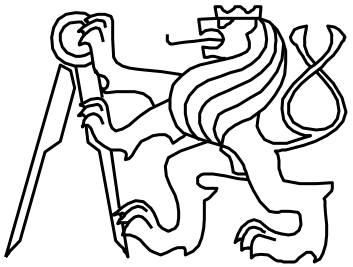
Příprava studijního programu Informatika je podporována projektem financovaným z Evropského sociálního fondu a rozpočtu hlavního města Prahy.
Praha & EU: Investujeme do vaší budoucnosti

Řídící struktury, if, while, switch



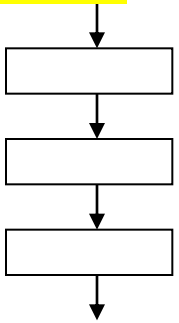
BI-PA1 Programování a algoritmizace 1, ZS 2012-2013
Katedra teoretické informatiky
© Miroslav Balík
Fakulta informačních technologií
České vysoké učení technické

Řídicí struktury

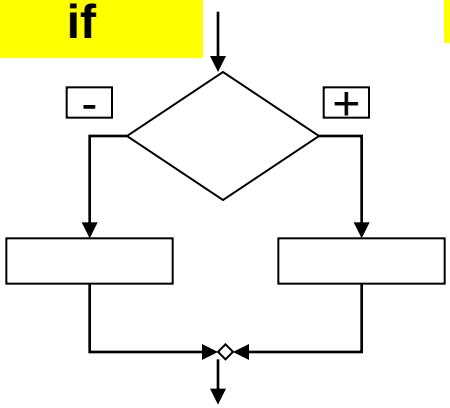
- Řídicí struktura je programová konstrukce, která se skládá z dílčích příkazů a předepisuje pro ně způsob provedení
- Tři druhy řídicích struktur:
  - ***posloupnost***, předepisuje postupné provedení dílčích příkazů
  - ***větvení***, předepisuje provedení dílčích příkazů v závislosti na splnění určité podmínky
  - ***cyklus***, předepisující opakované provedení dílčích příkazů v závislosti na splnění určité podmínky
- Řídicí struktury mají obvykle formu strukturovaných příkazů
- Postupně zavedeme následující složené příkazy:
  - složený příkaz a blok pro posloupnost
  - příkaz *if* pro větvení
  - příkazy *while, do* a *for* pro cyklus
  - příkaz *switch* (přepínač) pro větvení

