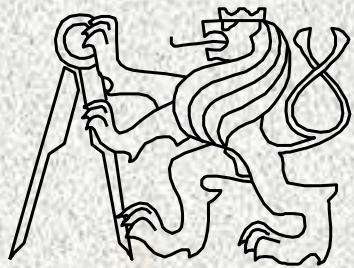


Jazyk C – operátory, příkazy



A0B36PR2-Programování 2

Fakulta elektrotechnická

České vysoké učení technické

C – Obsah přednášky

- Porovnání JAVA vs „C“
 - Metody a funkce
 - Konstrukce metod a funkcí
 - Vstupní parametry funkce
 - Návratová hodnota funkce
 - Rozklad problému na podproblémy (funkce)
 - Výstup údajů na obrazovku
 - Vstup čísel z klávesnice
 - Iterace a rekurze
 - Programovací styly
 - Naivní styl
 - Procedurální styl

C - Obsah

Přehled

- Výrazy a operátory
- Operátory - priorita, asociativita
- Operátory - počet operandů
- Operátory - aritmetické
- Operátory - přiřazovací
- Operátory - relační
- Operátory - logické
- Operátory - bitové logické
- Operátory - přístupu do paměti
- Operátory – ostatní
- Změna typu (přetypování)
- Příkazy, blok (složený příkaz)

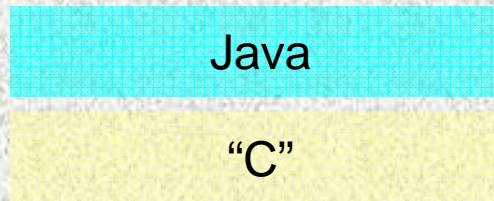
C - Obsah

Přehled

- Příkazy, řízení běhu programu
- Příkazy, podmíněný příkaz
- Příkazy, programový přepínač
- Příkazy, cyklus for(...)
- Příkazy, cyklus while()
- Příkazy, cyklus do..while()
- Příkazy, continue
- Příkazy, break
- Příkazy, return
- Příkazy, goto

Použití barevné značení

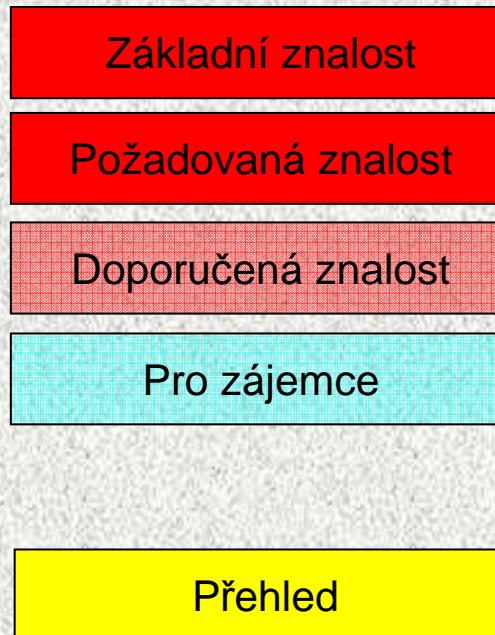
Rozlišení jazyka



Srovnatelné vlastnosti



Rozlišení stupně znalostí



Pomůcka pro “C”

JAVA – statické metody /podobné/ (1)

- Faktoriál pomocí metod (rozklad na dílčí problémy)

```
import java.util.*;
public class Faktorial {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        System.out.println (n + "! = " + faktorial(n));
    }
    static int ctiPrirozene() {
        Scanner sc = new Scanner(System.in);
        System.out.println ("zadejte přirozené číslo");
        int n = sc.nextInt();
        if(n < 1) {
            System.out.println (n + " není přirozené číslo");
            System.exit(0);
        }
        return n;
    }
    static int faktorial ( ) { . . . }
}
```

JAVA – statické metody /podobné/ (1)

- Faktoriál pomocí metod (rozklad na dílčí problémy) – pokrač.

```
static int faktorial(int n) {  
    int i = 1;  
    int f = 1;  
    while (i < n) {  
        i = i + 1;  
        f = f * i;  
    }  
    return(f);  
}
```




5

C – funkce /podobné/ (1)

- Faktoriál pomocí funkcí (rozklad na dílčí problémy)

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

```
int ctiPrirozene(void); // Prototyp funkce
int faktorial(int f);   // pouze deklarace
int main(int argc, char** argv) {
    int n, f;
    n = ctiPrirozene();
    printf("\n %d! = %d \n\n", n, faktorial(n));
    return (EXIT_SUCCESS);
}
```



C – funkce /podobné/ (1)

- Faktoriál pomocí funkcí (rozklad na dílčí problémy) – pokrač.

```
int ctiPrirozene(void) {  
    int n;  
    printf("Zadejte prirodzene cislo (max 12) n = ");  
    scanf("%d", &n);  
    if(n < 1) {  
        printf("n = %d neni prirodzene cislo \n\n", n);  
        exit(EXIT_FAILURE);  
    }  
    return(n);  
}
```

- není static (vše je static, C nemá třídy)
- Prázdný seznam parametrů se deklaruje void
- **EXIT_FAILURE, EXIT_SUCCESS**

C – funkce /podobné/ (1)

- Faktoriál pomocí funkcí (rozklad na dílčí problémy) – pokrač.

```
int faktorial(int n) {  
    int i = 1;  
    int f = 1;  
    while (i < n) {  
        i = i + 1;  
        f = f * i;  
    }  
    return(f);  
}
```




5

JAVA – statické metody /podobné/ (2)

- Největší společný dělitel - NSD

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

    public static void main(String[] args) {
        int a=sc.nextInt();
        int b=sc.nextInt();
        System.out.println("Nejvetsi spolecny delitel" +
                           a + ", " + b + "je " +
nsd(a,b));
    }
    static int nsd(int x, int y) {
        while (x != y)
            if (x > y) x = x - y;
            else y = y - x;
        return x;
    }
}
```



C – funkce /podobné/ (2)

- Největší společný dělitel - NSD

