4. Řidící struktury, výrazy

B0B99PRPA – Procedurální programování

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

- Část 1 – Řídicí struktury
  - Větvení
  - Cykly
  - Konečnost cyklu
  - Příklady na cykly
- Část 2 – Výrazy
  - Výrazy a operátory
  - Přiřazení
- Část 3 – Zadání 3. domácího úkolu

Kódovací styl

Jak psát programy?

```
/* International Obfuscated C Code Contest 1990 */
v,i,j,k,l,s,a[99]
main()
{
  for(scanf("%d", &s);*a-s;v=a[*j=v]-a[i],k=i<s,j+=(v=j<s&&(!k&&!!
    printf(2+"\n\n%c"-!l<<!j)," #Q"[1^v?(1^j)&1:2])&&++l||a[i]<s&&v&&
    v-i+j&&v+i-j))&&!(l%=s),v||(i==j?a[i+=k]=0:++a[i])>=s*k&&++a[--i]
    );
}
```

https://www.ioccc.org/1990/baruch.hint

Interakce programu s uživatelem

- Při spuštení programu lze předat parametry (textové řetězce)
- Při ukončení programu lze předat návratovou hodnotu

Konvence: správné ukončení 0, jinak chybový kód.

- Při běhu programu lze číst ze standardního vstupu a zapisovat na standardní výstup
- Při spuštení programu lze vstup i výstup přesměrovat z/do souboru
- Každý terminálový program má standardní vstup (`stdin`) a výstup (`stdout`) a dále pak standardní chybový výstup (`stderr`), které lze v shellu presměrovat

```
$ ./a.out <stdin.txt >stdout.txt 2>stderr.txt
```

- Pro práci s chybovým výstupem lze využít funkci `fprintf`
  - Prvním argumentem soubor, jinak stejná syntaxe jako `printf`
  - Soubory `stdout`, `stdin` a `stderr` jsou definovány v `<stdio.h>`

Příklad programu s výstupen na stderr

```c
#include <stdio.h>

int main(void) {
  int ret = 0, a, b;

  fprintf(stdout, "Zadej jedno cele cislo: ");

  a = scanf("%d", &b);

  if (a > 1) {
    fprintf(stdout, "Bylo zadano cislo %d\n", b);
  }
  else {
    fprintf(stdout, "Cislo nebylo zadano spravne.\n");
    fprintf(stderr, "I/O error\n");
    ret = -1;
  }
  return ret;
}
```

Část I

Řídicí struktury

I. Řídicí struktury

Programový přepínač switch

- Příkaz `switch` (přepínač) umožnuje větvení programu do více větví na základě ruzných hodnot výrazu výčtového (celočíselného) typu, jako jsou např. `int`, `char`, `short`, `enum`.
- Tvar příkazu

```
switch (vyraz) {
  case konstanta1:
    prikazy1; break;
  case konstanta2:
    prikazy2; break;
  // ---
  case konstantaN:
    prikazyN; break;
  default:
    prikazydef;
}
```

- Přepínač `switch(vyraz)` větví program do `N` větví
- Hodnota `vyraz` je porovnávána s `N` konstantními výrazy typu `int` příkazy `case konstantai: ...`
- Hodnota `vyraz` musí být celočíselná a hodnoty konstant musí být navzájem různé
- Pokud je nalezena shoda, program pokračuje od tohoto místa dokud nenajde příkaz `break` nebo konec příkazu
- Pokud shoda není nalezena, program pokračuje nepovinnou sekcí `default`

Sekce **default** se zpravidla uvádí jako poslední

- Příkazy switch mohou být vnořené

Programový přepínač switch – příklad

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

```c
if (n == 1) {
  printf("*");
}
else if (n == 2) {
  printf("**");
}
else if (n == 3) {
  printf("***");
}
else if (n == 4) {
  printf("****");
}
else printf("---");
```

- Co se vypíše, pokud ve větvích nebudou příkazy `break` a `n=3`?

I. Řídicí struktury

Cykly while a do-while

- Tvar příkazu **while**

```
while (podminka)
  prikaz
```

- Tvar příkazu **do-while**

```
do
  prikaz
while (podminka)
```

Příklad

```
1  q = x;
2  while (q >= y) {
3    q = q - y;
4  }
```

```
1  q = x;
2  do {
3    q = q - y;
4  } while (q >= y);
```

- Jaká je hodnota proměnné `q` po skončení cyklu?

