Řídící struktury

Blok

- = posloupnost příkazů v `{}`
- syntaxe:

```
{
        prikaz 1;
        prikaz 2;

        prikaz N;
}
```

žádný `;`

- použití
  - sdružení několika příkazů → přehlednost
  - situace si žádá více příkazů (řídící struktury), ale C vyžaduje příkaz jediný

if

- neúplná podmínka

0 (0.0) → false ...

```
if (V)
    Příkaz;
```

- úplná podmínka

```
if (V)
    Příkaz 1;
else
    Příkaz 2;
```

- více příkazů

```
if (V)
{
    Příkaz 1;
    Příkaz 2;
}
else
    Příkaz 3;
```

- výraz `V`
  - typicky logický

```
if (x > (a + b))
if (!(x <= -5) || (x > 10))
```

  - méně tradiční (pokud možno nedělat)

```
if (a)
if (!a)
if (a >= (b = c))
```

```
double uhel;
if (uhel)          // if (uhel != 0.0)
```

```
if (1)
```
pro účely ladění

- extrémní

vždy „true“: podmínka → pointer (nenulová adresa)

```
if ("text")
```

- chybný (syntakticky správný)

```
if (a = 5)         // zřejmě správně if (a == 5)
```
vždy „true“

- vnořené `if`

```
if (V)
   if (V)
      Příkaz 1;
   else
      Příkaz 2;
```
`else` vždy k nejbližšímu `if`

- lze ovlivnit závorkami

```
if (V)
{
   if (V)
      Příkaz 1;
}
else
   Příkaz 2;
```

switch

```
switch (V)
{
      case navesti_1:
            Prikazy;
            break;
          |
      case navesti_X:
            Prikazy;
            break;
      default:
            Prikazy;
            break;
}
```
ordinálního typu (celá čísla, `enum`)

pouze konstanty

počet neomezen

ukončuje větev

nemusí být

poslední nemusí být

- na pořadí sekcí nezáleží

- ANSI C: max. 257 `case` sekcí
- Příklad:

```
char znak;
switch (znak)
{
        case 'A':
                velke++;
                printf("mam A");
                break;
        case 5:
                nechtene++;
                break;
}
```

= 65

- propadání

```
switch (V)
{
        case K1:
        case K2:
        case K3:
                Prikaz 1;
                break;
        case K4:
                Prikaz 2;
                Prikaz 3;
        case K3:
                Prikaz 4;
                break;
}
```

`case` bez příkazů

C OK; C# OK

C OK; C# Error

`case` s příkazy; chybí `break`

Opakování terminologie cyklů

- *řídící proměnná cyklu* = (ŘP) proměnná, na které závisí ukončení/pokračování cyklu
  - nejlépe pouze jedna

  jména řídících proměnných → `i`, `j`, `k` …
- *podmínka řídící pokračování cyklu* = logický výraz obsahující řídící proměnnou cyklu
- *hlavička cyklu* = klíčové slovo `for` nebo `while` a výraz v následujících `()`
  - = nutná administrativa cyklu
- *tělo cyklu* = příkazy, které se budou opakovat (výkonný kód cyklu)
  - jeden příkaz nebo blok
- cykly:
  - `for` – před spuštěním cyklu je počet opakování předem znám
  - `while` – počet opakování není znám, nemusí proběhnout ani jednou
  - `do while` – počet opakování není znám, min. jeden běh
  - `break` / `continue` – řídí další pokračování cyklů

while

0 (0.0) → false ...

```
while (V)
    Příkaz;
```

- Příklady

IŘP

spíše pro `for`

```
int i = 0;
while (i < 4)
{
      printf("%d\n", i);
      i++;
}
```

ZŘP; Nezapomínat!!!

- nechtěná nekonečná smyčka

```
int i = 0;
while (i < 4);
{
      printf("%d\n", i);
      i++;
}
```

Prázdný příkaz (tělo)

- nekonečný cyklus – programy v embedded aplikacích

```
// inicializace zařízení
while (1)
{
      // načtení vstupů
      // zpracování vstupů
      // nastavení výstupů
}
```

do-while

```
do
    Příkaz;
while (V)
```

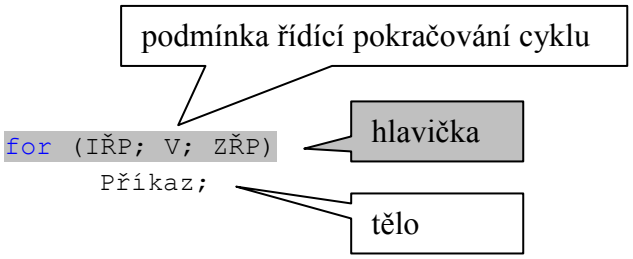
- příklad

```
int i = 0;
do
{
      printf("%d\n", i);
      i++;
}
while (i < 4);
```

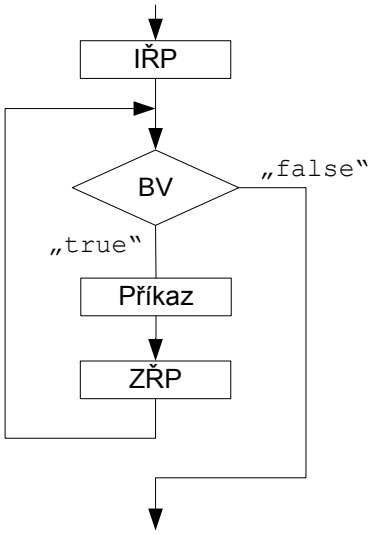
musí být!

for

- zkrácený zápis ostatních cyklů



- sémantika



- Příklad – tradiční použití

pozor!

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

- méně tradiční použití – více ŘP

operátor ,

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

- méně tradiční použití – chybějící IŘP

```
int i;
// nacteni hodnoty "i" z klavesnice
for(; i < 10; i++)
{
      printf("i = %d\n", i);
}
```

- nekonečný cyklus – programy v embedded aplikacích

```
// inicializace zařízení
for (;;)
{
      // načtení vstupů
      // zpracování vstupů
      // nastavení výstupů
}
```

break

- použit v těle cyklu – ukončí cyklus
- pro `for`, `while`, `do while` (`switch`)

```
int i;
double x;
for(i = 0; i < 10; i++)
{
      // cteni "x" z klavesnice
      if (x < 0.0)
            break;
      printf("x = %lf\n", x);
}
```

x < 0 → cyklus končí

continue

- použit v těle cyklu – zbytek příkazů v těle je přeskočen (cyklus nekončí)
- pro `for`, `while`, `do while`

```
int i;
double x;
for(i = 0; i < 10; i++)
{
      // cteni "x" z klavesnice
      if (x < 0.0)
            continue;
      printf("x = %lf\n", x);
}
```

netiskne záporná x

goto

- nepodmíněný skok na návěští

- použití – jen když nelze jinak → vždy lze nahradit kombinací řídící struktura + `break` / `continue`, `return`
- příklad:

```
int i, j;
for (i = 0; i < 10; i++)
{
      printf("vnejsi smycka - i = %d\n", i);
      for ( j = 0; j < 3; j++ )
      {
            printf("vnitrni smycka - j = %d\n", j);
            if ( i == 5 )
                  goto stop;
      }
}
printf("cykly konec\n");      // to se nikdy nevytiskne
stop:
      printf("skok na \"stop\" - i = %d\n", i);
```

- omezení → neskákat:
  - do `if` nebo `else` bloků zvnějšku
  - dovnitř těla cyklů zvnějšku
  - dovnitř bloku `switch`