```
#include <stdio.h>
#include <stdlib.h>
int nsd(int x, int y); // Function prototype

int main(int argc, char** argv) {
    int m, n, delitel;
    scanf("%d", &m);
    scanf("%d", &n);
    printf("\nNejmensi spol.delitel = %d \n\n",nsd(m,n));
    return (EXIT_SUCCESS);
}

int nsd(int x, int y) {
    while (x != y)
        if (x > y) x = x - y;
        else y = y - x;
    return x;
}
```

1

JAVA – vstup čísel z klávesnice /jiné/ (3)

```
import java.util.*;
public class CtiKlavesnici {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        System.out.println ("n = " + n);
        double d = sc.nextDouble();
        System.out.println ("d = " + d);
    }
}
```

The diagram consists of seven numbered circles (1-7) on the right side of the slide. Green arrows point from these circles to specific lines of code in the Java program:

- Circle 1 points to the first line: `import java.util.*;`
- Circle 2 points to the second line: `public class CtiKlavesnici {`
- Circle 3 points to the third line: `Scanner sc = new Scanner(System.in);`
- Circle 4 points to the fourth line: `int n = sc.nextInt();`
- Circle 5 points to the fifth line: `System.out.println ("n = " + n);`
- Circle 6 points to the sixth line: `double d = sc.nextDouble();`
- Circle 7 points to the seventh line: `System.out.println ("d = " + d);`

C – vstup čísel z klávesnice /jiné/ (3)

- Funkce scanf() s ošetřením chybných vstupních dat

```
int nextInt(int *cislo);
```

```
int nextDouble(double *cislo);
```

- ```
int main(int argc, char** argv) {
 int n;
 double d;
 if (!nextInt(&n)) {
 printf("Neni cele cislo \n");
 return (EXIT_FAILURE);
 }
 if (!nextDouble(&d)) {
 printf("Neni realne cislo \n");
 return (EXIT_FAILURE);
 }
 printf("\n = %d, d = %f \n", n, d);
 return(EXIT_SUCCESS);
}
```

1

3

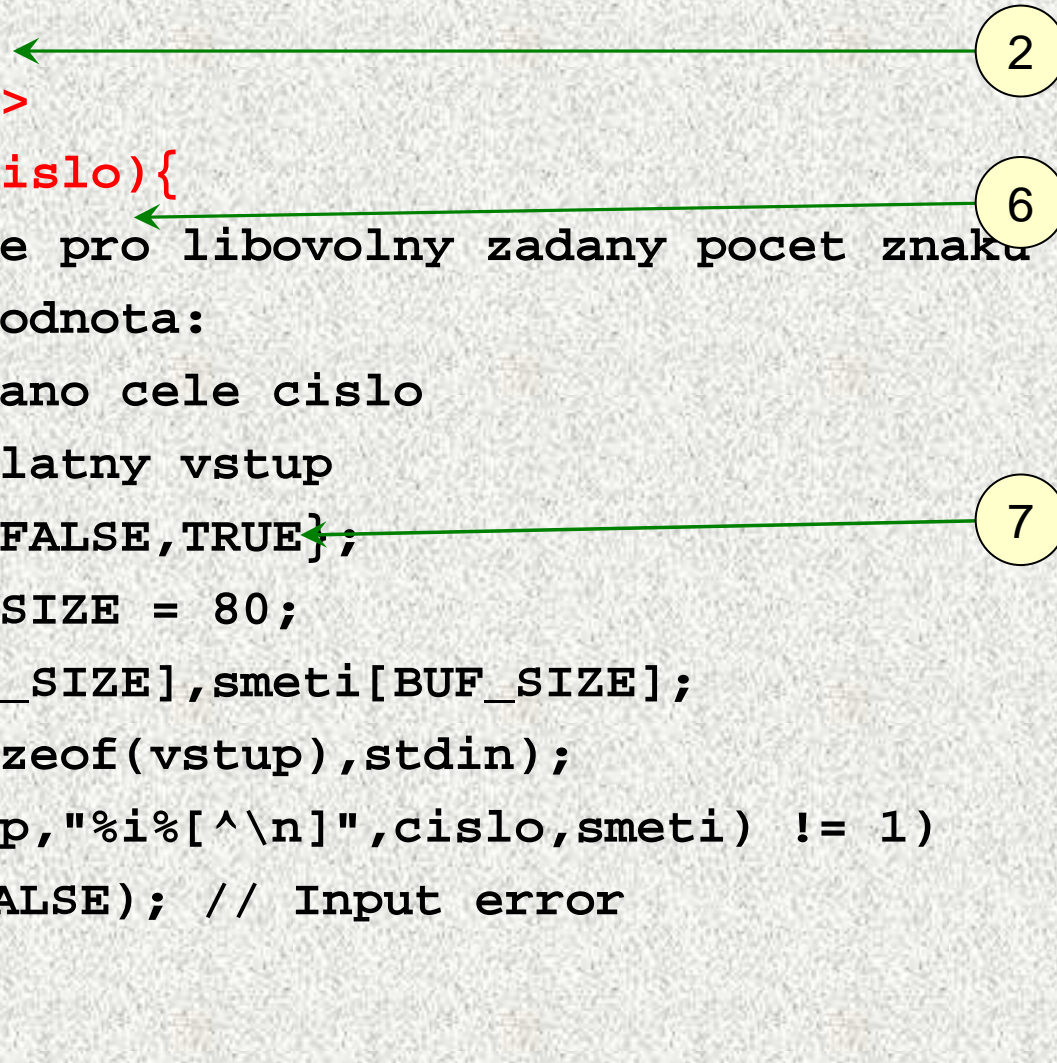
4

5

## C – vstup čísel z klávesnice – nextInt() /jiné/ (3)

- Funkce scanf() s ošetřením chybných vst.dat - pokrač.

