



# Řídicí struktury v programování

---

Jitka Kreslíková, Aleš Smrčka

2023

Fakulta informačních technologií  
Vysoké učení technické v Brně

IZP – Základy programování

# Řídicí struktury v programování

---

☐ Funkce

☐ Příkazy

Konvence:

 Zjednodušení kódu z prostorových důvodů. V reálu **nepoužívat!**  
 Např. chybí ošetření chybového stavu, nevhodná konstrukce, atd.



# Funkce

---

- ❑ Obecný formát programu.
- ❑ Deklarace funkce (prototyp).
- ❑ Definice funkce (implementace).
- ❑ Volání funkcí.
- ❑ Funkce s parametry.



# Oběcný formát programu

```
/* zde vložit hlavičkové soubory */  
/* zde umístit případné prototypy funkcí */  
/* globální definice */
```

```
návratový-typ funkce1(seznam-parametrů)
```

vstupní/výstupní  
parametry

```
{
```

```
    /* tělo funkce1 */
```

```
}
```

```
·
```

```
·
```

```
/* globální definice */
```

```
návratový-typ funkceN(seznam-parametrů)
```

```
{
```

```
    /* tělo funkceN ..... */
```

```
}
```

```
int main(seznam-parametrů)
```

```
{
```

```
    /* tělo funkce main */
```

```
}
```

jméno funkce

datový typ, jehož  
hodnotu funkce vrátí



# Funkce - deklarace

## □ Deklarace funkce (prototyp).

*návratový-typ* *jménoFunkce* (*seznam-parametrů*);

*Příklad:*

```
double sqrt (double x) ;
```

- deklarace funkce před jejím použitím
- využití v hlavičkových souborech



# Funkce - definice

❑ Definice funkce (implementace funkce).

*návratový-typ* jménoFunkce ( *seznam-parametrů* )  
{  
    *seznam-příkazů*  
}

**Příklad:**

```
void mojeFunkce(void)
{
    printf("Toto je test.");
}
```

Její prototyp je:

```
void mojeFunkce(void);
```

není středník



# Funkce - volání

**Příklad: `double sqrt (double x);`**

```
#include <stdio.h>
#include <math.h> // potřebné pro použití sqrt()
int main(void)
{
    double y;
    double vysledek;
    printf("Zadejte racionalni cislo : \n");
    scanf("%lf", &y);
    vysledek = sqrt(y);
    printf("Odmocnina z cisla je: %f\n", vysledek);
    return 0;
}
```

Neošetřený vstup!

`printf("Odmocnina z cisla je: %f\n", sqrt(y));`



# Funkce - příkaz return

---

- ❑ Příkaz **return** má obecný formát:  
**return** *expression*<sub>opt</sub> ;
- ukončuje provádění funkce
- může se vyskytnout kdekoli v těle funkce





# Funkce - příkaz return

**Příklad:** definice funkce pro výpočet druhé mocniny zadaného čísla.

```
#include <stdio.h>
int get_sqr(void); // prototyp
int main(void)
{
    int sqrm;
    sqrm = get_sqr( );
    printf("Square: %d", sqrm);
    return 0;
}
int get_sqr(void) // definice
{
    int num;
    printf("Zadejte cele cislo: \n");
    scanf("%d", &num);
    return num*num; // druhá mocnina čísla
}
```

funkce je definována až po jejím volání → musel být použit prototyp

**Funkce s vedlejším efektem!**



# Funkce - parametry funkcí

*Příklad:* Definice funkce, která sečte dvě čísla a zobrazí výsledek.

```
#include <stdio.h>
void tisk_sum(int x, int y);
int main(void)
{
    int num1, num2;
    printf("Zadejte dve cela cisla: ");
    scanf("%d%d", &num1, &num2);
    tisk_sum(num1, num2);
    return 0;
}
void tisk_sum(int x, int y)
{
    printf("Soucet cisel je: %d \n", x + y);
}
```