Cyklus while

- Tvar příkazu

```
while (podminka) prikaz
```

- Průběh cyklu
  1. Vyhodnotí se výraz `podminka`
  2. Pokud `podminka != 0`, provede se příkaz `prikaz`, jinak cyklus končí
  3. Opakování vyhodnocení výrazu `prikaz`
- Řídicí výraz
  - vyhodnocení na začátku cyklu

    Cyklus se nemusí provést ani jednou
  - aktualizace v těle cyklu, jinak je cyklus nekonečný

Cyklus do-while

- Tvar příkazu

```
do prikaz while (podminka);
```

- Průběh cyklu
  1. Provede se příkaz `prikaz`
  2. Vyhodnotí se výraz `podminka`
  3. Pokud `podminka != 0`, cyklus se opakuje
- Řídicí výraz
  - vyhodnocení na konci cyklu

    cyklus se provede vždy alespoň jednou
  - aktualizace v těle cyklu, jinak je cyklus nekonečný

Cyklus while – příklad

- Program na výpočet faktoriálu přirozeného čísla

$$n! = \prod_{k=1}^{n} k$$

```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;
}
```

Cyklus 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

```
1  int f = 1;
2  int i = 1;          // počáteční hodnota řídící proměnné
3  while (i<=n) {      // řídící výraz
4      f = f*i;        // tělo cyklu
5      i = i+1;        // změna řídící proměnné
6  }
```

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

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

Cyklus for

- Tvar příkazu

  `for (inicializace; podminka; zmena) prikaz`

- Odpovídá cyklu while ve tvaru:

```
inicializace;
while (podminka) {
  prikaz;
  zmena;
}
```

Příklad

```
1  for (int i = 0; i < 10; ++i) {
2    printf("i = %i\n", i);
3  }
```

Změnu řídicí proměnné lze zapsat operátorem inkrementace `++` nebo dekrementace `--`, lze též použít zkrácený zápis přiřazení, např. `+=`.

- Výrazy `inicializace` a `zmena` mohou být libovolného typu
- Libovolný z výrazů lze vynechat
- **break** – cyklus lze nuceně opustit příkazem `break`
- **continue** – část těla cyklu lze vynechat příkazem `continue`
- Při vynechání řídicího výrazu `podminka` se cyklus bude provádět nepodmíněně

Cyklus for – příklady

- Správný zápis

```
for (i = 0; i < 10; i++) {}
for (; a < 4.0; a += 0.2) {}
for (; i < 10;) {}
for (;; i++) {}  // nekonečný cyklus
for (;;) {}      // nekonečný cyklus, ekv.  while(1)
for (i = 1; i < 10; printf("%i\n", i), i++);
```

- Nesprávné použití

```
for () {}                    // chybí středníky
for (i = 1, i == x, i++) {} // čárky místo středníků
for (x < 4) {}               // chybí středníky
```

Příkaz návratu na vyhodnocení řídicího výrazu continue

- Příkaz `continue` lze použít pouze v těle cyklu
  - `for ()`, `while ()`, `do-while ()`
- Příkaz způsobí přerušení vykonávání těla cyklu a nové vyhodnocení řídicího výrazu

Příklad

```
1  int i;
2  for (i = 0; i < 15; ++i) {
3    if (i % 2 == 0) {
4      continue;
5    }
6    printf("i: %2d\n", i);
7  }
```

```
i:  1
i:  3
i:  5
i:  7
i:  9
i: 11
i: 13
```

lec04/loop-continue.c

Příkaz nuceného ukončení cyklu break

- Příkaz `break` lze použít pouze v těle řídících struktur
  - `for()`, `while()`, `do...while()`, `switch()`
- Program pak pokračuje následujícím příkazem.