```
#include <stdio.h>
#include <stdlib.h>
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);
}
```



# C – vstup čísel z klávesnice – nextDouble()

*/jiné/* (3)

- Funkce scanf() s ošetřením chybných vstupních dat – pokrač.

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

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

```
int nextDouble(double *cislo){
```

7

```
 // === 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);
```

```
}
```



# JAVA - Hra NIM /podobné/ (4)

```
import java.util.*;
public class Nim {
 static int pocet; // aktuální počet zápalek
 static boolean stroj; // =true znamená, že bere počítač
 public static void main(String[] args) {
 zadaniPoctu();
 stroj = false; // zacina hrac
 do {
 if (stroj) bereStroj();
 else bereHrac();
 stroj = !stroj;
 } while (pocet > 0);
 if(stroj) System.out.println("Vyhrál jsem");
 else System.out.println("Vyhrál jste, gratuluji");
 }
 static void zadaniPoctu() { ... }
 static void bereHrac() { ... }
 static void bereStroj() { ... }
}
```

1

2

3

4

5

# JAVA - Hra NIM /podobné/ (4)

```
static void zadaniPoctu() {
 Scanner sc = new Scanner(System.in);
 do {
 System.out.println("Zadejte pocet zapalek (15-
35)");
 pocet = sc.nextInt();
 } while (pocet < 15 || pocet > 30);
}
static void bereStroj() {
 System.out.println("Pocet zapalek " + pocet);
 int x = (pocet - 1) % 4;
 if (x == 0) x = 1;
 System.out.println("Odebiram " + x);
 pocet -= x;
}
```

6

7

# JAVA - Hra NIM /podobné/ (4)

```
static void bereHrac() {
 Scanner sc = new Scanner(System.in);
 int x; boolean chyba;
 do {
 chyba = false;
 System.out.println("Pocet zapalek " + pocet);
 System.out.println("Kolik odeberete");
 x = sc.nextInt();
 if (x<1) {
 System.out.println("Prilis malo");
 chyba = true;
 }
 else
 if (x>3 || x>pocet) {
 System.out.println("Prilis mnoho");
 chyba = true;
 }
 } while (chyba);
 pocet -= x;
}
```

8

9

# C - Hra NIM /podobné/ (4)

```
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1 // konstanty
#define FALSE 0
int zadaniPoctu(void); // deklarace
int bereStroj(int pocet);
int bereHrac(int pocet);

int main(int argc, char** argv) {
 int pocet, stroj=FALSE;
 pocet = zadaniPoctu();
 do {
 if (stroj) pocet = bereStroj(pocet);
 else pocet = bereHrac(pocet);
 stroj = !stroj;
 } while (pocet > 0);
 printf("Vsechny zapalky odebrany \n");
 if (!stroj) printf("Vyhrál jsem \n\n");
 else printf("\nVyhrál jste, gratuluji \n\n");
 return (EXIT_SUCCESS);
}
```

## C - Hra NIM /podobné/ (4)

```
int zadaniPoctu(void) {
 int pocet = 0;
 do {
 printf("Zadejte pocet zapalek (od 15 do 35) = "
 scanf("%d", &pocet);
 } while (pocet < 15 || pocet > 30);
 return (pocet);
}
int bereStroj (int pocet) {
 int x;
 printf("Pocet zapalek = %d \n\n", pocet);
 x = (pocet - 1) % 4;
 if(x == 0) x = 1;
 printf("Odebiram %d \n", x);
 pocet -= x;
 return(pocet);
}
```

6

7

# C - Hra NIM /podobné/ (4)

```
int bereHrac(int pocet) {
 int x, chyba;
 do {
 chyba = FALSE;
 printf("Pocet zapalek = %d \n\n", pocet);
 printf("Kolik odeberete (1..3) ? ");
 scanf("%d", &x);
 if (x < 1) {
 printf("Prilis malo \n");
 chyba = TRUE;
 } else
 if (x > 3 || x > pocet) {
 printf("Prilis mnoho \n");
 chyba = TRUE;
 }
 } while (chyba);
 pocet -= x;
 return (pocet);
}
```

```
graph TD
 8((8)) --> scanf[scanf("%d", &x);]
 scanf --> while[while (chyba);]
 while --> 8
 8 --> 9((9))
 9 --> return[return (pocet);]
```

# JAVA - Hanojské věže /podobné/ (5)

```
import java.util.*;
```

```
public class HanojskeVeze {
 public static void main(String[] args) {
 Scanner sc = new Scanner(System.in);
 int pocetDisku = sc.nextInt();
 prenesVez(pocetDisku, 1, 2, 3);
 }

 static void prenesVez(int vyska, int odkud,
 int kam, int pomoci) {
 if (vyska > 0) {
 prenesVez(vyska-1, odkud, pomoci, kam);
 System.out.printf
 ("Prenes disk z %d na %d \n", odkud,
kam);
 prenesVez(vyska-1, pomoci, kam, odkud);
 }
 }
}
```

1

2

3

4

5

## C - Hanojské věže /podobné/ (5)

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

// Hanojske veze
int zadaniPoctuDisku(void); // deklarace
void prenesVez(int vyska, int odkud, int kam, int
pomoci);
int main(int argc, char** argv) {
 int pocetDisku;
 printf("Hanojske veze \n\n");
 pocetDisku = zadaniPoctuDisku();
 prenesVez(pocetDisku, 1, 2, 3);
 printf("\n");
 return (EXIT_SUCCESS);
}
```