! argumenty

(formální) parametry



# Funkce - parametry funkcí

Pokud funkci definujeme dříve než je použita nemusí být předtím uveden prototyp funkce. Následující funkce ani nevrací žádnou hodnotu.

```
#include <stdio.h>
void tisk_soucet(int x, int y)
{
    printf("Soucet cisel je: %d \n", x + y);
}
int main(void)
{
    int num1, num2;
    printf("Zadejte dve cela cisla: ");
    scanf("%d%d", &num1, &num2);
    tisk_soucet(num1, num2);
    return 0;
}
```

funkce nevrací hodnotu





# Funkce - parametry funkcí

*Příklad:* funkce s parametry, funkce vrací hodnotu

```
#include <stdio.h>
int soucet(int x, int y)
{
    return x + y;
}
int main(void)
{
    int num1, num2;
    printf("Zadejte dve cela cisla: ");
    ! scanf("%d%d", &num1, &num2);
    printf("Soucet cisel je: %d \n", soucet(num1, num2));
    return 0;
}
```

funkce vrací hodnotu



# Příkazy

---

- ☐ Složený příkaz.
- ☐ Výraz, příkaz.
- ☐ Oblast platnosti identifikátorů.
- ☐ Podmíněné příkazy.
- ☐ Cykly.
- ☐ Příkazy skoku.



## Syntaxe příkazu:

*statement:*

*labeled-statement*

*compound-statement*

*expression-statement*

*selection-statement*

*iteration-statement*

*jump-statement*



# Složený příkaz (blok)

[C99]

□ Syntaxe složeného příkazu:

```
compound-statement:  
    { block-item-listopt }  
block-item-list:  
    block-item  
    block-item-list block-item  
block-item:  
    declaration  
    statement
```

konfrontujte s  
Herout: Učebnice jazyka C  
kap.: 3.2



# Složený příkaz - příklad

*Příklad:* Převod stop na metry a metry na stopy. Jde o ilustrativní příklad, ve kterém z prostorových důvodů nejsou ošetřeny všechny vstupy.

```
/* **** */
/*  Převod stop na metry a metry na stopy      */
/*          1 metr = 3,281 stop                  */
/* **** */
#include <stdio.h>
const float KOEF_PREVODU=3.28;
```





# Složený příkaz – příklad pokr.

```
void vypisPrevod(int volba)
```

```
{  
    if(volba == 1) // stopy na metry  
    {  
        float num;  
        printf("Zadejte pocet stop: ");  
        ! scanf("%f", &num); // !!!  
        printf("%f stop je %f metru\n", num, num/KOEF_PREVODU);  
    }  
    else if(volba == 2) // metry na stopy  
    {  
        printf("Zadejte pocet metru: ");  
        float num;  
        ! scanf("%f", &num); // !!!  
        printf("%f metru je %f stop\n", num, num*KOEF_PREVODU);  
    }  
    else  
    {  
        printf("Chybna volba.");  
    }  
}
```

složené příkazy

lokální definice



# Složený příkaz – příklad pokr.

```
int main(void)
{
    int volba = 0;
    printf("Zadejte volbu \n"
           "1: Stopy na metry\n"
           "2: Metry na stopy\n: ");

    if (scanf("%d", &volba) == 1)
        vypisPrevod(volba);
    else
        return EXIT_FAILURE;
    return EXIT_SUCCESS;
}
```



# Výraz, příkaz

## □ Syntaxe výrazového příkazu:

*expression-statement:*  
*expression<sub>opt</sub> ;*

*Příklad: Příkaz, prázdný příkaz*

```
a = b + c;  
int c;  
// přeskočí mezery  
while ((c = getchar()) == ' ' )  
{  
  
// přeskočí prázdné řádky  
while ((c = getchar()) == '\n')  
;  
}
```



