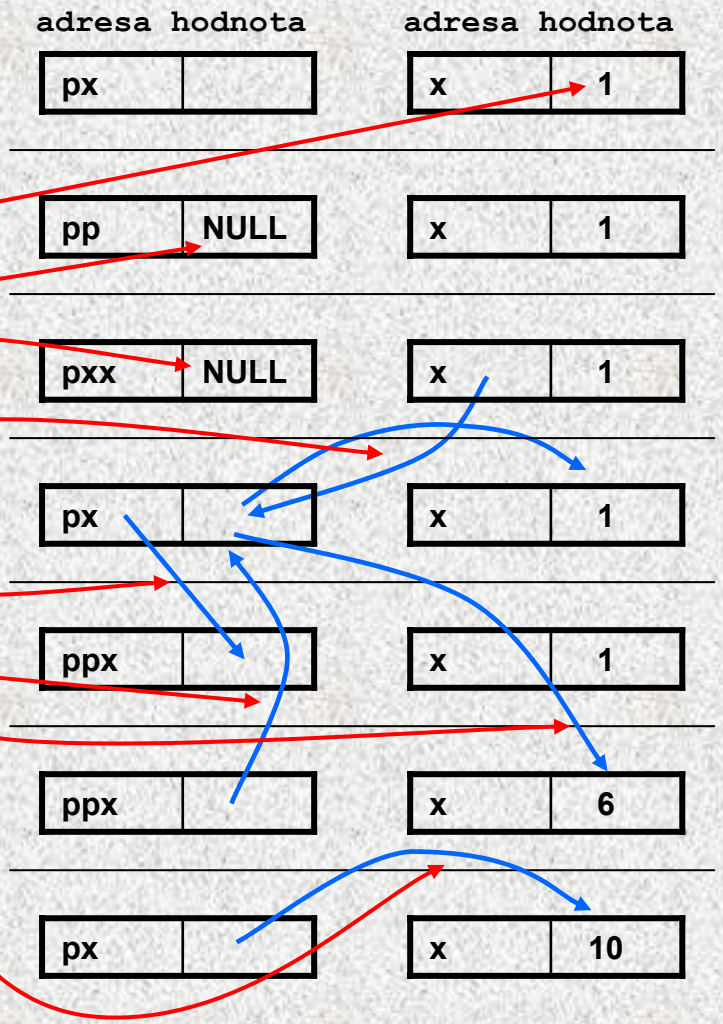


Základní znalost

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;
```



Základní znalost

C - Ukazatel (pointer)

C - Ukazatel (pointer) - aritmetické operace:

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

C - Ukazatel na pole (pointer to array):

- *Aritmetické* operace jsou *užitečné* když ukazatel *ukazuje na pole*
- Jméno pole je konstantní ukazatel na počátek pole (na prvek a[0])
- 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)*

```
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[5]
        px=&a[3];             // px ukazuje na a[3]
```


Základní znalost

C - Ukazatel (pointer)

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

- V složitějších deklaracích ukazatelů mohou být nezbytné závorky

```
př:  double a[10];          // Pole z prvku double
     double (*pa)[10];      // Ukazatel na pole 10 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
```

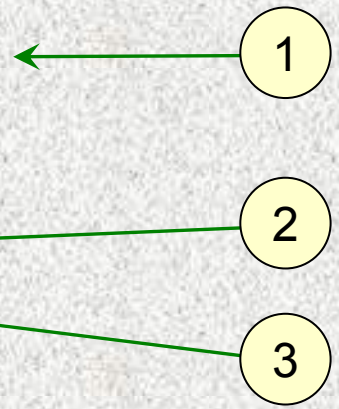
C - Ukazatel na funkci:

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


Požadovaná znalost

C – pole a ukazatele /jiné/ (7)

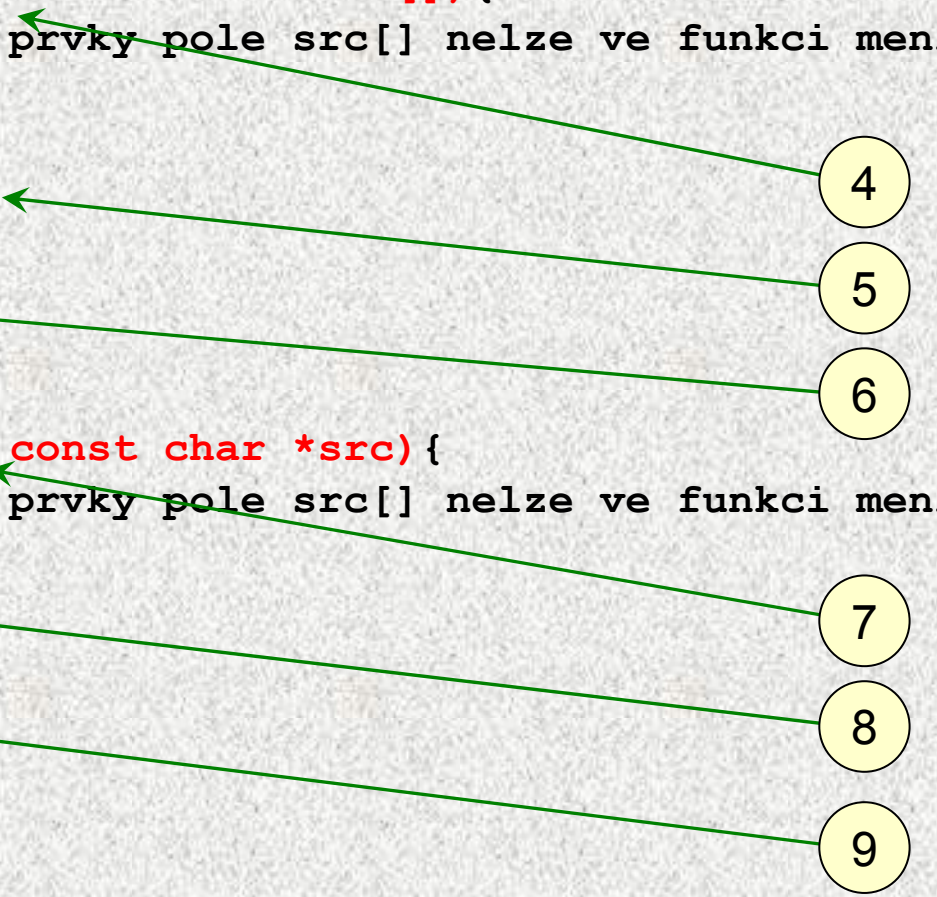
```
#include <stdio.h>
#include <stdlib.h>
// Pointers and arrays - ekvivalence syntaxe
void copyArray1(char dst[], const char src[]);
void copyArray2(char *dst, const char *src);
void copyArray3(char *dst, const char *src);
void copyArray4(char dst[], const char src[]);
int main(int argc, char** argv) {
    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)