1

2

3

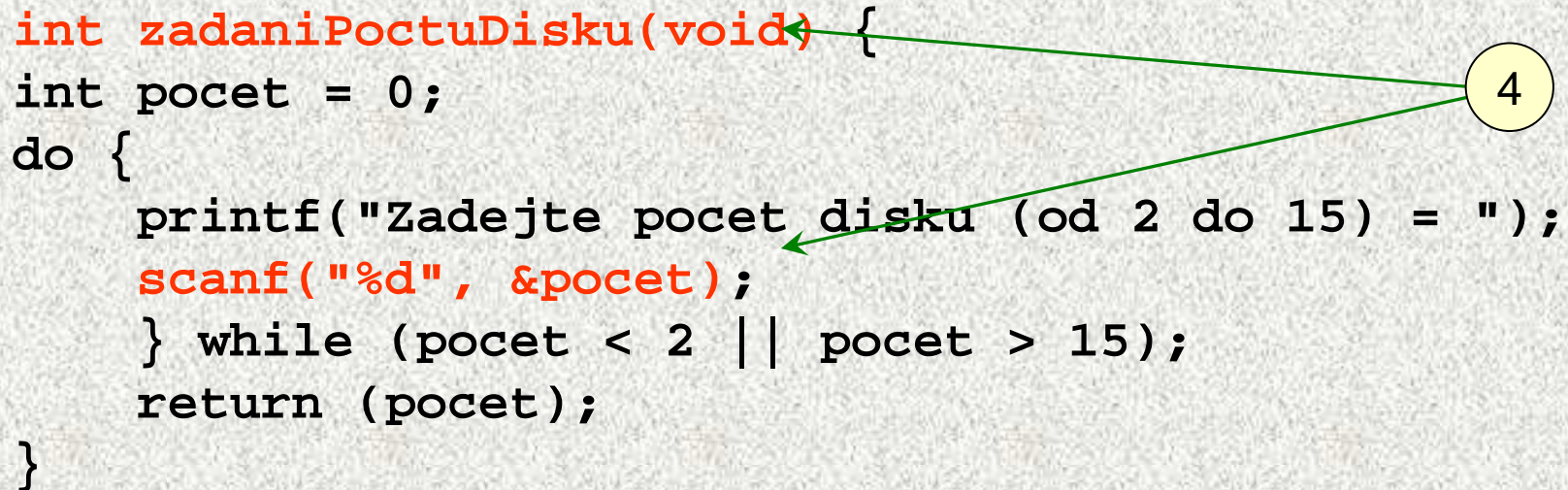
4

5



## C - Hanojské věže /podobné/ (5)

```
int zadaniPoctuDisku(void) {
 int pocet = 0;
 do {
 printf("Zadejte pocet disku (od 2 do 15) = ");
 scanf("%d", &pocet);
 } while (pocet < 2 || pocet > 15);
 return (pocet);
}
```

A diagram consisting of a yellow circle containing the number '4' on the right side. Two green arrows originate from this circle. One arrow points to the opening curly brace of the function definition 'int zadaniPoctuDisku(void) {'. The other arrow points to the 'scanf' function call within the 'do' loop of the same function.

```
void prenesVez(int vyska, int odkud,
 int kam, int pomoci) {
 if (vyska > 0) {
 prenesVez(vyska-1, odkud, pomoci, kam);
 printf("Prenes disk z %d na %d \n", odkud, kam);
 prenesVez(vyska-1, pomoci, kam, odkud);
 }
}
```

# JAVA - Naivní styl – Čítač /podobné/ (6)

```
import java.util.*
public class Citac1 {
 final static int pocHodn = 0; static int hodn, volba;
 public static void main(String[] args) {
 Scanner sc = new Scanner(System.in);
 hodn = pocHodn;
 do {
 System.out.println("Hodnota = " + hodn);
 System.out.println
 ("0) Konec\n1) Zvětši\n2) Zmenši\n3) Nastav");
 System.out.print("Vaše volba: ");
 volba = sc.nextInt();
 switch (volba) {
 case 0: break;
 case 1: hodn++; break;
 case 2: hodn--; break;
 case 3: hodn = pocHodn; break;
 default: System.out.println("Nedovolena volba");
 }
 } while (volba > 0);
 System.out.println("Konec");
 }
}
```

1  
2  
3  
4

# C - Naivní styl – Čítač /podobné/ (6)

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char** argv) {
 const int POC_HODNOTA = 0; int hodn, volba;
 hodn = POC_HODNOTA;
 do {
 printf("Hodnota = %d \n\n", hodn);
 printf("0) Konec\n1) Zvetsi\n2)
 Zmensi\n3) Nastav \n\n");
 printf("Vase volba ");
 scanf("%d", &volba);
 printf("\n");
 switch (volba) {
 case 0: break;
 case 1: hodn++; break;
 case 2: hodn--; break;
 case 3: hodn = POC_HODNOTA; break;
 default: printf("Nepovolena volba");
 }
 } while (volba > 0);
 printf("Konec\n\n");
 return (EXIT_SUCCESS);
}
```

1

2

3

4

# JAVA - Procedurální styl – Čítač /jiné/ (7)

```
import java.util.*;

public class Citac {
 final static int POC_HODN = 0; // třídní konstanta
 static int hodn; // třídní proměnná

 public static void main(String[] args) {
 int volba;
 hodn = POC_HODN;
 do {
 System.out.println("Hodnota = " + hodn);
 volba = menu();
 if (volba > 0) operace(volba);
 } while (volba > 0);
 System.out.println("Konec");
 } // main() END
```

The diagram consists of six numbered circles (1-6) on the right side, with green arrows pointing to specific lines of code in the Java snippet above. Circle 1 points to the import statement. Circle 2 points to the class declaration. Circle 3 points to the first comment. Circle 4 points to the opening brace of the main method. Circle 5 points to the assignment of POC\_HODN to hodn. Circle 6 points to the menu() call in the do-while loop.

# JAVA - Procedurální styl – Čítač /jiné/ (7)



globální  
statická

```
static void operace(int op) { // pro všechny 3
operace
 switch (op) {
 case 1: hodn++; break;
 case 2: hodn--; break;
 case 3: hodn = POC_HODN; break;
 }
}

static int menu(){. . .}
} // class Citac END
```

7

8

# JAVA - Procedurální styl – Čítač /jiné/ (7)

```
static int menu(){
 Scanner sc = new Scanner(System.in);
 int volba;
 do {
 System.out.println("0. Konec");
 System.out.println("1. Zvětši");
 System.out.println("2. Zmenši");
 System.out.println("3. Nastav");
 System.out.print("Vaše volba: ");
 volba = sc.nextInt();
 if (volba < 0 || volba > 3) {
 System.out.printf("\n Nepovolená volba
\n\n");
 volba = -1;
 }
 } while(volba < 0);
 return volba;
}
```

9

10

# C - Procedurální styl – Čítač /jiné/ (7-1)

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

1

```
int menu (void);
int operace (int volba);
```

// deklarace

2

```
int main(int argc, char** argv) {
 int volba;
 printf("Hodnoda = %d \n\n", operace(4));
 do {
 volba = menu();
 if (volba > 0)
 printf("\nHodnota = %d \n\n", operace(volba));
 } while(volba > 0);
 printf("\nKonec \n\n");
 return (EXIT_SUCCESS);
}
```

6

# C - Procedurální styl – Čítač /jiné/ (7-1)



Lokální  
statická

```
int operace (int op) { // pro všechny 4 operace
#define POC_HODNOTA 0;
static int hodn = POC_HODNOTA;
switch (op) {
case 1: hodn++; break;
case 2: hodn--; break;
case 3: hodn = POC_HODNOTA; break;
case 4: break;
}
return(hodn);
}
```

3

4

8

5

7




# C - Procedurální styl – Čítač /jiné/ (7-1)