# Oblast platnosti identifikátoru

---

- ❑ Na úrovni souboru
  - od místa deklarace
  - do konce překládaného modulu (zdrojového souboru)
- ❑ Deklarace parametru funkce (při definici)
  - od místa deklarace parametru
  - do ukončení bloku definice funkce
- ❑ V rámci bloku
  - od deklarace
  - do konce aktuálního bloku



# Oblast platnosti identifikátoru

*Příklad:* Ukázka zastínění identifikátorů proměnných.

```
char znak = 'A';
int main(void)
{
    int cislo = 1;
    printf("%c%d", znak, cislo);           // A1
    char znak = 'B';                       // zastínění
    printf("%c%d", znak, cislo);           // B1
    {
        char znak = 'C';                   // zastínění
        printf("%c%d", znak, cislo);       // C1
        float cislo = 2.0;                 // zastínění
        printf("%c%.2f", znak, cislo);     // C2.00
    }
    printf("%c%d", znak, cislo);           // B1
}
```



# Oblast platnosti identifikátoru

- ❑ Makra a symbolické konstanty
  - od místa definice (`#define`)
  - do místa potlačení definice (`#undef`)
  - nebo do konce modulu
  - Nelze zastínit!

*Příklad:* Ukončení platnosti makra – v tomto případě konstanty.

```
#define N 10
int pole[N] = {0};
#undef N
int lany[N*N] = {0}; // zde už hodnotu N
                     // nelze použít
```



# Podmíněné příkazy

[C99]

Syntaxe:

*selection-statement:*

***if** ( expression ) statement*

***if** ( expression ) statement **else** statement*

***switch** ( expression ) statement*



# Příkaz if

*Příklad:* Je zadané číslo záporné ?

```
#include <stdio.h>
int main(void)
{
    int num;
    printf("Zadejte cele cislo:  ");
    scanf("%d", &num);
    if(num < 0)
        printf("\nCislo %d je zaporne.\n", num);

    return 0;
}
```





# Příkaz if

Poznámka k formátování:

```
// pozor na formátování, takto ne!
```

```
if (num < 0)
```

```
    printf(...);
```



```
    printf(...); // !
```

```
// lépe takto
```

```
if (num < 0)
```

```
{
```

```
    printf(...);
```

```
}
```

```
printf(...);
```



# Příkaz if-else

*Příklad: Zadané číslo je záporné nebo přirozené?*

```
#include <stdio.h>
int main(void)
{
    int num;
    printf("Zadejte cele cislo:  ");
    !scanf("%d", &num);
    if(num < 0)
        printf("\nCislo %d je zaporne.\n", num);
    else
        printf("\nCislo %d je prirozene.\n", num);

    return 0;
}
```



# Příkaz if-else

*Příklad:* Podíl dvou celých čísel, zbytek po dělení.

```
#include <stdio.h>
int main(void)
{
    int a, b;
    printf("Zadejte dve cela cisla:");
    scanf("%d %d", &a, &b);
    if (b == 0)
        printf("\nNulou delit nelze!\n");
    else
    {
        int podil, zbytek;
        podil = a / b;
        zbytek = a % b;
        printf("Jejich podil je: %d, zbytek po deleni "
               "je: %d\n", podil, zbytek);
    }
    return 0;
}
```





# Příkaz if-else

---

## □ Vnořený příkaz if:

Když je příkaz **if** součástí jiného příkazu **if** nebo **else**, říká se, že je vnořen do vnějšího **if**.

```
if (count > max)    // vnější if
    if (error)      // vnořený if
        printf("Chyba, zkuste to znovu." );
```



# Příkaz if-else

```
// takto ne
if (p)
    if (q) printf("p i q jsou pravdivé\n");
else printf("Ke kterému příkazu patří toto else?\n");

// takto ano
if (p)
    if (q)
        ↑ printf("p i q jsou pravdivé\n");
    else
        printf("Nyní je příslušnost else jasně vidět.\n");
```

vnořený **if-else**



# Příkaz if-else

Vnořování může mít i složitější strukturu. Příkaz **if** lze použít i na místě příkazu za **else** (tato konstrukce je obvyklejší). Výsledná konstrukce bývá často nazývána **if-else-if**:

*Příklad:*

```
if ( <expression1> ) <statement1>  
  
else if ( <expression2> ) <statement2>  
  
else if ( <expression3> ) <statement3>  
    ...  
    ...  
else if ( <expressionN> ) <statementN>  
  
else <statementN+1>
```



# Příkaz if-else

---

*Příklad:* zobrazí alfanumerický znak.

```
#include <stdio.h>
int main(void)
{
    printf("Zadejte alfanumericky znak:\n");
    int znak = getchar();
    printf("\nZadali jste  %c", znak);
}
```



# Příkaz if-else

*Příklad: pokračování*

```
if ((znak >= 'a') && (znak <= 'z'))
{ // malá písmena
    printf("\nZnak je male pismeno\n");
}
else if ((znak >= 'A') && (znak <= 'Z'))
{ // velká písmena
    printf("\nZnak je velke pismeno\n");
}
else if ((znak >= '0') && (znak <= '9'))
{ // číslice
    printf("\nZnak je cislice\n");
}
else
{ // jiný než alfanumerický znak
    printf("\nNeni zadan alfanumericky znak!\n");
}
return 0;
}
```

Úkol: co se zobrazí, když zadáme písmeno á ??





# Příkaz if-else

---

*Příklad:* zobrazí alfanumerický znak.

```
#include <stdio.h>
int main(void)
{
    printf("Zadejte alfanumericky znak:\n");
    int znak = getchar();
    printf("\nZadali jste  ");
    putchar(znak);
}
```



# Příkaz if-else

*Příklad:* zobrazí alfanumerický znak pokračování - takto ne!!!!

```
if ((znak >= 'a') && (znak <= 'z'))
    printf("\nZnak je male pismeno\n");
if ((znak >= 'A') && (znak <= 'Z'))
    printf("\nZnak je velke pismeno\n");
if ((znak >= '0') && (znak <= '9'))
    printf("\nZnak je cislice\n");
if (!( (znak >= 'a' && znak <= 'z') ||
      (znak >= 'A' && znak <= 'Z') ||
      (znak >= '0' && znak <= '9')) )
    printf("\nNeni zadan alfanumericky znak!\n");
return 0;
}
```



- ❑ Přepínač slouží k větvení výpočtu podle hodnoty celočíselného výrazu.

*selection-statement:*

***switch*** ( *expression* ) *statement*

*labeled-statement:*

***case*** *constant-expression* : *statement*

***default*** : *statement*



# Přepínač

---

- ❑ Vyhodnotí výraz, hodnotu porovnává s **case** návěštími. Pokud nalezne odpovídající hodnotu, předá řízení na toto návěští.
- ❑ Pokud hodnotě výrazu neodpovídá žádné návěští, je řízení předáno návěští **default**.
- ❑ Pokud **default** chybí a nevykoná se žádná varianta, řízení se předá bezprostředně za příkaz přepínače.
- ❑ V jednom přepínači
  - nelze mít dvě návěští se stejnou hodnotou.
  - může být **default** nejvýše jednou.
- ❑ Návěští **case** lze ukončit příkazem **break**. Pokud tomu tak není, program bude pokračovat přes všechna následující návěští, dokud nenarazí na **break, return** nebo konec přepínače.



# Přepínač

---

- ❑ Příkaz **break** může být v rámci přepínače umístěn v libovolně zanořeném bloku.
- ❑ Příkaz **break** se vždy vztahuje k nejbližšímu nadřazenému přepínači nebo cyklu.
- ❑ Pro **default** platí stejná pravidla jako pro ostatní návěští. Konvence říká, že by se mělo zapisovat na konec přepínače.
- ❑ Návěští **case** může být obsaženo uvnitř jiných příkazů (v rámci přepínače), kromě případného vnořeného přepínače.