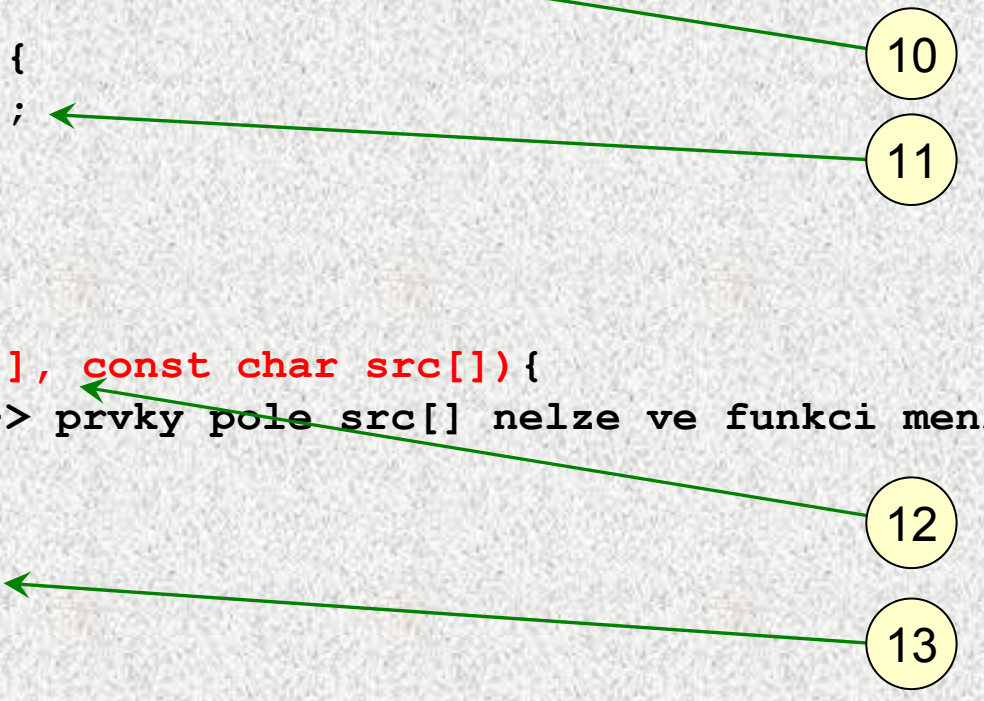
```
void copyArray1(char dst[], const char src[]){
    // const char src[] -> prvky pole src[] nelze ve funkci menit
    int i=0;
    while(src[i] != '\0'){
        dst[i] = src[i++];
    }
    dst[i] = '\0';
}

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)

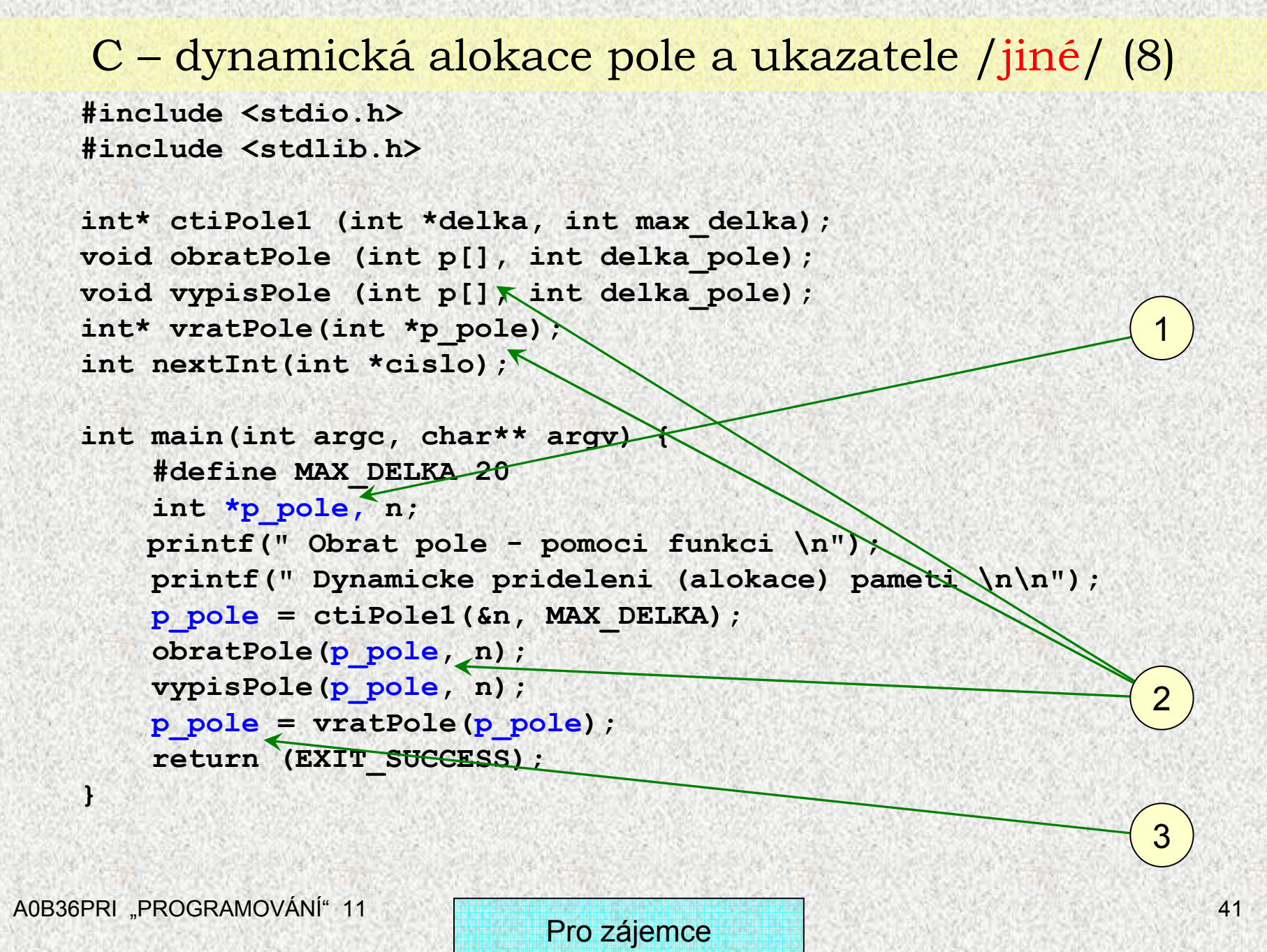
```
void copyArray3(char *dst, const const char *src){
    // const char src[] -> prvky pole src[] nelze ve funkci menit
    int i=0;
    while(src[i] != '\0'){
        *dst++ = src[i++];
    }
    *dst = '\0';
}

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 – 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);
}
```



Pro zájemce

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
}
```

4

5

6

Pro zájemce

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

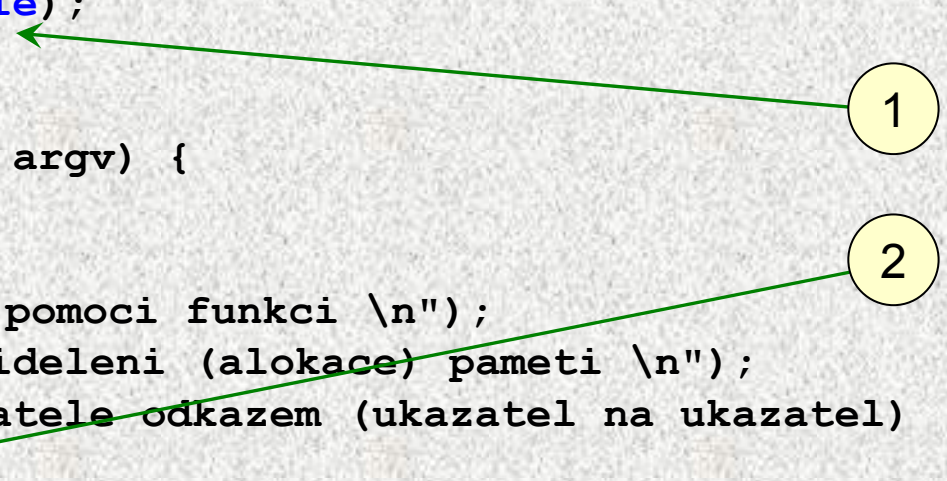
```
void obratPole (int p[], int delka_pole){
    int i, x, n_pul;
    n_pul= delka_pole/2;
    for(i=0; i<n_pul; i++ ){
        x = p[i];
        p[i] = p[delka_pole-i-1];
        p[delka_pole-i-1] = x;
    }
}

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);
}
```

7

Pro zájemce

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

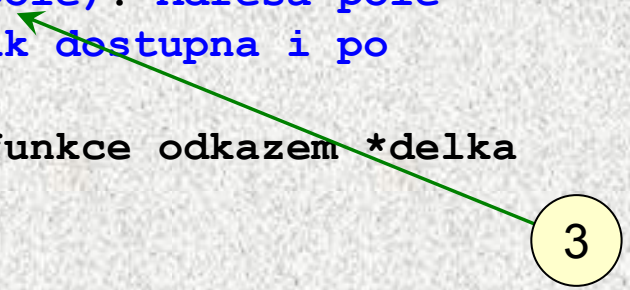
```
#include <stdio.h>
#include <stdlib.h>
// Otoc pole
void ctiPole2 (int **p_pole, int *delka, int max_delka);
void obratPole (int p[], int delka_pole);
void vypisPole (int p[], int delka_pole);
void vratPole1(int **p_pole);
int nextInt(int *cislo);

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)
  \n\n");
    ctiPole2(&p_pole, &n, MAX_DELKA);
    obratPole(p_pole, n);
    vypisPole(p_pole, n);
    vratPole1(&p_pole);
    return (EXIT_SUCCESS);
}
```