```
int menu (void) {
 int volba;`
 do {
 printf("0. Konec \n");
 printf("1. Zvetsti \n");
 printf("2. Zmensi \n");
 printf("3. Nastav \n");
 printf("4. Hodnota \n");
 printf("\nVase volba: ");
 scanf("%d", &volba);
 if (volba < 0 || volba > 4) {
 printf("\n Nepovolená volba \n\n");
 volba = -1;
 }
 } while (volba < 0);
 return volba;
}
```

9

10

  
Neošetřené  
chyby,  
lépe viz (7-2)

# C - Procedurální styl – Čítač /jiné/ (7-2)

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

int menu (void);
int operace (int volba);
int nextInt(int *cislo);

int main(int argc, char** argv) {
 int volba;
 printf("Hodnoda = %d \n\n", operace(4));
 while ((volba = menu()) > 0) {
 printf("\n Hodnota = %d \n\n", operace(volba));
 }
 printf("\n Konec \n\n");
 return (EXIT_SUCCESS);
}
```

10

## C - Procedurální styl – Čítač /jiné/ (7-2)

```
int operace (int op) { // pro všechny 4 operace
 #define POC_HODNOTA 0;
 static int hodn = POC_HODNOTA;
 switch (op) {
 case 1: hodn++; break;
 case 2: hodn--; break;
 case 3: hodn = POC_HODNOTA; break;
 case 4: break;
 }
 return(hodn);
}
```

- int operace() shodné se (7-1)

# C - Procedurální styl – Čítač /jiné/ (7-2)

```
int menu (void) {
 enum boolean {FALSE,TRUE};
 int volba;
 do {
 printf(" 0. Konec \n");
 printf(" 1. Zvetsti \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);
}
```

Ošetřené chyby



- Ošetřeny nesprávné vstupní údaje

# C - Procedurální styl – Čítač /jiné/ (7-2)

```
int nextInt(int *cislo) { ← 10
 // Stav: odladeno
 // === 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);
}
```

- Ošetřeny nesprávné vstupní údaje

# C – Výrazy a operátory

## **C - Výraz se skládá z operátorů a operandů**

- Nejjednodušší výraz tvoří jen konstanta, proměnná, volání funkce
- Výraz sám může být operandem
- Výraz má *typ a hodnotu* (pouze výraz typu *void* hodnotu *nemá*)
- Výraz zakončený středníkem je příkaz

př. výrazy:

|                                              |                            |
|----------------------------------------------|----------------------------|
| <code>4*315</code>                           | <code>// typ int</code>    |
| <code>1.0+sin(x)</code>                      | <code>// typ double</code> |
| <code>srand((unsigned)time(NULL))</code>     | <code>// typ void</code>   |
| <code>(int*)malloc(count*sizeof(int))</code> | <code>// typ int*</code>   |

- Ve výrazu s více operátory určuje postup výpočtu *priorita operátorů*,
- V případě operátorů se stejnou prioritou pak *jejich asociativita*  
(L->R zleva doprava, R->L zprava doleva)
- Postup výpočtu výrazu lze změnit použitím kulatých závorek

# C - Operátory - priorita, asociativita

## C - Priorita a asociativita operátorů

- Pořadí vyhodnocování výrazu se řídí prioritou a asociativitou
- $18/2*3+11$  je 38 ....  $(18/2)*3 + 11$

| Priorita | Operátor                           | Asociativita | Priorita | Operátor                             | Asociativita |
|----------|------------------------------------|--------------|----------|--------------------------------------|--------------|
| 1        | () [] ->                           | L->R         | 9        | ^                                    | L->R         |
| 2        | ! ~ ++ -- + -<br>(type) * & sizeof | R->L         | 10       |                                      | L->R         |
| 3        | * / %                              | L->R         | 11       | &&                                   | L->R         |
| 4        | + -                                | L->R         | 12       |                                      | L->R         |
| 5        | << >>                              | L->R         | 13       | ?:                                   | R->L         |
| 6        | < <= > >=                          | L->R         | 14       | = += -= *= /= %=<br>&= ^=  = <<= >>= | R->L         |
| 7        | == !=                              | L->R         | 15       | ,                                    | L->R         |
| 8        | &                                  | L->R         |          |                                      |              |

# C - Operátory - počet operandů

## **C - Operátory mají 1, 2 nebo 3 operandy**

- Všechny unární operátory mají stejnou prioritu
- Znak `-`, `+`, `*`, `&` jsou podle počtu operandů operátory unární nebo binární
- V C je jediný ternární operátor `?:` (otazník-dvojtečka)(conditional operator)
- Pořadí vyhodnocení operandů není definováno (mimo popsané případy jako `++x`, `x--`, ...)

př:

```
funkce1() + funkce2() // Ktera funkce bude volana
 // prvni neni definovano
```

- `int a = 3;`
- `a+=a++ + ++a * a++;`
- JAVA – v `a` bude 31
- C, C++ - 26,31,... cokoli, ale programátor se nedoví, že jde o nejednoznačný zápis



# C - Operátory - aritmetické

## C - Aritmetické operátory

| Operátor                | Význam        | Příklad        | Výsledek                                                                      |
|-------------------------|---------------|----------------|-------------------------------------------------------------------------------|
| *                       | Násobení      | $x * y$        | Součin $x$ a $y$                                                              |
| /                       | Dělení        | $x / y$        | Podíl $x$ a $y$                                                               |
| %                       | Dělení modulo | $x \% y$       | Zbytek po dělení $x$ a $y$                                                    |
| +                       | Sčítání       | $x + y$        | Součet $x$ a $y$                                                              |
| -                       | Odčítání      | $x - y$        | Rozdíl $x$ a $y$                                                              |
| +                       | Kladné znam.  | $+x$           | Hodnota $x$                                                                   |
| <sup>(unary)</sup><br>- | Záporné znam. | $-x$           | Hodnota $-x$                                                                  |
| ++                      | Inkrementace  | $++x$<br>$x++$ | $x=x+1$ před výpočtem výrazu s $x$<br>Výpočet výrazu s $x$ a následně $x=x+1$ |
| --                      | Dekrementace  | $--x$<br>$x--$ | $x=x-1$ před výpočtem výrazu s $x$<br>Výpočet výrazu s $x$ a následně $x=x-1$ |

- Operandů aritmetických operátorů - *libovolný aritmetický typ*
- Výjimka: % (zbytek po dělení) - jen operandů typu *int*

# C - Operátory - přiřazovací

| Operátor | Význam               | Příklad | Výsledek                                                                              |
|----------|----------------------|---------|---------------------------------------------------------------------------------------|
| =        | Jednoduché přiřazení | x = y   | Přiřadí hodnotu y do x                                                                |
| op=      | Složené přiřazení    | x +=y   | x op=y je ekvivalentní x = x op y, kde op je binární aritmetický nebo bitový operátor |

Levý operand *musí být l-value* (location-value, left-value) - tj musí představovat paměťové místo pro uložení výsledku.