Řídící struktury

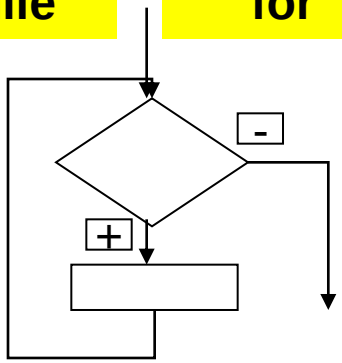
sekvence



if

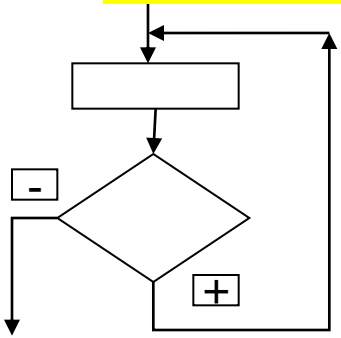


while

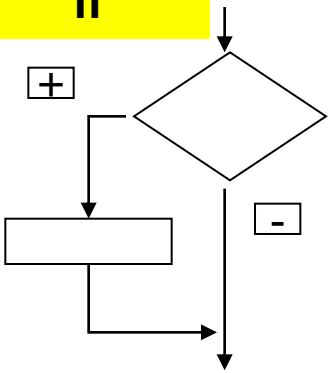


for

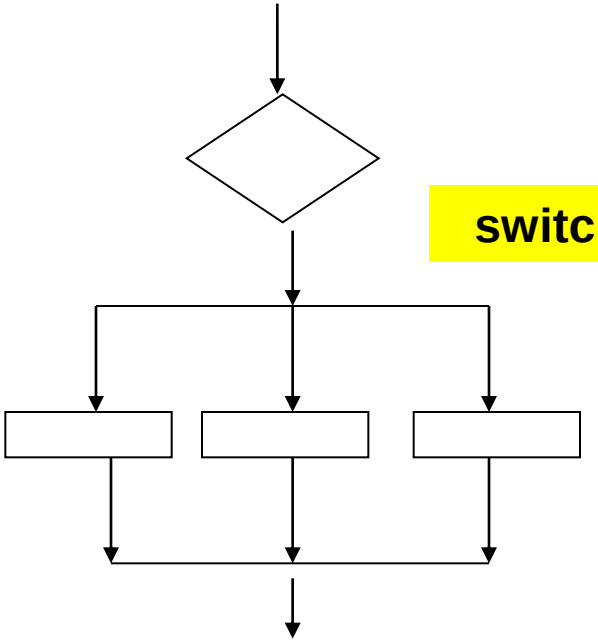
do



if



switch



Složený příkaz a blok

- Složený příkaz: **{**

  posloupnost příkazů

  }
  - Složený příkaz činí z posloupnosti příkazů (z hlediska syntaktických pravidel) jediný příkaz
  - Použití uvidíme u příkazu *if* a příkazů cyklu
- Blok: **{**

  posloupnost deklarací

  posloupnost příkazů

  }
  - Blok je základem pro definici funkce (viz funkce *main* v předchozích příkladech)
  - Deklarace jsou v bloku lokální, tzn. neplatí vně bloku
  - V bloku funkce platí globální deklarace (deklarace na úrovni souboru) uvedené v souboru před definicí funkce
  - Lokální deklarace zastíní globální (nelokální) deklarace
  - Bloky lze do sebe vnořovat

Globální ~ lokální deklarace

- Pro ilustraci zastínění globální deklarace lokální předběhneme - použijeme program se dvěma funkcemi

```
/* prog4-1.c */
#include <stdio.h>
#include <stdlib.h>
int x = 1, y = 2; /* globalni promenne */

void fun(void)  {
    int a = 1;
    printf("funkce fun:\n");
    printf("x=%d y=%d a=%d\n", x, y, a);
}

int main(void)  {
    int x = 10; /* lokalni x zastinuje globalni x */
    int a = 20;
    fun();
    printf("funkce main:\n");
    printf("x=%d y=%d a=%d\n", x, y, a);
    return 0;
}
```

Příkaz if

- Příkaz *if* (podmíněný příkaz) umožňuje větvení na základě podmínky
- Má dva tvary:

  if (podmínka) příkaz1 else příkaz2

  if (podmínka) příkaz1

  ,kde *podmínka* je výraz, jehož nenulová hodnota znamená *splněno* (*true*) a nulová hodnota *nesplněno* (*false*)
- Příklad (do *min* uložit a pak vypsat menší z hodnot *x* a *y*):

  1. varianta (prog4-2a.c)

```
if (x < y) min = x;
 else min = y;
printf("mensi cislo je %d\n", min);
```

  2. varianta (prog4-2b.c)

```
min = x;
if (y < min) min = y;
printf("mensi cislo je %d\n", min);
```

Příkaz if

- Jestliže v případě splnění či nesplnění podmínky má být provedeno více příkazů, je třeba z nich vytvořit složený příkaz nebo blok.

- Příklad: jestliže $x < y$, vyměňte hodnoty těchto proměnných

```
if (x < y)  {
   int pom = x;
   x = y;
   y = pom;
}
```

- Špatné řešení:

```
if (x < y)
  int pom = x;
  x = y;
  y = pom;
```

Příkaz if - příkad

Do *min* uložte menší z čísel *x* a *y* a do *max* uložte větší z čísel

(prog4-2c.c)

```
if (x < y) {
    min = x;
    max = y;
} else {
    min = y;
    max = x;
}
```

Špatné řešení:

```
if (x < y)
    min = x;
    max = y;
else min = y; //unexpected token else
    max = x;
```