# Přepínač

*Příklad:* Doporučená syntaxe a použití

```
switch (vyraz)
{
    case 1:
        printf("volba 1");
        break;
    case 2: // volby lze spojovat takto
    case 3:
        printf("volba 2 nebo 3");
        break;
    //case 2: - nelze mít dvě stejná návěští
    default:
        printf("vsechny ostatni volby");
        break;
}
```



# Přepínač

*Příklad:* Jak nepoužívat **case** uvnitř jiného **case**.

```
switch (znak)
{
    case 'A':
        {
            int tmp = 12;
            ! case 'B': // !!!
                printf("Kolik je tmp? %d\n", tmp);
        }
}
```



# Porovnání switch - if-else-if

---

- ❑ (Rozhodovací) výraz
  - **if** – lze testovat hodnoty různých typů
  - **switch** – výhradně celočíselné výrazy
- ❑ Větvení
  - **if-else-if** – provede nejvýše jednu z variant
  - **switch** – může provést více variant po sobě, pokud nejsou ukončeny pomocí **break**
- ❑ Umístění implicitní varianty
  - **switch** – **default** se může vyskytovat kdekoliv
  - **if-else-if** – pouze na konci





## Syntaxe:

*iteration-statement:*

**while** ( *expression* ) *statement*

**do** *statement* **while** ( *expression* ) ;

**for** ( *expression*<sub>opt</sub> ; *expression*<sub>opt</sub> ; *expression*<sub>opt</sub> ) *statement*

**for** ( *declaration* *expression*<sub>opt</sub> ; *expression*<sub>opt</sub> ) *statement*

## □ Příkaz while

- dopředu neznáme počet iterací
- cyklus **nemusí** proběhnout **ani jednou**



# Příkaz while

*Příklad:* počítá počet zadaných číselných a nečíselných znaků.

```
// Příkaz cyklu while
int ciska = 0;
int neciska = 0;
int znak;

while ((znak = getchar()) != EOF)
{
    if (znak >= '0' && znak <= '9')
        ciska++;
    else
        neciska++;
}

printf("Zadali jste %d cisel a %d "
       "neciselnych znaku.\n", ciska, neciska);
```



## Syntaxe:

*iteration-statement:*

***do*** *statement* ***while*** ( *expression* ) ;

### □ Příkaz do-while

- opakuje příkaz (*statement*) dokud je výraz (*expression*) pravdivý
- příkazová část se provede **vždy alespoň jednou**



# Příkaz do-while

*Příklad:* čeká na zadání správné operace a dvou čísel, pak operaci s čísly provede.

*Poznámka:* V rámci zjednodušení program nemá všude ošetřen chybný vstup od uživatele.

```
#include <stdio.h>
int main(void)
{
    int a, b;
    int ch;
    printf("Chcete:\n"
           "Scitat, Odecitat, Nasobit nebo Delit?\n");
    /* přinutí uživatele zadat správnou odpověď */
    do {
        printf("\nZadejte první písmeno operace: \n");
        ch = getchar();
    } while(ch!='S' && ch!='O' && ch!='N' && ch!='D');
    printf("\n");
```



# Příkaz do-while

*Příklad:* pokračování

```
printf("Zadejte prvni cislo: ");
```



```
scanf("%d", &a); //!
```

```
printf("Zadejte druhe cislo: ");
```



```
scanf("%d", &b); //!
```

```
if(ch=='S')
```

```
    printf("%d\n", a+b);
```

```
else if(ch=='O')
```

```
    printf("%d\n", a-b);
```

```
else if(ch=='N')
```

```
    printf("%d\n", a*b);
```

```
else if(ch=='D' && b!=0)
```

```
    printf("%d\n", a/b);
```

```
return 0;
```

```
}
```

přepište pomocí  
přepínače



# Příkaz do-while

---

*Poznámka k syntaxi:* Protože se druhá část příkazu **do-while** velmi podobá cyklu **while** s prázdným tělem, je velmi žádoucí, aby bylo klíčové slovo **while** psáno vždy za uzavírací složenou závorkou na jednom řádku:

```
do {  
    .  
    .  
    // příkazy  
    .  
    .  
} while (podmínka); //zde je středník nezbytný!
```