Pro zájemce

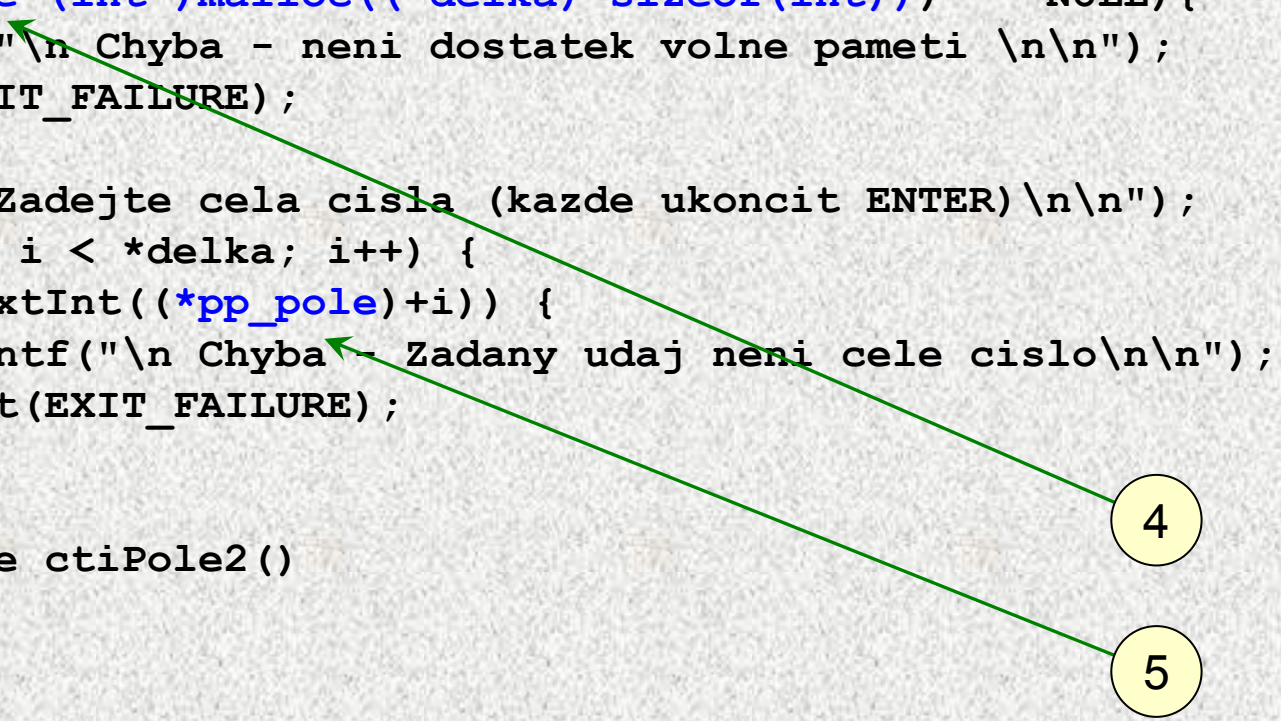
C – dyn. alokace pole a ukazatel na ukazatel /jiné/ (9)

```
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 value) 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)

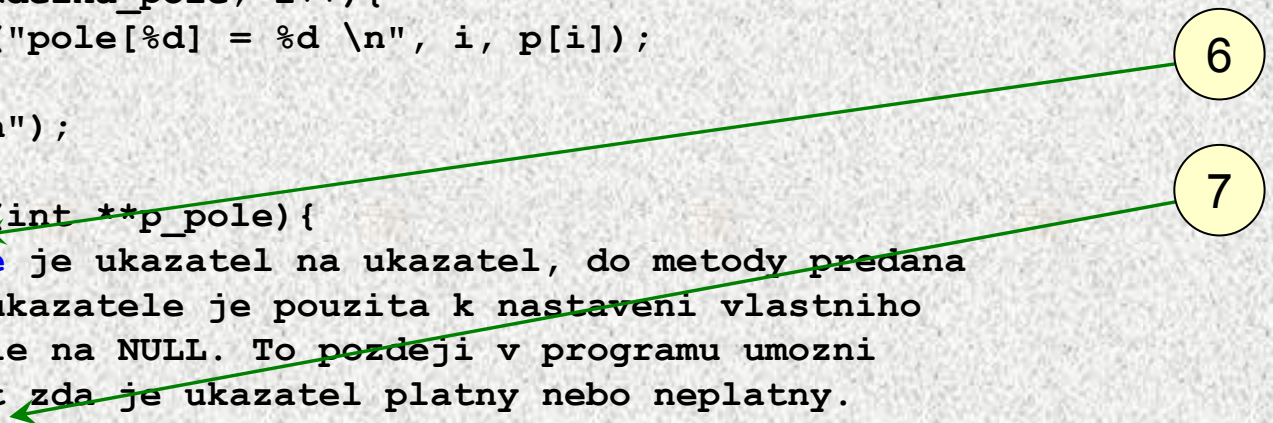
```
    // Pokracovani funkce ctiPole2 z predchozi stranky

    // 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()
```



Pro zájemce

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
}
```



Pro zájemce

C - Struktura (struct)

C - Struktura (struct):

- Struktura je *množina prvků* (proměnných), které *nemusí* být *stejného typu*
- *Skladba* struktury je *definovaná uživatelem* jako *nový typ* sestavený z již definovaných typů

- K prvkům struktury *se přistupuje tečkovou notací*
- K prvkům struktury je *možné přistupovat i pomocí ukazatele* na strukturu operátorem **->**

- Struktury *mohou být vnořené*

- Pro struktury *stejného typu* je definována operace *přiřazení struct1=struct2* (pro proměnné typu pole přímé přiřazení není definováno, jen po prvcích)

- Struktury (jako celek) *nelze porovnávat* relačním operátorem ==

- Struktura *může* být *do funkce předávána hodnotou i odkazem*
- Struktura *může* být *návratovou hodnotou funkce*

C - Struktura (struct)

C - Struktura (struct) - sablona (tag):

```
př:  struct Tid{                  // <==== Tid=jmeno sablony (tag)
          char jmeno[20];         // Prvky struktury, pole
          char adresa[50];        // - " -              pole
          long int telefon;       // - " -              int
          };
     struct Tid sk1,skAvt[20];    // struktura, pole struktur
     struct Tid *pid;             // ukazatel na strukturu
     sk1.jmeno="Jan Novak";       // teckova notace
     sk1.telefon=123456;
     skAvt[0].jmeno="Jan Novak";  // prvek pole
     skAvt[3].telefon=123456;
     pid=&sk1;                    // do pid adresa struktury
     pid->jmeno="Jan Novak";      // odkaz pomoci ->
     pid->telefon=123456;
     (*pid).jmeno="Jan Novak";    // odkaz pomoci *
     (*pid).telefon=123456;
```

C - Struktura (struct)

C - Struktura (struct) - nový typ (typedef):

```
př:  typedef struct {                  // <==== Pomoci Typedef
            char jmeno[20];            // Prvky struktury, pole
            char adresa[50];           // - " -              pole
            long int telefon;          // - " -              int
            }Tid,*Tidp;
     Tid sk1,skAvt[20];                // struktura, pole struktur
     Tidp pid;                         // ukazatel na strukturu
     sk1.jmeno="Jan Novak"; // teckova notace
     sk1.telefon=123456;
     skAvt[0].jmeno="Jan Novak";       // prvek pole
     skAvt[3].telefon=123456;
     pid=&sk1;                         // do pid adresa struktury
     pid->jmeno="Jan Novak";           // odkaz pomoci ->
     pid->telefon=123456;
     (*pid).jmeno="Jan Novak";         // odkaz pomoci *
     (*pid).telefon=123456;
```

C - Struktura (struct)

C - Struktura (struct) - inicializace:

```
př:
    struct Tid{                    // <==== Tid=jmeno sablony (tag)
       char jmeno[20];                // Prvky struktury, pole
       char adresa[50];               // - " -                  pole
       long int telefon;              // - " -                  int
    };

    struct Tid sk1={"Jan Novak",
                    "Na kopecku 23",
                     123456};
```


Pro zájemce

C – struktura – kvadraticka rovnice /jiné/ (10)

```
#include <stdio.h>
#include <stdlib.h>
typedef struct{
    double re1; double im1; double re2; double im2;
}Tcomplex;
int kvadratickaRovnice (double a, double b, double c, Tcomplex *koreny);
void vypisKorenuKvadratRovnice (char jmenoPromenne[], Tcomplex s);
void vypisKomplexCisla (double re, double im);
int nextDouble(double *cislo);

int main(int argc, char** argv) {
    Tcomplex s; double a,b,c;
    printf(" Zadejte koeficienty a, b, c \n\n");
    if(!nextDouble(&a) || !nextDouble(&b) || !nextDouble(&c)){
        printf("\n Chyba - zadany udaj neni cislo\n\n");
        return(EXIT_FAILURE);
    }
    printf("\n");
    if(!kvadratickaRovnice(a,b,c,&s)){
        printf(" Neplatne zadani (a=0)\n\n");
    }else{
        vypisKorenuKvadratRovnice("Koren x",s);
    }
    return (EXIT_SUCCESS);
}
```