Příkaz if

- Do příkazu *if* lze vnořit libovolný příkaz, tedy i podmíněný příkaz
- Příklad: do *s* uložte –1, 0 nebo 1 podle toho, zda *x* je menší než nula, rovno nule nebo větší než nula

```
if (x < 0) s = -1;
else if (x == 0) s = 0;
else s = 1;
```

Poznámka: tento příklad jsme řešili minulý týden pomocí podmíněného výrazu

- Příklad: do *max* uložte největší z čísel *x*, *y* a *z*

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

Hledání a výpis druhého největšího prvku - naivně

- Ze dvou čísel x, y:

```
if (x>y) printf("Max 2 je %d\n" ,y);
else printf("Max 2 je %d\n" ,x);
```

- Ze třech čísel x, y,z:

```
if (x>y && y>z) printf("Max 2 je %d\n" ,y);
else if (z>y && y>x) printf("Max 2 je %d\n" ,y);
else if (x>z && z>y) printf("Max 2 je %d\n" ,z);
else if (y>z && z>x) printf("Max 2 je %d\n" ,z);
else if (y>x && x>z) printf("Max 2 je %d\n" ,x);
else if (z>x && x>z) printf("Max 2 je %d\n" ,x);
```

počet různých pořadí je 3*2*1 = 6

- Ze čtyř čísel x,y,z,zz:

```
if (x>y && y>z && z>zz) printf("Max 2 je %d\n" ,y);
else if (z>y && y>x ....
```

počet různých pořadí je 4*3*2*1 = 24

- Z pěti čísel …

počet různých pořadí je 5*4*3*2*1 = 120

- a co když nebudou všechna čísla různá?

Hledání a výpis druhého největšího prvku - lépe

1. Zavedeme dvě proměnné max a max2
2. Nastavíme počáteční hodnoty z prvních dvou hodnot posloupnosti (**`x`** a **`y`**) Stačilo by obě nastavit na 0? Nebo obě nastavit na první hodnotu?

   -5, -45, -12, -82 10, 5, 2, -45
3. pro každé další číslo ze vstupní posloupnosti rozhodneme, zda je nutné jejich hodnoty změnit (čteme je pomocí cyklu, viz. dále)

totéž jinak: **`max=x>y?x:y;`**

ad 1: `int max, max2;` **`max2=x>y?y:x;`**

ad 2:
```
if(x>y){max=x; max2=y;}
  else{max=y; max2=x;}
```

Aktualizuj max i max2
{max2 = max;max = z;}

Aktualizuj max2
{max2 = z;}

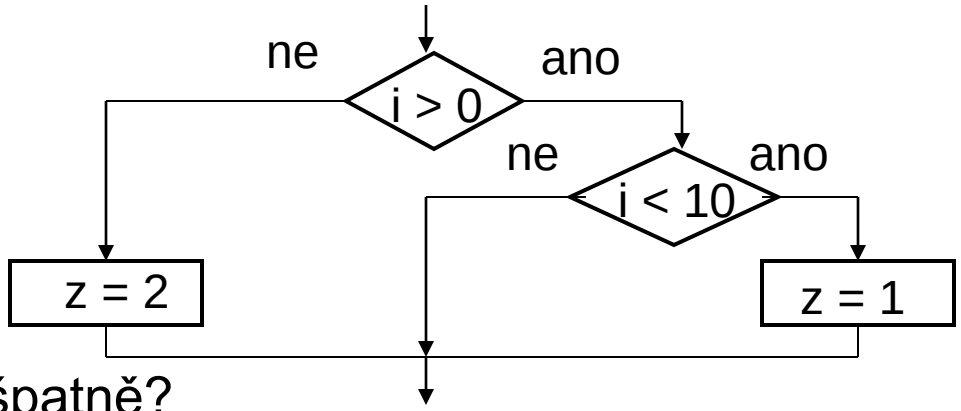
Příchozí číslo (**z**) je

- **větší než** max
- **menší (nebo roven) než max, ale větší než** max2
- **menší než** max2

Nedělej nic

Příkaz if

- Pozor na vnoření neúplného *if* do úplného *if*
- Příklad: zapište příkazem *if* následující větvení:



- Dobře nebo špatně?

```
if (i > 0) {
        if (i < 10) z = 1;
} else z = 2;
```

- Dobře nebo špatně?

```
if (i > 0)
        if (i < 10) z = 1;
        else z = 2;
```

Příklad- pro zadaný rok zjisti, zda je přestupný

- Přestupný rok je dělitelný 4 a buď není dělitelný 100 nebo je dělitelný 400

```
/* prog4-2d.c */
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    int rok;
    printf("zadej rok: ");
    scanf("%d", &rok);
    printf("rok %d ", rok);
    if (rok % 4 == 0 && (rok % 100 != 0 || rok % 400 == 0))
        printf("je prestupny\n");
    else printf("neni prestupny\n");
    return 0;
}
```

Přestupný x nepřestupný

1986
1900
2000
1500
0

Pozn.: Přestupné roky zavedl Řehoř XIII (Gregoriánský kalendář) v roce 1582, v Čechách leden 1584, na Moravě v říjnu. Češi měli v roce 1584 velikonoce o 4 týdny dříve než na Moravě

Pořadí podmínek a rychlost programu

- celé číslo od nuly do 99999 určí, jeho desítkový zápis

```
in
// I
if(x<10) p = 1;
else if(x<100) p = 2;
else if(x<1000)p = 3;
else if(x<10000)p = 4
else p = 5;
```

Lze to „rychleji“:
(int)log10(x+0.5)+1;
// + 0.5 kvůli 0, zkuste to odstranit a co dostanete?

```
ení 2.
if(x>9999) p = 5;
else if(x>999) p = 4;
else if(x>99) p = 3;
else if(x>9) p = 2;
else p = 1;
```

7 ... jedno místo
58 ... dvě
1101 ... 4 místa

osk

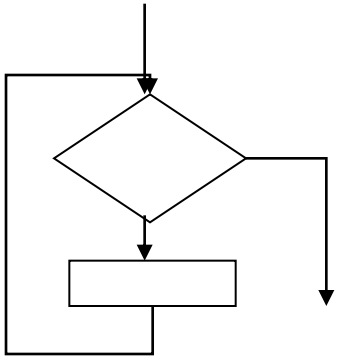
buji

číslo 0 – 9, jedno por nání
číslo 10 - 99, dvě
číslo 100 – 999, tři
číslo 1000 – 99999, čtyři
10*1+90*2+900*3+99000*4 =
388890 porovnání

číslo 0 – 99, čtyři
číslo 100 - 999, tři
číslo 1000 – 9999, dvě
číslo 10000 – 99999, jedno
100*4+900*3+9000*2+90000*1 =
111100 porovnání

Druhé řešení je téměř 4 krát rychlejší !!!

Příkaz while



- Základní příkaz cyklu, který má tvar

  while (*podmínka*) *příkaz*

- Příklad:

```
q = x;
while (q>=y) q = q-y;
```

(jsou-li hodnotami proměnných *x* a *y* přirozená čísla, co je hodnotou proměnné *q* po skončení uvedeného cyklu?)

Vyzkoušejte pro x = 10, y = 3.

Příkaz while

- Má-li se opakovaně provádět více příkazů, musí tvořit složený příkaz
- Příklad:

```
q = x;
p = 0;
while (q>=y) {
  q = q-y;
  p = p+1;
}
```

Jsou-li hodnotami proměnných *x* a *y* přirozená čísla, co je hodnotou proměnných *p* a *q* po skončení uvedeného cyklu?

Vyzkoušejte pro x = 10, y = 3.