# Příkaz for

[C99]

## Syntaxe:

*iteration-statement:*

**for** ( *expression<sub>opt</sub> ; expression<sub>opt</sub> ; expression<sub>opt</sub>* ) *statement*  
**for** ( *declaration expression<sub>opt</sub> ; expression<sub>opt</sub>* ) *statement*

## □ Příkaz for

**for** ( *inicializace ; test podmínky ; inkrementace* ) *příkaz*



# Příkaz for

*Příklad:* zobrazí čísla od 1 do 10. Na nový řádek vypíše řetězec "konec".

```
// Příkaz cyklu for
for(int num = 1; num < 11; ++num)
    printf("%d ", num);
printf("\nkonec");
```

Diagrammatic annotations for the code above:

- inicializace** (initialization) points to `int num = 1`
- test podmínky** (test condition) points to `num < 11`
- inkrementace** (incrementation) points to `++num`
- An orange box contains the text: **co se stane, když místo ++num použijeme num++ vyzkoušejte** (what happens when instead of ++num we use num++ try it out), with a line pointing to the incrementation part of the for loop.

## *Poznámka:*

Pro opakování několika příkazů lze použít jako tělo cyklu složený příkaz.





# Příkaz for

[C99]

*Příklad:* zobrazí čísla od 1 do 10. Na nový řádek vypíše řetězec "konec,,. Shodné zadání jako předchozí.

*iteration-statement:*

**for** ( *expression*<sub>opt</sub> ; *expression*<sub>opt</sub> ; *expression*<sub>opt</sub> ) *statement*  
**for** ( *declaration* ; *expression*<sub>opt</sub> ; *expression*<sub>opt</sub> ) *statement*

// Příkaz cyklu for

zamyslete se: má, či nemá tam být středník ??

for(int num = 1; num < 11; ++num)  
 printf("%d ", num);  
printf("\nkonec");



# Příkaz for

[C99]

## ❑ Chybějící výrazy

- jsou legální možnosti
- chybí-li prostřední, vyhodnotí se jako pravdivý výraz (používat výjimečně!)

*iteration-statement:*

**for** ( *expression<sub>opt</sub>* ; *expression<sub>opt</sub>* ; *expression<sub>opt</sub>* ) *statement*

**for** ( *declaration* *expression<sub>opt</sub>* ; *expression<sub>opt</sub>* ) *statement*

*Příklad:* Chybějící výrazy – nekonečný cyklus.

```
// Příkaz cyklu for
int num = 0;
for( ; ; )
    printf("%d ", num++);
printf("\nkonec");
```

vyzkoušejte, sledujte hodnoty, jaké se budou zobrazovat, když dojde k překročení rozsahu kladných hodnot proměnné *num*



# Příkaz for

*Příklad:* Součin a součet čísel od 1 do 6. Tělo cyklu je složený příkaz.

```
#include <stdio.h>
#define POCET 6
int main(void)
{
    int sum = 0;
    int soucin = 1;
    for(int num = 1; num <= POCET; ++num)
    {
        sum += num;
        soucin *= num;
    }
    printf("Soucin a soucet cisel od 1 do %d\n", POCET);
    printf("Soucin: %d soucet: %d \n", soucin, sum);
    return 0;
}
```



# Příkaz for

---

Cyklus **for** může probíhat i naopak. Například tento úsek snižuje hodnotu řídící proměnné cyklu:

```
for (int num=20; num>0; --num)
```

Hodnota řídící proměnné cyklu může být zvyšována nebo snižována i o jinou hodnotu než 1.