Příklad

```
1  int i = 10;
2  while (i > 0) {
3    if (i == 5) {
4      printf("opoustim cyklus\n");
5      break;
6    }
7    printf("i: %2d", i--);
8  }
9  printf("konec cyklu i: %d\n", i);
```

```
i: 10
i: 9
i: 8
i: 7
i: 6
opoustim cyklus
konec cyklu i: 5
```

lec04/loop-break.c

Příkaz nepodmíněného lokálního skoku goto

- Syntax: `goto navesti;`
- Příkaz předá řízení na místo určené návěštím `navesti`, lze použít pouze v těle funkce
- Skok nesmí směřovat dovnitř bloku, který je vnořený do bloku, kde je příslušné `goto` umístěno

Příklad

```
1  int test = 3;
2  for (int i = 0; i < 10; i++) {
3    if (i == test) {
4      goto OUT;
5    }
6    printf ("i = %i\n", i);
7  }
8  return 0;
9  OUT: return -1;
```

I. Řídicí struktury

Konečnost cyklu

- Konečnost algoritmu – pro přípustná data skončí v konečné době
- Aby byl algoritmus konečný, musí každý cyklus v něm uvedený skončit po konečném počtu kroků
- Jedním z důvodu neukončení programu je **zacyklení**
- **Zacyklení** – program opakovaně vykonává cyklus, jehož podmínka ukončení není nikdy splněna

```
1  while (i != 0) {
2    j = i - 1;
3  }
```

- Cyklus se neprovede ani jednou,
- nebo neskončí.
- Záleží na hodnotě řídicí proměnné `i` před voláním cyklu

Základní pravidlo konečnosti cyklu

- Provedením těla cyklu se musí změnit hodnota proměnné použité v podmínce ukončení

```
for (int i = 0; i < 5; ++i) {
// --
}
```

- Uvedené pravidlo konečnost cyklu nezaručuje

```
int i = -1;
while ( i < 0 ) {
  i = i - 1;
}
```

- Konečnost cyklu závisí na hodnotě proměnné před vstupem do cyklu.

Konečnost cyklu

```
1  while (i != n) {
2    // příkazy nemenici hodnotu promenne i
3    i++;
4  }
```

- Vstupní podmínka konečnosti uvedeného cyklu
  - $i \leq n$ pro celá čísla

Jak by vypadala podmínka pro proměnné typu double?

- Splnění vstupní podmínky konečnosti cyklu musí zajistit příkazy předcházející cyklu
- Zabezpečený program testuje přípustnost vstupních dat

I. Řídicí struktury

Je číslo palindrom?

- **Palindrom** – symetrický útvar, stejný význam při čtení zleva i zprava
- Příklady: 131, 5896985, ...
- Možné řešení: výpočet reverzního čísla

```
1  int n, num, rev;
3  scanf("%d", &n);
5  num = n;
7  while(n != 0) {
8    rev = (rev * 10) + (n % 10);
9    n  /= 10;
10 }
12 if (rev == num) printf("cislo %d je palindrom", n);
13 else printf("cislo %d neni palindrom", n);
```

Prohození první a poslední číslice celého čísla

Vstup: celé číslo X – AxxxB
Výstup: celé číslo Y – BxxxA

1: $B \leftarrow X\%10$
2: $p \leftarrow log10(X)$ ▷ počet číslic
3: $A \leftarrow X/pow(10, p)$

4: $Y \leftarrow B * pow(10, p)$
5: $Y \leftarrow Y + num\%pow(10, p)$
6: $Y \leftarrow Y - B$
7: $Y \leftarrow Y + A$

Část II

Výrazy

II. Výrazy

Výrazy a operátory

Přiřazení

Výrazy

- Výraz předepisuje výpočet hodnoty určitého vstupu
- Výraz může obsahovat
  - **operandy** – proměnné, konstanty, volání funkcí nebo jiné výrazy
  - **operátory**
  - **závorky**
- Pořadí operací předepsaných výrazem je dáno **prioritou** a **asociativitou** operátorů

Příklad

```
1  10 + x * y  // poradi vyhodnoceni 10 + (x * y)
2  10 + x + y  // poradi vyhodnoceni (10 + x) + y
```