1

2

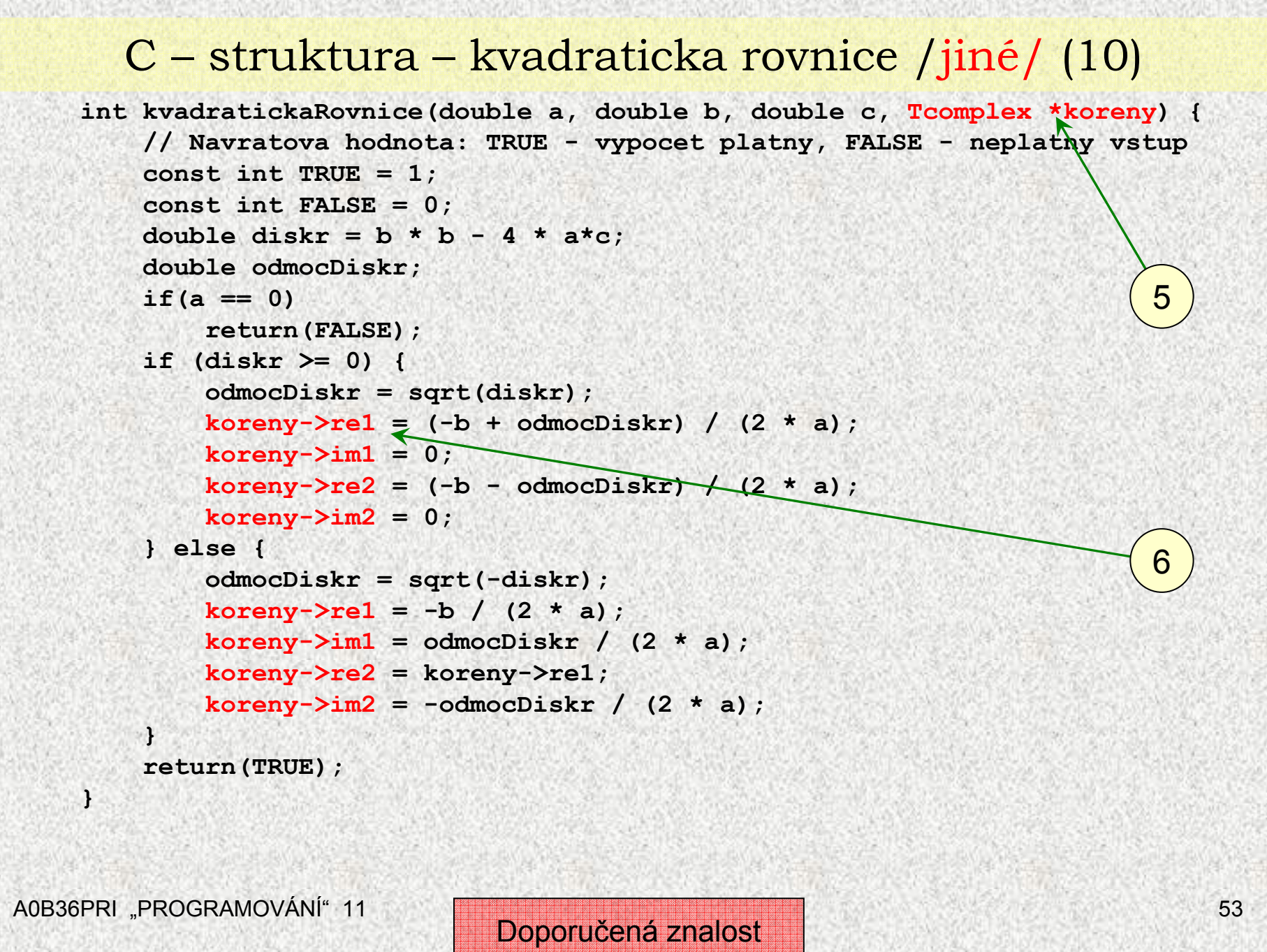
3

4

Doporučená znalost

C – struktura – kvadraticka rovnice /jiné/ (10)

```
int kvadratickaRovnice(double a, double b, double c, Tcomplex *koreny) {
    // Navratova hodnota: TRUE - vypocet platny, FALSE - neplatny vstup
    const int TRUE = 1;
    const int FALSE = 0;
    double diskr = b * b - 4 * a*c;
    double odmocDiskr;
    if(a == 0)
        return(FALSE);
    if (diskr >= 0) {
        odmocDiskr = sqrt(diskr);
        koreny->re1 = (-b + odmocDiskr) / (2 * a);
        koreny->im1 = 0;
        koreny->re2 = (-b - odmocDiskr) / (2 * a);
        koreny->im2 = 0;
    } else {
        odmocDiskr = sqrt(-diskr);
        koreny->re1 = -b / (2 * a);
        koreny->im1 = odmocDiskr / (2 * a);
        koreny->re2 = koreny->re1;
        koreny->im2 = -odmocDiskr / (2 * a);
    }
    return(TRUE);
}
```



Doporučená znalost

C – struktura – kvadraticka rovnice /jiné/ (10)

```
void vypisKorenuKvadratRovnice(char jmenoPromenne[],Tcomplex s){
    printf(" %s1 = ",jmenoPromenne);
    vypisKomplexCisla(s.re1,s.im1);
    printf("\n\n");
    printf(" %s2 = ",jmenoPromenne);
    vypisKomplexCisla(s.re2,s.im2);
    printf("\n\n");
}

void vypisKomplexCisla (double re, double im){
    if(im >= 0){
        printf("%.2f + j%.2f",re,im);
    }else{
        printf("%.2f - j%.2f",re,-im);
    }
}
```

Doporučená znalost

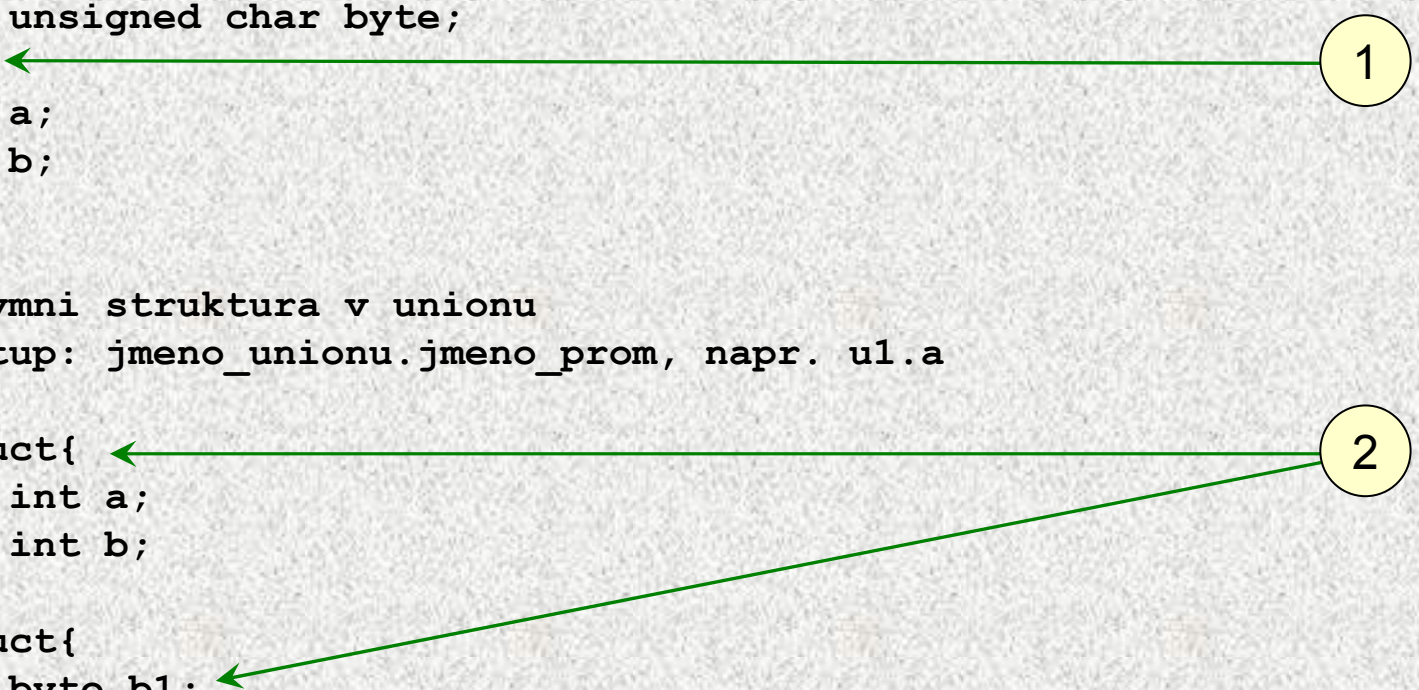
C – struktura – kvadraticka rovnice /jiné/ (10)

```
int nextDouble(double *cislo){
    // Stav: odladeno
    // === Bezpecne pro libovolny zadany pocet znaku ===
    // Navratova hodnota:
    // TRUE  - zadano realne 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,"%lf%[^\n]",cislo,smeti) != 1)
        return(FALSE); // Input error
    return(TRUE);
}
```


Doporučená znalost

C – union a struktura v unionu /jiné/ (11)

```
// Union, struktura v unionu (anonymni a pojmenovana)
// Union - prekryta pamet promennych

typedef unsigned char byte;
union {
    int a;
    int b;
}uab;

// Anonymni struktura v unionu
// Pristup: jmeno_unionu.jmeno_prom, napr. u1.a
union {
    struct{
        int a;
        int b;
    };
    struct{
        byte b1;
        byte b2;
        byte b3;
        byte b4;
    };
}u1;
```



Pro zájemce

C – union a struktura v unionu /jiné/ (11)

```
typedef struct{
    int x1;
    int x2;
}Tintx12;

typedef struct{
    byte bx1;
    byte bx2;
    byte bx3;
    byte bx4;
    byte by1;
    byte by2;
    byte by3;
    byte by4;
}Tbytexy14;
```

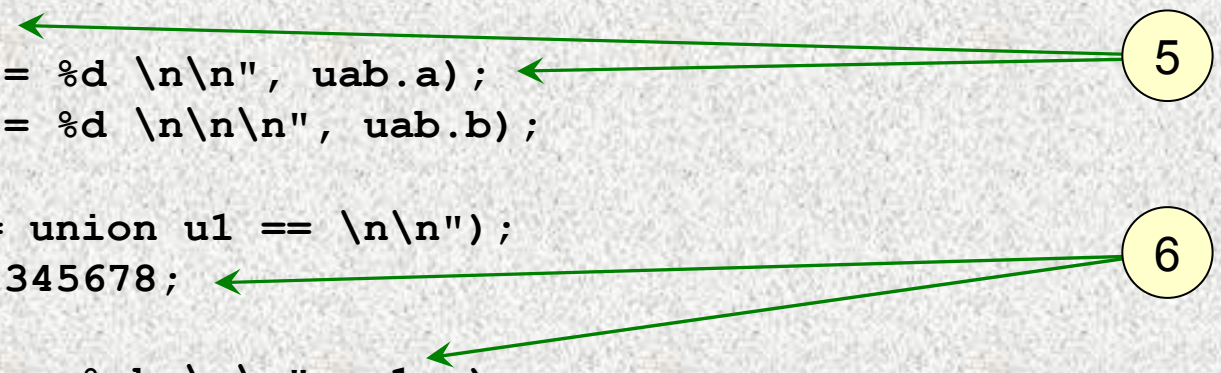
C - union a struktura v unionu /jiné/ (11)

```
// Pojmenovana struktura v unionu
// Pristup: jmeno_unionu.jmeno_struct.jmeno_promenne, napr. u1.intx.x1
union{
    Tintx12 intx;
    Tbytexy14 bytex;
}u2;
union{
   int *px1;
   int *px2;
   int *px3;
}u3;

typedef union{
   int x1;
   int x2;
   int x3;
}Tu4;
Tu4 u4, *pu4;
```