Příklad - faktoriál

- Výpočet faktoriálu přirozeného čísla *n* ($n! = 1 \times 2 \times \ldots \times n$)

```
/* prog4-3a.c */
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    int i = 1, f = 1, n;
    printf("zadejte prirozene cislo: ");
    scanf("%d", &n);
    while (i<n) {
      i = i+1;
      f = f*i;
    }
    printf("%d! = %d\n", n, f);
    return 0;
}
```

Příkaz do - while

- Příkaz cyklu do se od příkazu while liší v tom, že podmínka se testuje až za tělem cyklu
- Tvar příkazu: do *příkaz* while (*podmínka*);
- Vnořeným příkazem je nejčastěji složený příkaz
- Příklad (faktoriál, prog3-3b.c):

```
f = 1;
i = 0;
do {
        i = i+1;
        f = f*i;
} while (i<n);
```

- Poznámka k syntaxi: příkaz *do* je jediným strukturovaným příkazem, který končí středníkem
- Poznámka k sémantice: příkaz *do* provede tělo cyklu alespoň jednou, nelze jej tedy použít v případě, kdy lze očekávat ani jedno provedení těla cyklu (pokud nepoužijeme break)

Příkaz for

- Cyklus je často řízen proměnnou, pro kterou je stanoveno:
  - jaká je počáteční hodnota
  - jaká je koncová hodnota (podmínka pro provedení těla cyklu)
  - jak změnit hodnotu proměnné po každém provedení těla cyklu
- Příklad (prog3-3c.c):

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

počáteční hodnota řídicí proměnné

podmínka pro provedení těla cyklu

tělo cyklu

změna řídicí proměnné

- Cykly tohoto druhu lze zkráceně předepsat příkazem *for*:

```
f = 1;
for (i=1; i<=n; i=i+1) f=f*i;
```

Příkaz for

- Tvar příkazu for:

  for (inicializace ; podmínka ; změna) příkaz

- Provedení příkazu for:

```
inicializace;
while (podmínka) {
  příkaz
  změna;
}
```

- Změnu řídicí proměnné přičtením resp. odečtením 1 lze zkráceně předepsat pomocí operátoru inkrementace resp. dekrementace:

  x++ …. x se zvětší o 1

  x-- …. x se zmenší o 1

- Příklad (prog3-3c.c) :

```
f = 1;
for (i=1; i<=n; i++) f=f*i;
```

Zpracování posloupností

- Příklad: program pro součet posloupnosti čísel
- Hrubé řešení:

```
suma = 0;
while (nejsou přečtena všechna čísla) {
  přečti další celé číslo;
  suma = suma+dalsi;
}
```

- Jak určit, zda jsou přečtena všechna čísla?
- Možnosti:
  1. počet čísel bude vždy stejný, např. 5
  2. počet čísel bude dán na začátku vstupních dat
  3. vstupní data budou končit „zarážkou“, např. nulou
- Struktura vstupních dat formálně:
  1. $a_1\ a_2\ a_3\ a_4\ a_5$
  2. $n\ a_1\ a_2\ \ldots\ a_n$
  3. $a_1\ a_2\ \ldots\ a_5\ 0$ kde $a_i \neq 0$

Zpracování posloupností I

- vstupní data: a_1 a_2 a_3 a_4 a_5

```
/* prog4-4a.c */ /* soucet 5-ti cisel zadanych z klavesnice */
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int dalsi, suma = 0, i;
    printf("zadejte 5 cisel\n");

    for (i=1; i<=5; i++) {
        scanf("%d", &dalsi);
        suma = suma + dalsi;
    }
    printf("soucet je %d\n", suma);
    return 0;
}
```

Zpracování posloupností - II

vstupní data: $n \; a_1 \; a_2 \; \ldots \; a_n$

```
/* prog4-4b.c */
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int dalsi, suma = 0, i, n;
    printf("zadejte pocet cisel: ");
    scanf("%d", &n);
    printf("zadejte %d cisel\n", n);
    for (i=1; i<=n; i++) {
      scanf("%d", &dalsi);
      suma = suma + dalsi;
    }
    printf("soucet je %d\n", suma);
    return 0;
}
```

Zpracování posloupností - III

vstupní data: a_1 a_2 ... a_n 0 ,kde $a_i \neq 0$

```
/* prog4-4c.c */
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int dalsi, suma = 0;
    printf("zadejte posloupnost cisel zakoncenou nulou\n");
    scanf("%d", &dalsi);

    while (dalsi!=0) {
      suma = suma + dalsi;
      scanf("%d", &dalsi);
    }
    printf("soucet je %d\n", suma);
    return 0;
}
```