- Jednoduché přiřazení - povolené operandy
  - dva operandy - *aritmetického typu*
  - dva operandy typu - *struct nebo union* (stejných typů)
  - dva operandy typu - *pointer (stejného typu)* nebo *pravý operand=NULL* nebo *jeden pointer typu void*
- Operandů různých typů se převedou na *typ levého operandu*
- Přiřazení se vyhodnocuje R->L (zprava doleva)
  - Př: a=b=100; // stejné jako a=(b=100);

# C - Operátory - relační

## C - Relační operátory

| Operátor | Význam           | Příklad  | Výsledek                                         |
|----------|------------------|----------|--------------------------------------------------|
| <        | Menší než        | $x < y$  | 1 když $x$ je menší než $y$ , jinak 0            |
| <=       | Menší nebo rovno | $x <= y$ | 1 když $x$ je menší nebo rovno než $y$ , jinak 0 |
| >        | Větší než        | $x > y$  | 1 když $x$ je větší než $y$ , jinak 0            |
| >=       | Větší nebo rovno | $x >= y$ | 1 když $x$ je větší nebo rovno než $y$ , jinak 0 |
| ==       | Rovná se         | $x == y$ | 1 když se $x$ rovná $y$ , jinak 0                |
| !=       | Nerovná se       | $x != y$ | 1 když se $x$ nerovná $y$ , jinak 0              |

- Výraz s relačními operátory (relace) je typu *int*
- Výsledek je 1 nebo 0 - s významem 1 ~ true, 0 ~ false
- Povolené operandy relace:
  - - dva operandy *aritmetického typu*
  - - dva *ukazatele (pointers) shodného typu* nebo jeden
  - z nich *NULL* nebo typu *void*

# C - Operátory - logické

| Operátor | Význam      | Příklad | Výsledek                                          |
|----------|-------------|---------|---------------------------------------------------|
| &&       | Logické AND | x && y  | 1 když x ani y není rovno 0, jinak 0              |
|          | Logické OR  | x    y  | 1 když alespoň jeden z x, y není roven 0, jinak 0 |
| !        | Logické NOT | ! x     | 1 když x je roven 0, jinak 0                      |

- Logické operátory spojují výsledky relací do logického výrazu
- Povolené operandy:
  - *aritmetické typy*
  - *typ pointer*
- *Výsledek je 1 nebo 0 - s významem 1 ~ true, 0 ~ false*
- ***Ve výrazech s operátory && a || se vyhodnotí\_nejdříve levý operand,  
pokud je pak výsledek známý, pravý operand se nevyhodnocuje  
(zkrácené vyhodnocování logického výrazu)***

# C - Operátory - bitové

| Operátor | Význam       | Příklad      | Výsledek                                       |
|----------|--------------|--------------|------------------------------------------------|
| &        | Bitové AND   | $x \& y$     | 1 když x i y je rovno 1 (bit po bitu)          |
|          | Bitové OR    | $x   y$      | 1 když alespoň jeden z x, y je 1 (bit po bitu) |
| ^        | Bitové XOR   | $x \wedge x$ | 1 když pouze jeden z x, y je 1 (bit po bitu)   |
| ~        | Bitové NOT   | $\sim x$     | 1 když x je rovno 0 (bit po bitu)              |
| <<       | Posun vlevo  | $x \ll y$    | Posun x o y bitů vlevo                         |
| >>       | Posun vpravo | $x \gg y$    | Posun x o y bitů vpravo                        |

- Bitové operátory vyhodnocují operandy bit po bitu
- Operátory posunu (shift operators) posouvají celý bitový obraz o zvolený počet bitů vlevo nebo vpravo
- Při posunu vlevo - jsou uvolněné bity zleva - plněny 0
- Při posunu vpravo - jsou uvolněné bity zprava
  - - u čísel kladných nebo typu unsigned - plněny 0 (logical shift right)
  - - u čísel záporných - buď plněny 0 (logical shift right)
  - - nebo plněny 1 (arithmetic shift right)
  - (závisí na implementaci překladače)

# C - Operátory - přístupu do paměti

| Operátor | Význam               | Příklad | Výsledek                                             |
|----------|----------------------|---------|------------------------------------------------------|
| &        | Adresa proměnné      | &x      | Konstantní pointer na x                              |
| *        | Nepřímá adresa       | *p      | Proměnná nebo funkce adresovaná ukazatelem p         |
| [ ]      | Prvek pole           | x [ i ] | *(x+i), prvek pole x s indexem i                     |
| .        | Prvek struct / union | s.x     | Prvek x struktury / unionu s                         |
| ->       | Prvek struct / union | p->x    | Prvek x struktury / unionu s adresovaný ukazatelem p |

- Operandem operátoru & *nesmí být* - *bitové pole* a proměnná třídy *register*
- Operátor *nepřímé adresy* \* - umožňuje přístup pomocí *ukazatele (pointer)*

```
př: int a,*pa; // proměnná int a ukazatel na int
```

```
pa=&a; // adresa a do pa
```

```
*pa=45; // totez jako a=45
```

```
př: double a[10],*pa; // proměnná int a ukazatel na int
```

```
pa=a; // adresa pole a[] do pa (neni &)
```

```
*(pa+3)=12; // totez jako a[3] nebo pa[3]
```