Pro zájemce

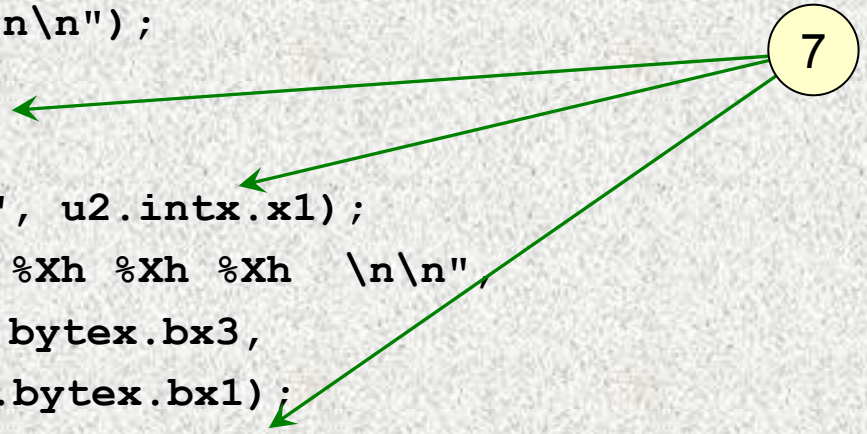
C - union a struktura v unionu /jiné/ (11)

```
int main(int argc, char** argv) {
    int x, y;
    byte b;
    printf("=== C10a.c ===\n\n");
    printf(" union, structure \n\n");

    printf(" == union ab == \n\n");
    uab.a = 1;
    printf(" a = %d \n\n", uab.a);
    printf(" b = %d \n\n\n", uab.b);

    printf(" == union u1 == \n\n");
    u1.a = 0x12345678;
    b = u1.b1;
    printf(" x1 = %xh \n\n", u1.a);
    printf(" b4,3,2,1 = %xh %xh  %xh %xh \n\n\n", u1.b4, u1.b3,
    u1.b2, u1.b1);

    // Pokrac.na dalsi strane
```



C - union a struktura v unionu /jiné/ (11)

```
// Pokrac. z predchozi strany
printf(" == union u2 == \n\n");
u2.intx.x1 = 0x87654321;
u2.intx.x2 = 0x00AB00CD;
printf(" x1 = %08Xh \n\n", u2.intx.x1);
printf(" bx4,3,2,1 = %Xh %Xh %Xh %Xh  \n\n",
        u2.bytex.bx4, u2.bytex.bx3,
        u2.bytex.bx2, u2.bytex.bx1);
printf(" x2 = %08Xh \n\n", u2.intx.x2);
printf(" by4,3,2,1 = %Xh %Xh %Xh %Xh  \n\n\n",
        u2.bytex.by4, u2.bytex.by3,
        u2.bytex.by2, u2.bytex.by1);
printf(" == union u3 == \n\n");
u3.px1 = &x;
*u3.px1 = 10;
printf(" x = %d, *px1 = %d, *px2 = %d, px3 = %d \n\n\n",
        x, *u3.px1, *u3.px2, *u3.px3);
// Pokrac. na dalsi strane
```



C - union a struktura v unionu /jiné/ (11)

```
    // Pokrac. z predchozi strany
    printf(" == union u4 == \n");
    pu4 = &u4;
    u4.x1 = 33;
    pu4->x3 = 55;
    printf("\n u4.x1 = %d, (*pu4).x2 = %d, pu4->x3 = %d \n\n\n",
           u4.x1, (*pu4).x2, pu4->x3);

    printf(" Konec programu \n\n");
    return (EXIT_SUCCESS);
}// main() END
```

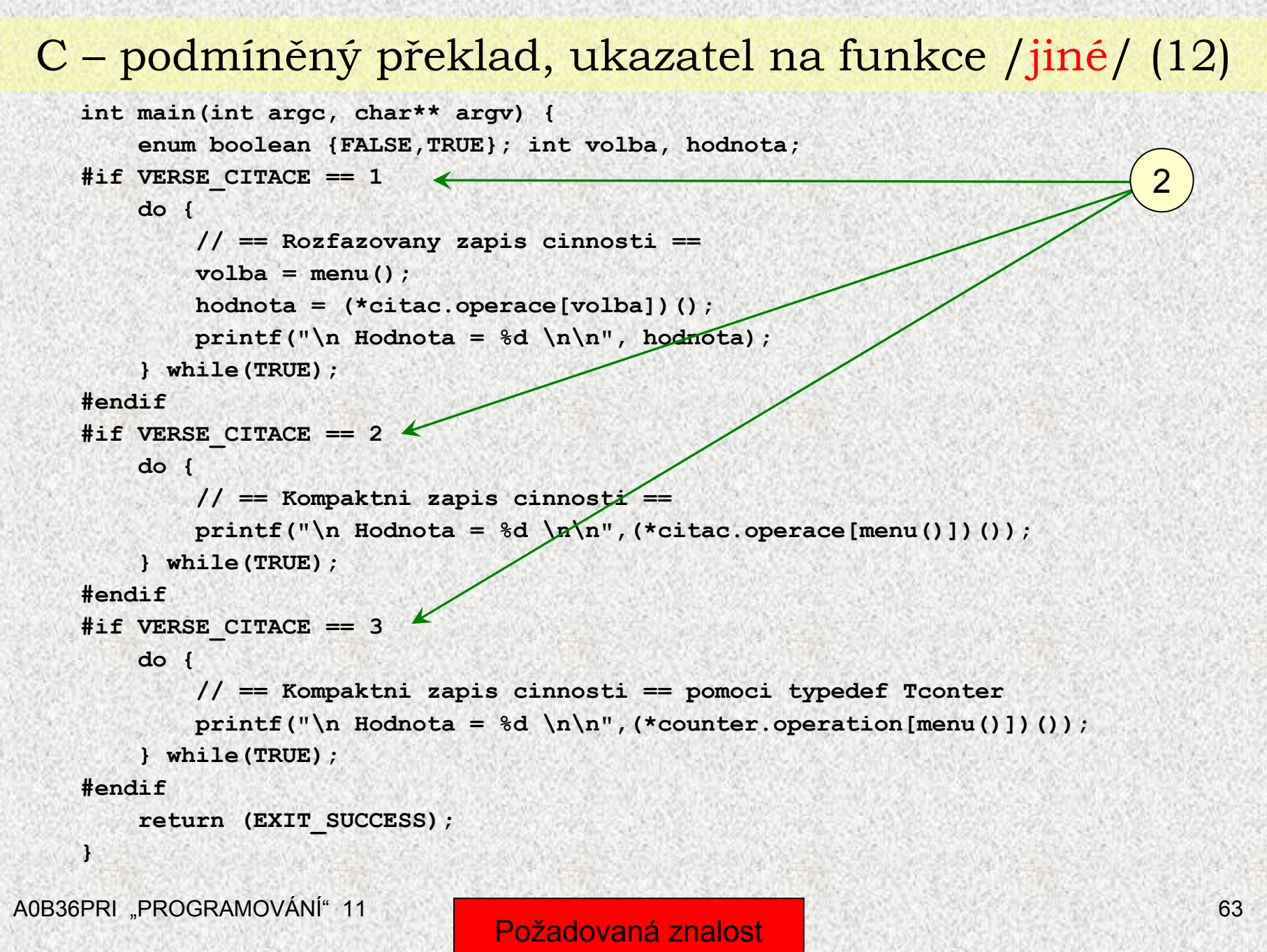
C – podmíněný překlad, ukazatel na funkce /jiné/ (12)

```
// Programovaci styly - Citac /proceduralni styl - reseni 4
// struktura, ukazatel na funkci, pole ukazatelu na funkci
// Podmineny preklad

#define VERSE_CITACE 3 // Platne: 1,2,3

int konec (void);    // Function prototypes
int zvetsi (void);
int zmensi (void);
int nastav (void);
int hodnota (void);

struct {
    int hodnota ;
    int (*operace[5])(void); // array of function pointers
} citac = {0,&konec,&zvetsi,&zmensi,&nastav,&hodnota};

typedef struct {
    int value ;
    int (*operation[5])(void); // array of function pointers
}Tcounter;

Tcounter counter = {0,&konec,&zvetsi,&zmensi,&nastav,&hodnota};
```

Požadovaná znalost

C – podmíněný překlad, ukazatel na funkce /jiné/ (12)

```
int main(int argc, char** argv) {
    enum boolean {FALSE,TRUE}; int volba, hodnota;
#if VERSE_CITACE == 1
    do {
        // == Rozfazovany zapis cinnosti ==
        volba = menu();
        hodnota = (*citac.operace[volba])();
        printf("\n Hodnota = %d \n\n", hodnota);
    } while(TRUE);
#endif
#if VERSE_CITACE == 2
    do {
        // == Kompaktni zapis cinnosti ==
        printf("\n Hodnota = %d \n\n",(*citac.operace[menu()])());
    } while(TRUE);
#endif
#if VERSE_CITACE == 3
    do {
        // == Kompaktni zapis cinnosti == pomoci typedef Tconter
        printf("\n Hodnota = %d \n\n",(*counter.operation[menu()])());
    } while(TRUE);
#endif
    return (EXIT_SUCCESS);
}
```