*Příklad:* zobrazí čísla od nuly do sta po pěti.

```
for (int i=0; i<101; i+=5)
    printf ("%d ", i);
```



# Příkaz for

Neúplný cyklus **for** lze většinou přehledněji přepsat pomocí cyklu **while** nebo **do-while**.

*Příklad:* zobrazí čísla od nuly do devíti.

```
for(int i=0; i<10; )  
{  
    printf("%d ", i);  
    i++;  
}
```

```
int i = 0;  
while(i < 10)  
{  
    printf("%d ", i);  
    i++;  
}
```



# Příkaz for

Také lze používat více řídících proměnných cyklu:

*Příklad:* vypíše do sloupců dvě posloupnosti, jednu rostoucí a jednu klesající.

```
for(int i = 0, j = 10; i <= j; ++i, --j)
{
    printf("%d %d\n", i, j);
}
```

zamyslete se: jak to, že zde mohou být použity dva výrazy??



# Příkaz for

## Vnořování cyklů:

Když tělo cyklu obsahuje jiný cyklus, říká se, že je druhý cyklus vnořen do prvního. Jakýkoliv cyklus může být vnořen do jiného.

*Příklad:* zobrazí desetkrát čísla od jedné do deseti

```
// vnořený cyklus
for(int i=0; i<10; ++i)
{
    for(int j=1; j<11; ++j)
        printf("%d ", j); // vnořený cyklus
    printf("\n");
}
```



Syntaxe:

```
jump-statement:  
    goto identifier ;  
    continue ;  
    break ;  
    return expressionopt ;
```

- ❑ Ukončení cyklu příkazem **break**
  - umožňuje ukončit cyklus v libovolném místě těla cyklu
  - zpracování programu pokračuje příkazem následujícím za cyklem





# Příkaz **break**

*Příklad:* zobrazí jen čísla od jedné do deseti

```
// break v cyklu
for(int i=1; i<100; ++i)
{
    printf("%d ", i);
    if(i==10)
        break; /* ukončení cyklu */
}
// bude se pokračovat tady
```



# Příkaz **break**

Při vnořování cyklů ukončí **break** jen nejbližší cyklus, ve kterém je použit.

*Příklad:* zobrazí čísla 0 až 5 pětkrát.

```
// break v cyklu
for(int i=0; i<5; ++i)
{
    for(int j=0; j<100; ++j)
    {
        printf("%d", j);
        if(j==5) break;
    }
    printf("\n"); // break skočí sem
}
```





# Příkaz **continue**

Příkaz **continue** si vynutí nové vyhodnocení podmínky cyklu, přičemž se přeskočí všechny příkazy mezi ním a koncem těla cyklu.

*Příklad:* program, který nikdy nic nevypíše.

```
// continue v cyklu
for(int x=0; x<100; ++x)
{
    continue;
    printf("%d ", x); //nikdy se neprovede
}
```



# Příkaz **continue**

- ❑ V cyklech **while**, **do-while**
  - skok na test podmínky → (ne)pokračování cyklu.
- ❑ V cyklu **for**
  - provede inkrementační část cyklu
  - pak test podmínky cyklu → cyklus (ne)pokračuje.
- ❑ Jeden z vhodných případů použití **continue** je nové spuštění posloupnosti příkazů, když nastane chyba.

*Příklad:* Program počítá průběžný součet všech čísel zadaných uživatelem. Před přičtením hodnoty k průběžnému součtu otestuje správnost zadaného čísla tak, že ji uživatel musí zadat znovu. Pokud se tato dvě čísla liší, program použije **continue** na nový start cyklu.



# Příkaz **continue**

```
#include <stdio.h>
int main(void)
{
    int i, j;
    int total=0;
    do {
        printf("Zadejte dalsi cislo (0 = konec): ");
        scanf("%d", &i);
        printf("Zadejte cislo znovu: ");
        scanf("%d", &j);
        if(i != j)
        {
            printf("Cisla nesouhlasí\n");
            continue;
        }
        total += i;
    } while(i != 0);
    printf("Soucet je: %d", total);
    return 0;
}
```

!