# C - Operátory - ostatní

## C - Ostatní operátory (nepatří do žádné z předchozích kategorií):

| Operátor | Význam                | Příklad   | Výsledek                           |
|----------|-----------------------|-----------|------------------------------------|
| { }      | Volání funkce         | f1(x,y)   | Zavolej funkci f1 s parametry x, y |
| (type)   | Přetypování (cast)    | (int) x   | Změň typ x na int                  |
| sizeof   | Velikost prvku (byte) | sizeof(x) | Velikost x (v byte)                |
| ?:       | Podmíněný příkaz      | x?y:z     | Když x!=0 pak y jinak z            |
| ,        | Postupné vyhodnocení  | x, y      | Vyhodnoť nejdříve x pak y          |

- Operátor *přetypování* je možné použít - *pouze na operandy skalárních typů*
- Operandem *sizeof()* může být - *jméno typu nebo výraz*
- Podmíněný operátor *?:* (otazník-dvojtečka)

*př:*

```
max = x > y ? x : y;
```

*je totéž jako:*

```
if (x > y) max=x;
```

```
else max=y;
```

# C - změna typu (přetypování)

## **C - Přetypování (cast):**

- Změna typu za běhu programu - přetypování
- Přetypování:
  - *Explicitní* - zapisuje programátor

```
prom_noveho_typu = (novy_typ) prom_puvodniho_typu;
```

**př:**

```
int a;
float x = (float)a;
```

- *Implicitní* - provede překladač automaticky

- Pokud nový typ může reprezentovat původní hodnotu, přetypování ji vždy zachová



# C - změna typu (přetypování)

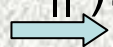
## C - Přetypování (*cast*) pokrač.:

- Priorita typu *int* (integer promotion):
  - Operandy typů *char*, *unsigned char*, *short*, *unsigned short* a *bitová pole* mohou být použity všude tam, kde je *povolený typ int nebo unsigned int*. Takové operandy jsou automaticky *přetypovány* na *int* nebo *unsigned int*.
    - C vždy očekává hodnoty alespoň typu *int*.

př:            `char c;`

`x=c+'0'; // c (typu char) je před výpočtem přetypována na int`

- Implicitní aritmetické konverze typu:
  - Pokud mají operandy binárních operátorů rozdílné typy i po konverzi na *int* (integer promotion) proběhne konverze podle typu který, leží *více vpravo* v následující tabulce (konverze neprobíhá pro operátory `=`, `&&`, `||`):



|     |              |      |               |       |        |             |
|-----|--------------|------|---------------|-------|--------|-------------|
| int | unsigned int | long | unsigned long | float | double | long double |
|-----|--------------|------|---------------|-------|--------|-------------|

V přiřazení (`=`) je výsledný typ roven typu na *levé straně*

*Ukazatel typu void* může být přetypován na *libovolný jiný typ ukazatele a naopak*.

# C - příkazy, blok (složený příkaz)

## **C - Příkaz:**

- příkaz = výraz zakončený středníkem
- příkaz tvořený pouze středníkem = prázdný příkaz

## **C - Blok (též složený příkaz):**

- *Blok = seznam deklarácí + seznam příkazů*
- Uvnitř bloku musí *deklarace předcházet příkazy*
- *Začátek a konec* bloku je vymezen složenými závorkami {.....}
- Bloky mohou být *vnořené* do jiného bloku
- *Vnořený blok* může začínat *kdekoliv uvnitř bloku*
- Syntaxe bloku (vnořených bloků):

```
{ // Zatek bloku
deklarace
prikazy
 { // Zatek vnoreneho bloku
 deklarace
 prikazy
 } // Konec vnoreneho bloku
} // Konec bloku
```

# C - příkazy, blok

## C - Blok a vnořený blok:

```
{ // Zacatek bloku 1
int i; // Deklarace
double sum=0.0;
for(i=0;i<10;i++) // Prikazy
{ // Vnoreny blok 2
sum+=i; // Prikazy
if(i==5)
{ // Vnoreny blok 3
int j; // Deklarace
sum+=2; // Prikazy
for(j=0;j<100;j++)
; // Prazdny prikaz
} // Konec bloku 3
} // Konec bloku 2
} // Konec bloku 1
```

# C - příkazy, řízení běhu programu

## **C - Příkazy pro řízení běhu programu:**

- Podmíněné větvení programu:
  - podmíněný příkaz - `if ( ), if ( )...else`
  - programový přepínač - `switch ( )`
- Cykly:
  - `for ( )`
  - `while ( )`
  - `do...while ( )`
- Nepodmíněné větvení programu:
  - `continue`
  - `break`
  - `return`
  - `goto`

# C - příkazy, podmíněný příkaz

## C - Podmíněný příkaz *if( ) else*, větvení programu do 2 směrů:

- *if (vyraz) prikaz1; [else prikaz2;]*
- Když je vyraz  $\neq 0$  provede se *prikaz1* jinak *prikaz2*
- Část *else...* je nepovinná
- Podmíněné příkazy mohou být vnořené

Př:

```
max=0;
if (x > y) max=x;
```

Př:

```
if (z == 200)
{
 // if 1
 if (x > y) max=x; // if 2
 else max=y; // else 2
}
//
else
//
 max=0;
```

# C - příkazy, programový přepínač

## **C - Programový přepínač *switch()*, větvení programu do *n* směrů:**

- *switch* (vyraz) příkaz;
- Hodnota *vyraz* je porovnána s *n* konstantními výrazy typu *int* (case konst\_x: .....)
- Všechny konst\_x musí být navzájem různé
- Hodnota *vyraz* musí být celočíselná
- Pokud je nalezena shoda, program pokračuje od tohoto místa dál, dokud *nenajde příkaz break* nebo *konec příkazu switch*
- *switch* se opustí příkazem *break*
- Pokud se shoda nenajde, program pokračuje sekcí *default* (je nepovinná)
- Pokud se shoda nenajde a sekce *default není zařazena*, celý *switch se vynechá*
- Příkazy *switch mohou být vnořené*
- Sekce *default* nemusí být zařazena jako poslední (je to ale zvykem)

# C - příkazy, programový přepínač

## C - Programový přepínač `switch()` pokrač.:

*př:*

```
switch (command) // Zacatek prepinnacle
{ //
 case 'g': // Zacatek sekce 1
 case 'G': akce1(); // |
 break; // Konec sekce 1
 case 's': // Zacatek sekce 2
 case 'S': akce2(); // |
 break; // Konec sekce 2
 case 'X': akce3(); // Zacatek sekce 3
 break; // Konec sekce 3
 default: chyba(); // Zacatek sekce default
 tiskniNapovedu(); // |
 break; // Konec sekce default
}
```