- Operátor * má vyšší prioritu než operátor +, operátor + je asociativní zleva

Výrazy a operátory

- **Výraz** se skládá z operátorů a operandů
  - Výraz sám může být operandem
  - Výraz má typ a hodnotu (Pouze výraz typu `void` hodnotu nemá.)
  - Výraz zakončený středníkem `;` je příkaz
- **Operátory** jsou vyhrazené znaky (ev. sekvence) pro zápis výrazu
  - Postup výpočtu výrazu s více operátory je dán prioritou operátoru
  - Postup výpočtu lze předepsat použitím kulatých závorek `(` a `)`
  - Obecně (mimo konkrétní případy) není pořadí vyhodnocení operandů definováno (nezaměňovat s asociativitou!)
    - Např. pro součet `f1() + f2()` není definováno, který operand se vyhodnotí jako první (tj. jaká funkce se zavolá jako první).
    - Pořadí vyhodnocení je definováno pro operandy v logickém součinu `AND` a součtu `OR`
- **Nedefinované chování** – vyhodnocení některých specifických výrazů není definováno a záleží na překladači: `i = ++i + i++;`

https://en.cppreference.com/w/c/language/eval_order

Operátory

- Binární operátory
  - Aritmetické – sčítání, odčítání, násobení, dělení
  - Relační — porovnání hodnot (menší, větší, . . .)
  - Logické — logický součet a součin
  - Operátor priřazení – na levé straně operátoru `=` je proměnná
- Unární operátory
  - indikující kladnou/zápornou hodnotu: `+` a `-`

    operátor `-` modifikuje znaménko výrazu za ním
  - modifikující proměnou: `++` a `--`
  - logický operátor doplněk: `!`
  - bitová negace (negace bit po bitu): `~`
  - operátor pretypování: `(jméno typu)`
- Ternární operátor
  - podmíněné přiřazení hodnoty: `?  :`

`http://www.tutorialspoint.com/cprogramming/c_operators.htm`

Aritmetické operátory

- Operandy aritmetrických operátorů mohou být libovolného číselného typu

  Výjimkou je operátor zbytek po dělení % definovaný pro int

*	Násobení	`x*y`	
/	Dělení	`x/y`	
%	Dělení modulo	`x%y`	Zbytek po dělení `x` a `y`
+	Sčítání	`x+y`	
-	Odčítání	`x-y`	
+	Kladné zn.	`+x`	
-	Záporné zn.	`-x`	
++	Inkrementace	`++x`, `x++`	Inkrementace před/po vyhodnocení výrazu
- -	Dekrementace	`--x`, `x--`	Dekrementace před/po vyhodnocení výrazu

Unární aritmetické operátory

- Unární operátory `++` a `--` mění hodnotu svého operandu

  Operand musí být **l-hodnota**, tj. výraz, který má adresu, kde je uložena hodnota výrazu (např. proměnná)

  - lze zapsat prefixově, např. `++x` nebo `--x`

    Operace je provedena **před** vyhodnocením výrazu.

  - nebo postfixově např. `x++` nebo `x--`

    Operace je provedena **po** vyhodnocení výrazu.

  - v obou případech se však liší výsledná hodnota výrazu!

Příklad

```
1  int i = 1, a;
2  a = i++;       // i=2 a=1
3  a = ++i;       // i=3 a=3
4  a = ++(i++);   // nelze, hodnota i++ neni l-hodnota
```

Relační operátory

- Operandy relačních operátorů mohou být
  - číselného typu,
  - ukazatele shodného typu nebo
  - jeden z nich `NULL` nebo
  - typ `void`

Menší než	`x<y`	`1` pro `x` je menší než `y`, jinak `0`
Menší nebo rovno	`x<=y`	`1` pro `x` menší nebo rovno `y`, jinak `0`
Větší	`x>y`	`1` pro `x` je vetší než `y`, jinak `0`
Větší nebo rovno	`x>=y`	`1` pro `x` vetší nebo rovno `y`, jinak `0`
Rovná se	`x==y`	`1` pro `x` rovno `y`, jinak `0`
Nerovná se	`x!=y`	`1` pro `x` nerovno `y`, jinak `0`

Relační operátory – příklad

```
1  #include <stdio.h>
3  int main()
4  {
5    int a = 10, b = 20, c = 30;
7    // (c > b > a) vyhodnoceno jako ((c > b) > a),
8    // asociativita '>' je zleva doprava.
9    // takze ((30 > 20) > 10) --> (1 > 20)
11   if (c > b > a)
12     printf("TRUE");
13   else
14     printf("FALSE");
16   return 0;
17 }
```

lec04/chained-comparison.c

Logické operátory

- Operandy mohou být číselné typy nebo ukazatele
- Výsledek 1 má význam true, 0 má význam false
- Ve výrazech `&&` a `||` se vyhodnotí nejdříve levý operand – pokud je výsledek dán levým operandem, pravý se již nevyhodnocuje