!

!



# Příkaz **return**

- ❑ příkaz **return** ukončí provádění funkce, která tento příkaz obsahuje
- ❑ ve funkci **main** ukončí příkaz **return** celý program

```
#define N 10
int x[N], a[N], b[N]; // definice globálních polí
int vypocet(int a[], int b[], int x[])
{
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            for (int k = 0; k < N; k++) {
                if (x[k] == 0) return 0; /* neúspěch */
                a[i] += b[j] / x[k];
            }
        }
    }
    return 1; /* úspěch */
}
```

zamyslete se: jak se vyhodnotí tento výraz??



# Příkaz **skoku**

[C99]

- ❑ Syntaxe příkazu skoku:

```
labeled-statement:  
    identifier : statement  
  
goto identifier ;
```

*Příklad:* následující **goto** přeskočí příkaz printf().

```
goto navesti;  
printf("Toto se nikdy nevytiskne.");  
navesti: printf("Toto se vytiskne.");
```

- ❑ Jedním z mála akceptovatelných použití **goto** je vyskočení z hluboko zanořené části programu, pokud nastane fatální chyba.
- ❑ Nepoužívat! (téměř) nikdy!

# Řídicí struktury v programování

---







# Kontrolní otázky

---

1. Jaký je obecný formát programu?
2. Co je deklarace funkce, kde se uvádí?
3. Co je definice funkce?
4. Jak se definuje funkce, která nevrací hodnotu?
5. K čemu slouží parametry funkce, kde se deklarují?
6. Kde a kdy se používají argumenty funkce?
7. Uveďte rozdíl mezi globální a lokální proměnnou.
8. Co se rozumí pojmem zastínění proměnné?
9. Jaká je struktura složeného příkazu?
10. Jaké jsou druhy podmíněných příkazů, uveďte jejich rozdíl.
11. Uveďte druhy cyklů a v čem se jednotlivé cykly liší?
12. Uveďte jednotlivé příkazy skoku a kde se používají.



# Úkoly k procvičení

---

Zpracování posloupností hodnot.

Vytvořte program v jazyce C pro:

1. součet  $n$  hodnot.
2. počet záporných, nulových a kladných hodnot z  $n$  hodnot.
3. aritmetický průměr kladných hodnot z neznámého počtu hodnot. Seznam hodnot je ukončený nulou.
4. zjištění největší hodnoty z  $n$  hodnot.
5. zjištění nejmenší a největší hodnoty z  $n$  hodnot.
6. zjištění 2 největších hodnot z  $n$  hodnot.
7. zjištění největší hodnoty a počet jejích výskytů z  $n$  hodnot.
8. zjištění nejmenší hodnoty a pořadí jejího prvního výskytu z  $n$  hodnot.
9. Jsou dány známky 1-5, koncová hodnota 0. Určete počet výskytů jedniček.
10. Jsou dány kladné hodnoty vyjadřující teplotu, zakončené nulou. Určete nejmenší teplotu.
11. Ze vstupu zadejte hodnotu *limit*. Proveďte součet čísel aritmetické posloupnosti  $1+2+\dots+n$ , tak, aby součet nepřesáhl hodnotu *limit*. Vypište maximální hodnotu  $n$ , pro kterou součet čísel  $1-n$  nepřesáhne hodnotu *limit*.
12. Zobrazte tabulku funkcí  $\sin(x)$ ,  $\cos(x)$ ,  $\tan(x)$ , kde  $x$  se mění v intervalu  $\langle 0, 90 \rangle$  stupňů.
13. Ze vstupu zadejte koeficienty  $a, b, c$ . Zobrazte tabulku hodnot polynomu  $ax^2+bx+c$ , kde  $x$  se mění v intervalu  $\langle 1, 10 \rangle$  po jedné. Pro výpočet polynomu definujte funkci.