Příkazy break a continue

- Příkazy *while* a *for* testují ukončení cyklu před provedením těla cyklu
- Příkaz *do* testuje ukončení cyklu po provedení těla cyklu
- Někdy je třeba ukončit cyklus v nějakém místě uvnitř těla cyklu (které je v tom případě tvořeno složeným příkazem)
- K tomu slouží příkaz *break* vnořený do podmíněného příkazu

```
while (...) {
  …
   if (ukončit) break;
  …
 }
```

- Příkaz *continue* předepisuje předčasné ukončení těla cyklu
- Následující příkaz vypíše čísla od 1 do 100 s výjimkou dělitelných 10:

```
for (i=1; i<=100; i++) {
        if (i%10 == 0) continue;
        printf("%d\n", i);
}
```

Konečnost cyklů

- Aby byl algoritmus konečný, musí každý cyklus v něm uvedený skončit po konečném počtu kroků
- Nekonečný cyklus je častou chybou
- Základní pravidlo pro konečnost cyklu:
  - provedením těla cyklu se musí změnit hodnota proměnné vyskytující se v podmínce cyklu
- Triviální příklad špatného cyklu:
  - `while (i!=0) j = i-1;`

  Tento cyklus se buď neprovede ani jednou, nebo neskončí
- Uvedené pravidlo konečnost cyklu ještě nezaručuje
- Konečnost cyklu závisí na hodnotách proměnných před vstupem do cyklu
  - `double x=0; int i = -1;`
  - `while (i<0) {x = x + sin(i* 0.6); i--;}`

Konečnost cyklů

```
while (i!=n) {
  P; // příkaz, který nezmění hodnotu proměnné i
  i++;
}
```

- Otázka: co musí splňovat hodnoty proměnných *i* a *n* před vstupem do cyklu, aby cyklus skončil?
- Odpověď – vstupní podmínka konečnosti cyklu, i a n celá čísla:

  $i <= n$
- A jak je to pro typ int? A jak pro double?
- Vstupní podmínku konečnosti cyklu lze určit téměř ke každému cyklu (někdy je to velmi obtížné)
- Splnění vstupní podmínky konečnosti cyklu musí zajistit příkazy předcházející příkazu cyklu
- Zásada: zabezpečený program testuje přípustnost vstupních dat

Konečnost cyklů – problém 3x+1

Problém se nazývá též syrakuský, Collatzův,...

Vstup: přirozené číslo n

```
while(n!=1){
  if(n%2==0) n = n/2;
  else n = 3*n + 1;
}
```

problém: skončí tento algoritmus a po kolika krocích?

pro n = 6 ... 6, 3, 10, 5, 16, 8, 4, 2, 1

pro n = 27 ... 27, 82, 41, 124, 62, 31, 94, 47, 142, 71, 214, 107, 322, 161, 484, 242,..., 9232, ..., 4, 2, 1 (111 iterací)

http://fyzmatik.pise.cz/112549-collatzuv-problem.html

Pro zajímavost

Cyklus for – programujeme efektivně

Příklad – zjištění, zda x je prvočíslo

```
int x = …, jePrvocislo = 1, i;
for (i=2; i<=(int)sqrt(x); i++){
    if(x%i==0){
   jePrvocislo = 0;
      break;
    }
}
```

- Při nalezení prvního dělitele je zbytečné pokračovat ve zkoušení dalších hodnot
- Výraz `(int)sqrt(x)` je počítán v každém cyklu i když se jeho hodnota nemění:

```
int horni_mez = (int)sqrt(x);
for (i=2; i<= horni_mez; i++){...
```