AND	`x&&y`	`1` pokud `x` ani `y` není rovno `0`, jinak `0`
OR	`x||y`	1 pokud alespoň jeden z `x`, `y` není rovno `0`, jinak `0`
NOT	`!x`	`1` pro `x` rovno `0`, jinak `0`

Bitové operátory

- Bitové operátory vyhodnocují operandy bit po bitu

Bitové AND	`x&y`	`1` když `x` i `y` je rovno `1`
Bitové OR	`x|y`	`1` když `x` nebo `y` je rovno `1`
Bitové XOR	`x^y`	`1` pokud pouze `x` nebo pouze `y` je `1`
Bitové NOT	`~x`	`1` pokud `x` je rovno `0`
Posun vlevo	`x<<y`	posun `x` o `y` bitů vlevo
Posun vpravo	`x>>y`	posun `x` o `y` bitů vpravo

Příklad

```
21 & 56     = 00010101 & 00111000     = 00010000
21 | 56     = 00010101 | 00111000     = 00101101
21 ^ 56     = 00010101 ^ 00111000     = 00111101
~21         = ~00010101               = 11101010
21 << 2     = 00010101 << 2           = 01010100
21 >> 1     = 00010101 >> 2           = 00001010
```

Operace bitového posunu

- Operátory bitového posunu posouvají celý bitový obraz proměnné nebo konstanty o zvolený počet bitů vlevo nebo vpravo
- Při posunu vlevo jsou uvolněné bity zleva doplňovány 0
- Při posunu vpravo jsou uvolněné bity zprava
  - u čísel kladných nebo čísel typu `unsigned` plněny `0`
  - u záporných čísel bud plněny `0` (logický posun) nebo `1` (aritmetický posun vpravo), dle implementace překladače.
- Operátory bitového posunu mají nižší prioritu než aritmetické operátory!
  - `i << 2 + 1` znamená `i << (2 + 1)`

Bitové operátory – příklad 1/2

- Nastavení N-tého bitu celého čísla

```
unsigned char cislo;
cislo |= (1<<N);
```

- Nulování N-tého bitu celého čísla

```
unsigned char cislo;
cislo &= ~(1<<N);
```

- Inverze N-tého bitu celého čísla

```
unsigned char cislo;
cislo ^= (1<<N);
```

- Získání hodnoty N-tého bitu celého čísla

```
unsigned char cislo;
char bit = (cislo & (1<<N)) >> N;
```

Bitové operátory – příklad


```c
#include <stdio.h>
#include <stdint.h>
int main()
{
  uint8_t a = 10;
  for (int i = 7; i >= 0; --i)
  {
    printf ("%i", (a >> i) & 1);
  }
  printf("\n");
  return 0;
}
```

lec04/bites.c

Operátory přístupu do paměti

- V C lze přímo přistupovat k adrese paměti proměnné, kde je hodnota proměnné uložena
- Přístup do paměti je prostřednictvím ukazatele (pointeru)
  - nesmírně silná vlastnost programovacího jazyka
  - vyžaduje pochopení principu práce s pamětí
  - podrobněji v 5. přednášce

`&`	adresa proměnné	`&x`	ukazatel na `x`
`*`	neprímá adresa	`*p`	proměnná adresovaná `p`
`[]`	prvek pole	`x[i]`	prvek pole `x` s indexem `i`
`.`	prvek struct/union	`s.x`	prvek `x` struktury `s`
`->`	prvek struct/union	`p->x`	prvek struktury adresovaný `p`

Další operátory

`()`	volání funkce	`f(x)`	volání funkce `f` s argumentem `x`
`(type)`	přetypování	`(int)x`	změna typu `x` na `int`
`sizeof`	velikost prvku	`sizeof(x)`	velikost `x` v bajtech
`?:`	podmíněný příkaz	`x?y:z`	proveď `y` pokud `x != 0` jinak `z`
`,`	postupné vyhodnocení	`x,y`	vyhodnotí `x` pak `y`, výsledek operátoru je výsledek posledního výrazu