Požadovaná znalost

C – podmíněný překlad, ukazatel na funkce /jiné/ (12)

```
int zvetsi (void){
    citac.hodnota++;
    return(citac.hodnota);
}

int zmensi (void){
    citac.hodnota--;
    return(citac.hodnota);
}

int nastav (void){
    citac.hodnota=0;
    return(citac.hodnota);
}

int hodnota (void){
    return(citac.hodnota);
}
```


Pro zájemce

C – podmíněný překlad, ukazatel na funkce /jiné/ (12)

```
int menu (void) {
    enum boolean {FALSE,TRUE};
    int volba;
    do {
        printf(" 0. Konec \n");
        printf(" 1. Zvetsi \n");
        printf(" 2. Zmensi \n");
        printf(" 3. Nastav \n");
        printf(" 4. Hodnota \n");
        printf("\n Vase volba: ");
        if(!nextInt(&volba) || volba < 0 || volba > 4){
            printf("\n Nepovolena volba \n\n");
        }else{
            return(volba);
        }
    } while (TRUE);
}
```


Pro zájemce

C - proměnné, deklarace, definice

C - Deklarace vs. Definice:

- *Deklarace* určuje interpretaci a vlastnosti identifikátoru(ů)
- *Definice* je *deklarace včetně přidělení paměti* (memory allocation) proměnným, konstantám nebo funkcím

C - Syntaxe deklarace:

- *[specifikator_pametove_tridy] typ D1 [,D2,…];*

  kde:

  - specifikator_pametove_tridy - *extern, static, auto, register*
  - typ - *primitivní typy* (char,int,…), *void, enum, struct, union, typedef name*

    (+ *const*, *volatile*, *short*, *long*, *unsigned*)
  - D1 [,D2,…] - seznam deklarátorů
    - jednoduchý deklarátor:

      identifikátor + *počáteční_hodnota* (nepovinná)
    - složený deklarátor

      identifikátor + *počáteční_hodnota*+*symboly* *, [ ] , ( )

      (pointer, array, function)

C - proměnné, deklarace, definice

C - Příklady deklarace proměnných:

- jednoduché deklarace

```
char znak;
int i,j,k;
unsigned int suma=0;                    // +inicializace
const int k1=250;                       // +konstanta
static double rychlost,zrychleni;
extern unsigned char status;
```

- složené deklarace

```
int pole[30];                           // jednorozm.pole
unsigned int poleA[]={10,20,30,40};     // +inicializace
char s1[]="Ahoj";                       // +inicializace
char s2[5]={'A','h','o','j','\0'};      // +inicializace
double da[2][3];                        // dvourozm.pole
char *sPtr;                             // ukazatel (pointer)
```

C - proměnné, paměťová třída

C - Deklarace, definice - umístění ve zdrojovém kódu:

- Mimo těla funkcí (tj mimo všechny bloky)
- Na začátek bloku {….}

C - Proměnná musí být deklarována dříve než se použije:

C - Pamětová třída proměnné (Storage Class):

- Paměťová třída *definuje*:
  - *Životnost* proměnné (*storage duration*) *během provádění* programu
  - *Viditelnost* proměnné (*scope*) z různých míst *daného modulu* programu
  - *Viditelnost* proměnné (*linkage*) z *ostatních modulů* programu
    - Pozn: zde modul programu=samostatně překládaný zdrojový soubor
- Paměťová třída *je definována*:
  - *Polohou* deklarace proměnné *v modulu* programu
  - *Specifikatorem_paměťové_třídy (storage class specifier)* (*auto*, *register*, *static*, *extern*)

C - proměnné, paměťová třída

C - Životnost proměnné (storage duration):

- *statická*
  - vytvoření - *jednou na začátku* běhu programu
  - zrušení - *po ukončení* programu
  - zajistí - překladač + [static]
  - inicializace - *explicitní* nebo *automatická na 0* / NULL.

- *dočasná*
  - vytvoření - automaticky *vždy při vstupu* programu *do bloku* {…}
  - zrušení - automaticky *vždy po ukončení bloku*
  - zajistí - překladač + [auto]
  - inicializace - *explicitní* nebo *nedefinovaná* !!

  Pozn: proměnné se nazývají *automatické* a zakládají se v zásobníku.

- *volitelná*
  - vytvoření - *na žádost* programátora za běhu programu
  - zrušení - *na žádost* programátora za běhu programu
  - zajistí - programátor voláním funkcí pro správu paměti
  - inicializace - *explicitní* nebo *nedefinovaná* !!

  Pozn: proměnné se nazývají *dynamické* a přidělují se z haldy (heap)

C - proměnné, paměťová třída

C - Viditelnost proměnné (scope) v modulu (souboru):

- Globální v modulu
  - zajistí se - polohou deklarace v modulu + static

- Lokální v bloku {….}
  - zajistí se - polohou deklarace v bloku {…}

  Pozn: viz následující příklady

C - Viditelnost proměnné (linkage) v ostatních modulech (celý program):

- Globální v programu
  - zajistí se - polohou deklarace v bloku + [extern]

- Lokální v modulu
  - zajistí se - polohou deklarace v bloku + static

  Pozn: viz následující příklady

C - proměnné, paměťová třída

C - Specifikátory paměťové_třídy (Storage Class Specifiers - SCS):

SCS	Význam
Auto	Definuje proměnnou jako dočasnou (automatickou). Lze použít jen pro proměnné deklarované uvnitř funkce. Implicitní nastavení je auto
Register	Doporučuje překladači umístit proměnnou do registru procesoru (rychlost přístupu). Ten nemusí vyhovět (nemá-li volné registry). Jinak jako proměnné auto.
static	Deklaruje proměnnou jako statickou uvnitř bloku {..}. Vně bloku (kde je proměnná implicitně statická) omezuje její viditelnost na modul.
extern	Rozšiřuje viditelnost statických proměnných z modulu na celý program

- Pozn: v deklaraci proměnné lze uvést vždy jen jeden SCS

C - proměnné, paměťová třída

C - Životnost a viditelnost proměnných:

LOKÁLNÍ V BLOKU
LOKÁLNÍ V MODULU
GLOBÁLNÍ V PROGRAMU

př:

```
int i;
static int max1=500;
extern int j;


int funkce1(int x, int y)
 {
 int a1;
 int a2=30;
 static int a3=50;
   {
    int b1;
    b1=funkce2(4);
    }
  }

static int funkce2(int m)
 {. . .}
```

STATICKÁ

```
extern int i;
static int max1=200;
int j;
int funkce1(int,int);

void main(void)
 {
 int test;
 test=funkce1(4,max1);
 //test=funkce2(I); //nelze
 funkce3();
 }

static void funkce3(void)
 {
 char c='A';
 . . .
 }
```

C - proměnné, pole (array)

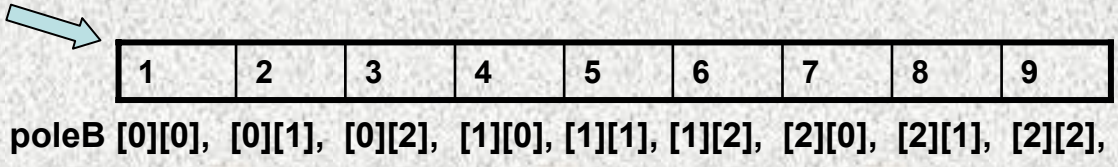
C - Pole (array):

- 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 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* prvkek takového pole je vždy *'\0' (null char)*
- C - *nekontroluje* za běhu programu, zda vypočítaný *index je platný*

C - proměnné, pole (array)

C - Deklarace pole (array):

```
př:
char poleA [10];  // jednorozmerne pole z prvku char
int poleB[3][3];  // dvourozmerne pole z prvku int
Uložení v paměti
```



C - Inicializace pole:

```
př:
double x[3]={0.1,0.5,1.3];
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:

```
př:
    ai[1][3]=15*2;
```

C - proměnné, ukazatel (pointer)

C - Ukazatel (pointer):

- *Ukazatel* je proměnná obsahující *adresu jiné proměnné* (nepřímá adresa)
- *Ukazatel* má též *typ* proměnné na kterou může *ukazovat*
  - ukazatel na char
  - ukazatel na int atd.
- Ukazatel může být též bez typu *(void*), pak může obsahovat *adresu libovolné proměnné*. Její velikost pak nelze z vlastností ukazatele určit
- *Neplatná adresa* v ukazateli má hodnotu konstanty *NULL*
- C za běhu programu *nekontroluje* zda adresa v ukazateli *je platná*

- *Adresa proměnné* se zjistí adresovým operátorem ***&***
- K proměnné na kterou *ukazatel ukazuje se přistoupí* operátorem nepřímé adresy ******* *(hvězdička*)

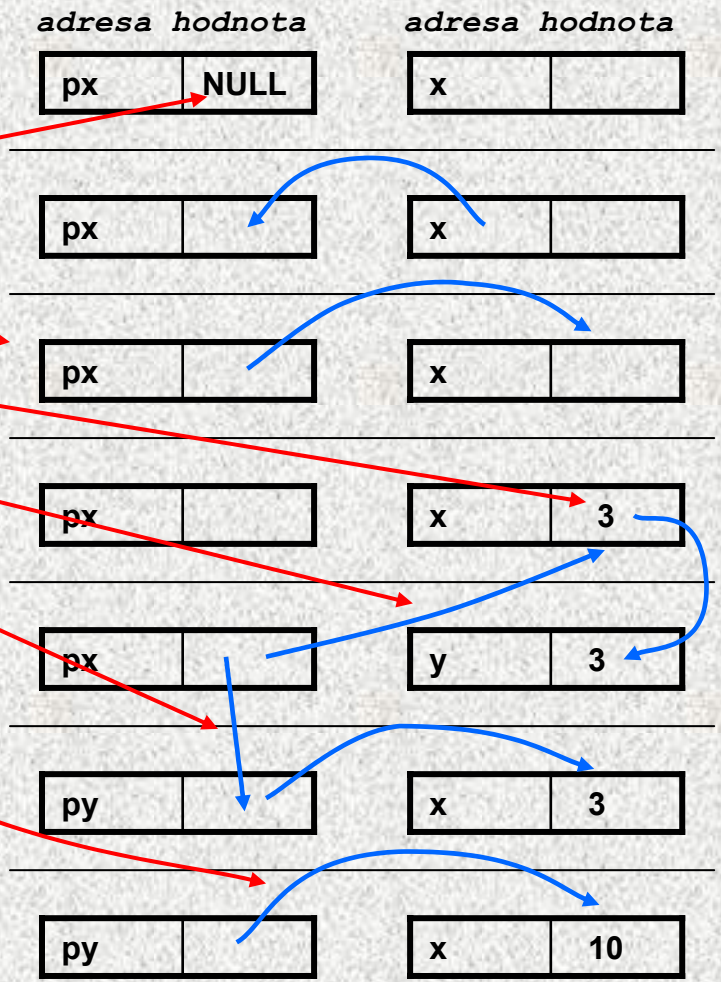
- Ukazatel může obsahovat též *adresu funkce* (ukazatel na funkci)
- Pomocí *ukazatele na funkci* lze *funkci volat*, včetně předání parametrů
- Pomocí *ukazatele* lze předávat parametry funkci *odkazem (call by reference*) (*standardní metoda v C* pro ostatní případy je *volání hodnotou (call by value))*

C - proměnné, 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;
```

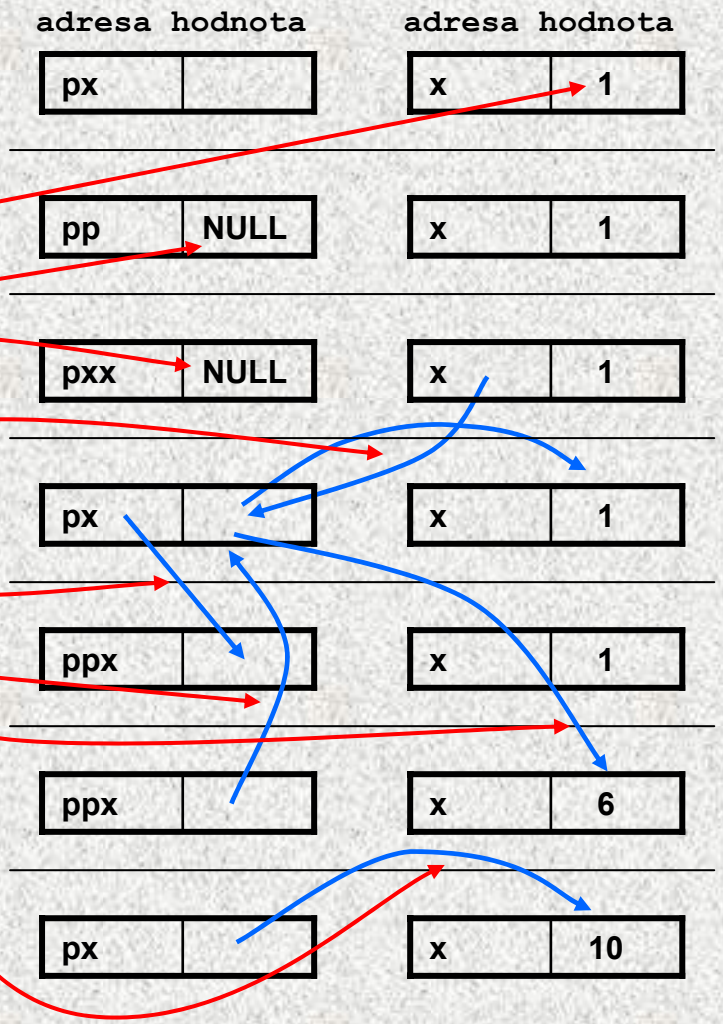


C - proměnné, ukazatel (pointer)

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 - proměnné, ukazatel (pointer)

C - Ukazatel (pointer) - aritmetické operace:

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

C - Ukazatel na pole (pointer to array):

- *Aritmetické* operace jsou *užitečné* když ukazatel *ukazuje na pole*
- Jméno pole je konstantní ukazatel na počátek pole (na prvek a[0])
- 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)*

```
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[5]
        px=&a[3];             // px ukazuje na a[3]
```

C - proměnné, ukazatel (pointer)

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

- V složitějších deklaracích ukazatelů mohou být nezbytné závorky

```
př:  double a[10];          // Pole z prvku double
     double (*pa)[10];      // Ukazatel na pole 10 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 - Ukazatel na funkci:

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

C - proměnné, struktura (struct)

C - Struktura (struct):

- Struktura je *množina prvků* (proměnných), které *nemusí* být *stejného typu*
- *Skladba* struktury je *definovaná uživatelem* jako *nový typ* sestavený z již definovaných typů

- K prvkům struktury *se přistupuje tečkovou notací*
- K prvkům struktury je *možné přistupovat i pomocí ukazatele* na strukturu operátorem **->**

- Struktury *mohou být vnořené*

- Pro struktury *stejného typu* je definována operace *přiřazení struct1=struct2* (pro proměnné typu pole přímé přiřazení není definováno, jen po prvcích)

- Struktury (jako celek) *nelze porovnávat* relačním operátorem ==

- Struktura *může* být *do funkce předávána hodnotou i odkazem*
- Struktura *může* být *návratovou hodnotou funkce*

C - proměnné, struktura (struct)

C - Struktura (struct) - sablona (tag):

```
př:  struct Tid{                  // <==== Tid=jmeno sablony (tag)
          char jmeno[20];         // Prvky struktury, pole
          char adresa[50];        // - " -              pole
          long int telefon;       // - " -              int
          };
     struct Tid sk1,skAvt[20];    // struktura, pole struktur
     struct Tid *pid;             // ukazatel na strukturu
     sk1.jmeno="Jan Novak";       // teckova notace
     sk1.telefon=123456;
     skAvt[0].jmeno="Jan Novak";  // prvek pole
     skAvt[3].telefon=123456;
     pid=&sk1;                    // do pid adresa struktury
     pid->jmeno="Jan Novak";      // odkaz pomoci ->
     pid->telefon=123456;
     (*pid).jmeno="Jan Novak";    // odkaz pomoci *
     (*pid).telefon=123456;
```

C - proměnné, struktura (struct)

C - Struktura (struct) - nový typ (typedef):

```
př:  typedef struct {                  // <==== Pomoci Typedef
            char jmeno[20];            // Prvky struktury, pole
            char adresa[50];           // - " -              pole
            long int telefon;          // - " -              int
            }Tid,*Tidp;
     Tid sk1,skAvt[20];                // struktura, pole struktur
     Tidp pid;                         // ukazatel na strukturu
     sk1.jmeno="Jan Novak"; // teckova notace
     sk1.telefon=123456;
     skAvt[0].jmeno="Jan Novak";       // prvek pole
     skAvt[3].telefon=123456;
     pid=&sk1;                         // do pid adresa struktury
     pid->jmeno="Jan Novak";           // odkaz pomoci ->
     pid->telefon=123456;
     (*pid).jmeno="Jan Novak";         // odkaz pomoci *
     (*pid).telefon=123456;
```

C - proměnné, struktura (struct)

C - Struktura (struct) - inicializace:

```
př:
     struct Tid{                        // <==== Tid=jmeno sablony (tag)
        char jmeno[20];                    // Prvky struktury, pole
        char adresa[50];                   // - " -                  pole
        long int telefon;                  // - " -                  int
     };

     struct Tid sk1={"Jan Novak",
                     "Na kopecku 23",
                      123456};
```

C - proměnné, sdílená paměť (union)

C - Sdílená paměť (union):

- **Union je *množina prvků* (proměnných), které *nemusí být stejného typu***
- ***Prvky* unionu *sdílejí* společně *stejná paměťová místa* (překrývají se)**
- ***Velikost* unionu je dána velikostí *největšího z jeho prvků***

- Skladba unionu je *definovaná uživatelem jako nový typ* sestavený z již definovaných typů

- K prvkům unionu *se přistupuje tečkovou notací*

př:

```
union Tnum{                 // <==== Tnum=jmeno sablony (tag)
        long n;
        double x
        };
union Tnum nx;              // nx - promenna typu union Tnum
nx.n=123456789L;            // do n hodnota long
nx.x=2.1456;                // do x hodnota double (prekryva n)
```

C – funkce - struktura

C - Funkce (function):

- C je *modulární jazyk* a *funkce* je jeho *hlavním stavebním blokem*
- Každý program v C obsahuje *minimálně* funkci *main()*
- Běh programu *začíná* na *začátku* funkce *main()*
- C používá *prototypu funkce* k deklaraci *informací* nutných pro překladač, aby mohl *správně přeložit* volání funkcí i v případě, že *definice funkce je umístěna dále v kódu* modulu nebo je *jiném modulu*
- *Prototyp* funkce (function prototype) se *skládá* pouze z *hlavičky funkce*, (odpovídá interface v Javě)
- *Definice* funkce *obsahuje hlavicku* funkce a její *tělo*
- Syntaxe definice funkce:

```
typ_navratove_hodnoty jmenoFunkce(seznam_parametru)
  {
  definice_lokalnich_promennych
  seznam_prikazu
  }
```

- Parametry se do funkce předávají hodnotou (call by value), parametrem může být i ukazatel (pointer). Ten pak dovolí předávat parametry i odkazem.

C – funkce - vlastnosti

C - Funkce (function) pokrač:

- *C nepovoluje* funkce *vnořené do jiných funkcí* (lokální funkce ve funkci)

- Jména funkcí jsou implicitně *extern*, a mohou se *exportovat* do *ostatních modulů* (samostatně překládaných souborů)
- Specifikátor *static* před jménem funkce *omezí viditelnost jejího jména pouze* na *daný modul (lokální funkce modulu*)

- *Formální* parametry funkce jsou *běžné lokální proměnné* inicializované skutečnými parametry při volání funkce
- C *dovoluje rekurzi*, lokální proměnné jsou pro *každé jednotlivé* volání *zakládány znovu* (v zásobníku). Kód funkce v C *reentrantní* (reentrant).

- Funkce *nemusí mít* žádné *vstupní parametry*, zapisuje se funkceX*(void*)
- Funkce *nemusí vracet žádnou funkční hodnotu*, pak je návratový typ void (je to *procedura*)

C – funkce - prototyp

C - Funkce (function) pokrač:

př:

```
int max(int a,int b);  // Prototyp funkce

int i=10,j=20,k;

void main(void)
 {                     // Zacatek programu
  k=max(i,j);
  . . .
 }
                       // Definice funkce
int max(int a, int b)  // Formalni parametry a,b jsou
 {                     //  tez lokalni promenne
  if(a > b)return(a);
  return(b);
 }
```

C – funkce - parametry

C - Funkce (function) pokrač:

př:

```
double sumaReciprocN(int n);  // Prototyp funkce
double vysledek;

void main(void)
  {                           // Zacatek programu
  vysledek=sumaReciprocN(50);
  . . .
  }
                              // Soucet rady 1/n
double sumaReciprocN(int n)   // Definice funkce
  {
  double x;
  int i;
  if(n <= 0)return(0);
  for(x=1,i=1;i < n; i++)
  x += 1/((double)i);
  return(x);
  }
```

C - Funkce (function) pokrač:

př:

```
#include <stdio.h>                  // Standardni knihovna
int faktorial(int n);               // Prototyp funkce
int vysledek;

void main(void)
  {                                 // Zacatek programu
  if((vysledek=faktorial(50))==0)
    printf("Chybne zadani");
  else
    printf("Faktorial=%d",vysledek);
  }
                                    // Faktorial
int faktorial(int n)                // Definice funkce
 {
  if(n < 0)return(0);
  if(n == 0)return(1);
  return(n * faktorial(n-1)); // Rekurse
 }
```

C – funkce – parametr - pole

C - Funkce (function) pokrač:

př:

```
void 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[]
void secti(int a[], int n)    // Definice funkce
 {
  int i;
  long suma=0;
  for(i=0;i<n;i++)
    suma += a[i];
  return(suma);
 }
```

C – funkce – parametr - 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 (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
  int i;
  for(i=0;s[i]==p[i];i++)
    if(s[i]=='\0' || p[i]=='\0')
         if(s[i]=='\0' && p[i]=='\0')return(0);
         else                       return(s[i]-p[i]);
}
```

C - funkce

C - Funkce (function) pokrač:

př:

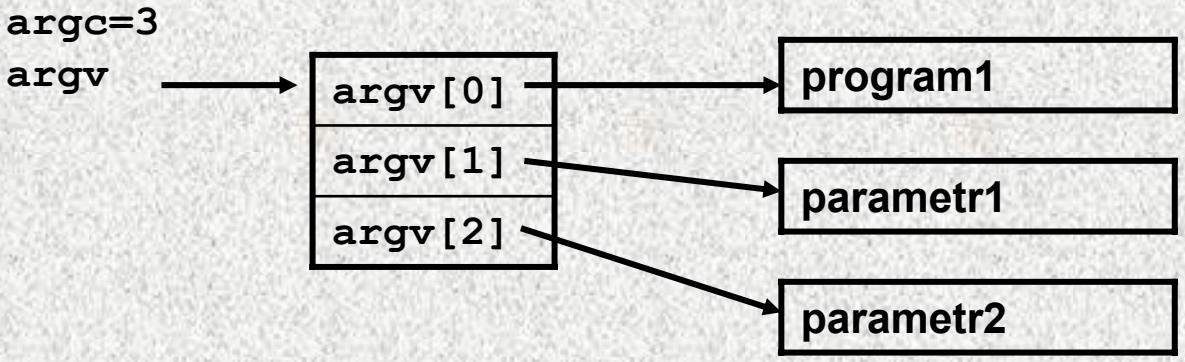
```
#include <stdio.h>                          // Standardni knihovna
int porovnejString(char *s, char *p);// Prototyp fce
char s[]="Nazdar";
char *g="Ahoj";

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
  for(i=0; *s == *p; i++)
    if(*s =='\0' || *p=='\0')
      if(*s =='\0' && *p=='\0')return(0);
      else                        return(*s - *p);
  }
```

C – funkce – parametry main()

C - Funkce (function) – vstupni parametry funkce main():

- Funkce *main()* přebírá parametry z *příkazového řádku*
- main() má *dva vstupní* parametry:

```
argc    – počet parametrů na příkazovém řádku
*argv[] – pole ukazatelů na příkazy příkazového řádku

int main(int argc, char **argv);
int main(int argc, char *argv); // Totez

př:
   příkazový radek ve tvaru c:\>program1 parametr1 parametr2
```



C - příkazy preprocesoru

C - Preprocesor (Preprocessor):

- *Zdrojový text* programu v C je před vlastní překladem *předzpracován*
- Předzpracování *provede Preprocesor* takto:
  - Odstraní komentáře
  - Nahradí makra jejich definicí
  - Vykoná ostatní direktivy preprocesoru
- Každá *direktiva* preprocesoru *začíná na samostatném* řádku *znakem* #
- Příliš *dlouhou* příkazovou řádku je možné *rozdělit znakem* \

- *Direktivy* preprocesoru umožňují:
  - *Definovat makra* a *rušit* jejich definice
    - #define, #undef
  - *Testovat* zda je *makro definováno*
    - defined
  - *Vkládat* do zdrojového textu *hlavičkové* soubory
    - #include
  - Řídit *podmíněný překlad*
    - -#if,#elif,#else,#endif,#ifdef,#ifndef
  - *Definovat nové* příkazy preprocesoru *závislé na implementaci*
    - #pragma

C - standardní knihovny

C – Standardní knihovny (ANSI C library – Standard Library) :

- Vlastní jazyk *C neobsahuje* žádné prostředky pro vstup a výstup dat, složitější matematické operace, práci s řetězci, třídění, blokové přesuny dat v paměti, práci s datem a časem, komunikaci s operačním systémem,správu paměti pro dynamické přidělování, vyhodnocení běhových chyb (run-time errors) apod.

- Tyto a další funkce jsou však *obsaženy ve standardních knihovnách* (*ANSI C Library*) dodávaných s překladači jazyka C.

- Uživatel *dostává* k dispozici *přeložený kód knihoven* (který se připojuje – linkuje k uživatelovu kódu) a *hlavičkové soubory* (headers) s *prototypy funkcí, novými typy, makry a konstantami*

- *Hlavičkové* soubory (obdoba interface v Javě) se *připojují k uživatelovu kódu* direktivou preprocesoru *#include* <…>.

- *Je zvykem*, že *hlavičkové* soubory mají *rozšíření *,h*, např. stdio.h

C - standardní knihovny

C – Standardní knihovny (ANSI C library – Standard Library) pokrač:

- Standardní knihovny jsou *rozděleny* do následujících částí:
  (uvedeno pro ANSI C95):

- Vstup a výstup (formátovaný i neformátovaný)
  - stdin.h

- Rozsahy čísel jednotlivých typů
  - limits.h

- Matematické funkce
  - stdlib.h
  - math.h

- Zpracování běhových chyb (run-time errors)
  - errno.h
  - assert.h

C - standardní knihovny

C – Standardní knihovny (ANSI C library – Standard Library) pokrač:

- Standardní knihovny jsou rozděleny do následujících částí: (uvedeno pro ANSI C95) pokrač:

- Klasifikace znaků (typ char)
  - ctype.h

- Práce s řetězci (string handling)
  - string.h

- Internacionalizace (adaptace pro různé jazykové mutace)
  - locale.h

- Vyhledávání a třídění
  - stdlib.h

- Blokové přenosy dat v paměti
  - string.h

C - standardní knihovny

C – Standardní knihovny (ANSI C library – Standard Library) pokrač:

- Standardní knihovny jsou rozděleny do následujících částí:
  (uvedeno pro ANSI C95) pokrač:

- Správa paměti (Dynamic Memory Management)
  - stdlib.h

- Datum a čas
  - time.h

- Komunikace s operačním systémem
  - stdlib.h
  - signal.h

- Nelokální skok (lokální je součástí jazyka, viz goto)
  - setjump.h

A0B36PRI - PROGRAMOVÁNÏ

Jazyk C

Část III.

Konec

České vysoké učení technické Fakulta elektrotechnická