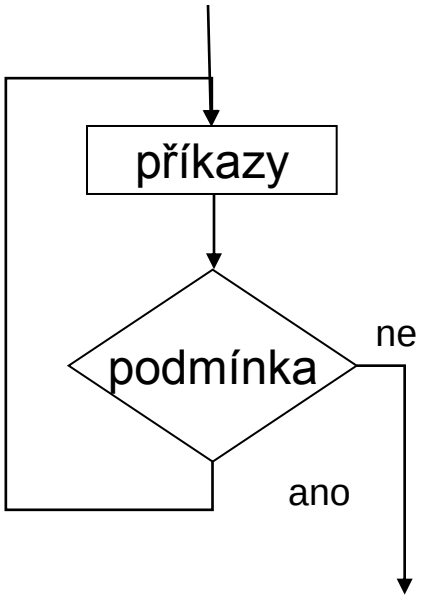
Metoda shozeného praporku

- Testování jedné podmínky pro všechny prvky dané posloupnosti (všechna sudá, všechna kladná čísla lichá, …
- Nastav praporek indikující splnění podmínky na hodnotu **true (1)**

- Postupně procházej posloupnost a pro každý nový člen ověř, že je podmínka stále splněna

- Pokud není splněna, shoď praporek
- Příklad: Zjistěte zda daná posloupnost obsahuje pouze **sudá čísla**

- Využijeme cyklus **do**



Metoda shozeného praporku - příklad

```
#include<stdio.h>
 #include<stdlib.h>
#include<math.h>
int main(void) {
    int vsechnaSuda = 1; // praporek, je zvednutý
    int cislo; // aktualní clen posloupnosti
    printf("Budou všechna čísla sudá?\n");
    printf("Zadávej postupně čísla, 0 -> stop.\n");
    do {
        scanf("%d", &cislo);
        if (cislo % 2 == 1) vsechnaSuda = 0;// pád praporku, liché
        else vsechnaSuda = 1;// častá chyba, nenahazovat!!
    } while (cislo != 0);
    if (vsechnaSuda)printf("Všechna sudá");
    else printf("Alespoň jedno bylo liché");
return 0;
}
```

Příkaz switch

- Příkaz *switch* (přepínač) umožňuje větvení do více větví na základě různých hodnot výrazu typu *int* nebo *char*
- Základní tvar příkazu:

switch (*výraz*) {

case *konstanta*$_1$: *příkazy*$_1$ break;

case *konstanta*$_2$: *příkazy*$_2$ break;

...

case *konstanta*$_n$: *příkazy*$_n$ break;

default : *příkazy*$_{def}$

}

kde *konstanty* jsou téhož typu, jako *výraz*

příkazy jsou posloupnosti příkazů

- Sémantika (zjednodušeně):
  - vypočte se hodnota *výrazu* a pak se provedou ty *příkazy*, které jsou označeny *konstantou* se stejnou hodnotu
  - není-li žádná větev označena hodnotou výrazu, provedou se *příkazy*$_{def}$

Příkaz switch - příklad

```
switch (n) {
        case 1: printf("*"); break;
        case 2: printf("**"); break;
        case 3: printf("***"); break;
        case 4: printf("****"); break;
        default: printf("---");
}
```

- Příkaz *break* dynamicky ukončuje větev
- Pokud jej neuvedeme, pokračuje se v provádění další větve
- Příklad: co se vypíše, má-li *n* hodnotu 3 a příkaz *switch* zapíšeme takto:

```
switch (n) {
        case 1: printf("*");
        case 2: printf("**");
        case 3: printf("***");
        case 4: printf("****");
        default: printf("---");
}
```

Příklad – pořadové číslo dne v roce

```c
/* prog4-5a.c */
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int den, mesic, rok, n=0;
    printf("zadejte den, mesic a rok: ");
    scanf("%d%d%d", &den, &mesic, &rok);
    switch (mesic) {
      case 1: n = den; break;
      case 2: n = 31+den; break;
      case 3: n = 59+den; break;
      case 4: n = 90+den; break;
      case 5: n = 120+den; break;
      case 6: n = 151+den; break;
      ...
      case 12: n = 334+den; break;
      default: printf("nedovoleny mesic\n");
    }
    if (mesic>2 && rok%4==0 && (rok%100!=0 || rok%400==0)) n = n+1;
    printf("%d.%d.%d je %d. den v roce\n", den, mesic, rok, n);
    return 0;
}
```