- Operandem operátoru `sizeof()` může být jméno typu nebo výraz

```
1  int a = 10;
2  printf("%lu %lu\n", sizeof(a), sizeof(a + 1.0));
```

- Příklad použití operátoru čárka

```
1  for (c = 1, i = 0; i < 3; ++i, c += 2)
2    printf("i: %d c: %d\n", i, c);
```

Operátor přetypování

- Změna typu za běhu programu se nazývá přetypování
  - Explicitní přetypování (cast) zapisuje programátor

```
1  int i;
2  float f = (float)i;
```

  - Implicitní přetypování provádí překladač automaticky při překladu
- Možné konverze při přiřazení

typ hodnoty	typ proměnné	poznámka ke konverzi
racionální	kratší racionální	zaokrouhlení mantisy
racionální	delší racionální	doplnění mantisy nulami
racionální	celočíselný	odseknutí necelé části
celočíselný	racionální	možná ztráta přesnosti
celočíselný	kratší celočíselný	odseknutí vyšších bitů
celočíselný unsgn.	delší celočíselný	doplnění nulových bitů
celočíselný sgn.	delší celočíselný	rozšíření znaménka

Priority operátorů

priorita	operátory	asociativita
1	`. -> () []`	zleva
2	`+ - ++ -- ! ~ (typ) & * sizeof`	zprava
3	`* / %`	zleva
4	`+ -`	zleva
5	`<< >>`	zleva
6	`< > <= >=`	zleva
7	`== !=`	zleva
8	`&`	zleva
9	`^`	zleva
10	`|`	zleva
11	`&&`	zleva
12	`||`	zleva
13	`?:`	zprava
14	`= += -= *= /= %= <<= >>= &= |= ^=`	zprava
15	`,`	zleva

II. Výrazy

Výrazy a operátory

Přiřazení

Přiřazení

- Nastavení hodnoty proměnné – inicializace místa v paměti
- Tvar přiřazovacího operátoru: **proměnná** = **výraz**

Výraz je literál, proměnná, volání funkce, ...

- Proměnné lze přiřadit hodnotu výrazu pouze identického typu
- Příklad implicitní konverze pri přirazení

```
1 // implicitni konverze 320.4 -> 320
2 int i = 320.4;
3 // implicitni oriznuti 320 -> 64
4 char c = i;
```

- Zkrácený zápis přiřazení: **proměnná** **operátor** = **výraz**

```
1 int i = 10;
2 double j = 12.6;
3 i = i + 1;
4 j = j / 0.2;
```

```
1 int i = 10;
2 double j = 12.6;
3 i += 1;
4 j /= 0.2;
```

Část III

Zadání 3. domácího úkolu

Zadání 3. domácího úkolu (HW03)

Téma: První cyklus

- **Motivace:** Automatické načítání vstupních dat
- **Cíl:** Využití cyklu jako základní programového konstrukce pro hromadné zpracování dat.
- **Zadání:** `https://cw.fel.cvut.cz/wiki/courses/b0b99prpa/hw/hw03`
  - Zpracování libovolně dlouhé posloupnosti celých čísel
  - Výpis načtených čísel
  - Výpis statistiky vstupních čísel
    - Počet načtených čísel;
    - Počet kladných a záporných čísel a jejich procentuální zastoupení na vstupu
    - Četnosti výskytu sudých a lichých čísel a jejich procentuální zastoupení na vstupu
    - Průměrná, maximální a minimální hodnota načtených čísel
- **Termín odevzdání:** **24.10.2020, 23:59:59**

Shrnutí přednášky

Diskutovaná témata

- Řidící struktury
  - Větvení
  - Cyklus
  - Přepínač
  - Příkazy `break` a `continue`
  - Konečnost cyklů
- Operátory
  - Přehled operátorů a jejich priorit
  - Přiřazení a zkrácený způsob zápisu

- Příště: pole, ukazatel, textový řetězec

Diskutovaná témata

- Řidící struktury
  - Větvení
  - Cyklus
  - Přepínač
  - Příkazy `break` a `continue`
  - Konečnost cyklů
- Operátory
  - Přehled operátorů a jejich priorit
  - Přiřazení a zkrácený způsob zápisu

- Příště: pole, ukazatel, textový řetězec