# C - příkazy, cyklus for(...)

## **C - Příkaz cyklu for( ; ; ):**

- *for ([vyraz1];[vyraz2];[vyraz3]) prikaz;*
- Cyklus for() používá řídicí proměnnou a probíhá následovně
  - inicializace (jednou před začátkem cyklu)
  - test řídicího výrazu
  - aktualizace proměnných na konci každého běhu cyklu
- Uvedené činnosti definují výrazy 1,2,3 v hlavičce cyklu
- Výrazy *výraz1* a *výraz3* mohou být *libovolného* typu
- *výraz2* je *řídicí* a musí být *skalárního* typu
- Libovolný z výrazů 1,2,3 lze vynechat
- Po *vynechání* řídicího výrazu 2 se cyklus bude provádět *nepodmíněně*
- Cyklus lze *nuceně opustit* příkazem *break*
- Část těla cyklu lze *vynechat* příkazem *continue* (od continue do konce těla)



# C - příkazy, cyklus for(...)

## C - Příkaz cyklu for( ; ; ) pokrač:

*př:*

```
#define DELAY 20
int i;
for(i=DELAY;i>0;i--) // Hlavicka cyklu
 ; // Telo cyklu (zde prazdne)
```

*př:*

```
#define MAXCOUNT 50
#define MAXSUMA 110
int i,suma;
for(i=0,suma=0;(i<MAXCOUNT)||(suma>MAXSUMA);i++)
{
 suma+=5; // Telo cyklu
}
```

# C - příkazy, cyklus for(...)

**C - Příkaz cyklu for( ; ; ) pokrač:**

```
for(;;)
 ; // Nekonecny cyklus

#define MAXCOUNT 50
#define MAXSUMA 110
int i,suma=0;
for(i=0;i<MAXCOUNT;i++)
{
 if (i==10)
 continue; // 1x vynech zbytek cyklu
 suma+=5;
 if (suma>MAXSUMA)
 break; // Ukonci cyklus predcasne
}
```

# C - příkazy, cyklus while( )

## **C - Příkaz cyklu while( ):**

- *while (vyraz1) prikaz;*
- Cyklus while() probíhá následovně
  - 1) vyhodnot' *vyraz1*,
  - 2) pokud je *vysledek !=0* proved' *příkaz*, jinak ukonči cyklus
  - 3) pokračuj bodem 1)
- Řídící *vyraz1* se vyhodnocuje *na začátku cyklu*, pokud je *výsledek ==0* cyklus se neprovede *ani jednou*
- Řídící *vyraz1* se *musí aktualizovat v těle cyklu*, jinak je cyklus *nekonečný*  
*př:*

```
int i=10,suma=0;
while (i > 0)
{
 suma+=i;
 i--; // Aktualizace ridici promenne
}
```

# C - příkazy, cyklus do..while( )

## **C - Příkaz cyklu do...while( ):**

- *do prikaz while vyraz1 ;*
- Cyklus do..while() probíhá následovně
  - 1) proved' *příkaz*
  - 2) vyhodnot' *vyraz1*,
  - 3) pokud je *vysledek ==0* ukonči cyklus, jinak pokračuj bodem 1)
- *Řídící vyraz1* se vyhodnocuje *na konci cyklu*, tělo cyklu se vždy provede *nejméně jednou*
- *Řídící vyraz1* se *musí aktualizovat v těle cyklu*, jinak je cyklus *nekonečný*

```
int i=10,suma=0;
do // Zavoroky bloku {...} zde nepovinne

 suma+=i;
 i--; // Aktualizace řídící proměnné
while(i > 0);
```

# C - příkazy, continue

## **C - Příkaz návratu na řídicí výraz *continue*:**

- *continue*;
- Příkaz *continue* lze použít pouze v těle cyklů
  - *for()*
  - *while()*
  - *do..while()*
- Příkaz *continue* způsobí vynechání zbylé části těla cyklu a nové vyhodnocení řídicího výrazu cyklu

*př:*

```
int i=10,suma=0;
do
 i--; // Aktualizace řídicí proměnné
 if(i == 4)continue; // 1x vynech zbytek cyklu
 suma+=i;
while(i > 0)
```

# C - příkazy, break

## **C - Příkaz nuceného ukončení cyklu *break*:**

- *break*;
- Příkaz *break* lze použít pouze v těle cyklů
  - *for()*
  - *while()*
  - *do..while()*
- a v těle programového přepínače *switch()*
- Příkaz *break* způsobí opuštění těla cyklu nebo těla *switch()*, program pokračuje prvním následujícím příkazem

*př:*

```
int i=10,suma=0;
while (i > 0)
{
 if(suma > 5)break; // Nucene opusteni cyklu
 suma+=i;
 i--; // Aktualizace ridici promenne
}
```

# C - příkazy, return

## **C - Příkaz ukončení funkce *return*:**

- *return* vyraz;
- Příkaz *return* lze použít pouze v těle funkce
- *return* ukončí funkci, vrátí návratovou hodnotu funkce určenou *hodnotou* vyraz a předá řízení volající funkci
- Příkaz *return* lze použít v těle funkce *vícekrát*
- U funkce typu *void f1()* nahrazuje uzavírací závorka těla funkce příkaz *return*

```
int maxPlusDve(int a, int b)
{
 if(a > b)
 return(a+2); // Vystup z funkce (zavorky
 // nepovinne)
 return(b+2); // Vystup z funkce
}
```

# C - příkazy, goto

## **C - Příkaz nepodmíněného lokálního skoku goto:**

- *goto* navesti;
- Příkaz *goto* lze použít pouze v těle funkce
- Cíl skoku musí ležet uvnitř stejné funkce (lokální skok)
- *goto* předá řízení na místo určené návěštím *navesti*
- Skok *goto* nesmí směřovat dovnitř bloku, který je vnořený do bloku, kde je *goto* umístěno

př:

```
int hledejMax(int *p)
{
 for(. . .)
 for(. . .)
 {
 if(. . .)goto error; // Ven z vnitřního bloku
 }
 return(. . .);
error: // Cíl skoku uvnitř funkce
 return(. . .);
}
```