



# Principy vyšších programovacích jazyků

---

Jitka Kreslíková, Aleš Smrčka

2022

Fakulta informačních technologií  
Vysoké učení technické v Brně

IZP – Základy programování



# Principy vyšších programovacích jazyků

---

- ☐ **Ukládání informací**
- ☐ **Zpracování informací**



# Ukládání informací

---

**ISC – Počítačový seminář**

- ☐ Reprezentace údajů.
- ☐ Datové struktury.
- ☐ Datové typy.
- ☐ Prvky programovacích jazyků.
- ☐ Proměnné, konstanty.



# Zpracování informací

---

- ❑ Proměnné a deklarace.
- ❑ Operátory a výrazy
  - Aritmetické operátory
  - Logické a relační operátory
  - Bitové operátory
  - Adresové operátory
  - Operátor přetypování
  - Operátor sizeof
  - Podmíněný operátor (ternární)
  - Operátor volání funkce
  - Přístupové operátory
  - Operátor čárka
  - Priorita operátorů



# Proměnné a deklarace

---

**Datový objekt** – obecné označení jakéhokoli údaje uloženého v operační paměti.

!!! Nezaměňovat s pojmem *objekt* v objektově orientovaném programování !!!.

Při programování potřebujeme uchovávat data.

Mohou to být:

- údaje zadané uživatelem (jeho jméno, věk, adresa, posloupnost čísel, kterou chce uživatel seřadit, ...),



# Proměnné

---

- ❑ údaje načtené z diskového souboru (textové, číselné, ...),
  - ❑ různé dočasné a pomocné hodnoty (počet opakování),
  - ❑ operandy, výsledky operací.
- 
- ❑ Paměťová místa ve kterých uchováváme data označujeme jako **proměnné**.
  - ❑ Proměnná je datový objekt určitého typu, hodnota tohoto objektu se může za běhu programu měnit.



# Proměnné a deklarace

---

**Deklarace proměnné** je konstrukce, která přidělí proměnné jméno a typ, ale nevytvoří ji. Najdeme je někdy v hlavičkových souborech, i když jejich použití tímto způsobem je neobvyklé.

**Definice proměnné** je v jazyce C současně deklarací a kromě jména a datového typu přidělí proměnné i paměťový prostor.

*Příklady:*

**int** i, j;

**char** znak;

**float** podil, odmocnina;

## Inicializace proměnné

Pokud již při definování proměnné víme, jakou bude mít počáteční hodnotu, můžeme ji inicializovat.

*Příklady:*

**char** znak = 'a';

**int** rozmer = 100;



# Operátory a výrazy

---

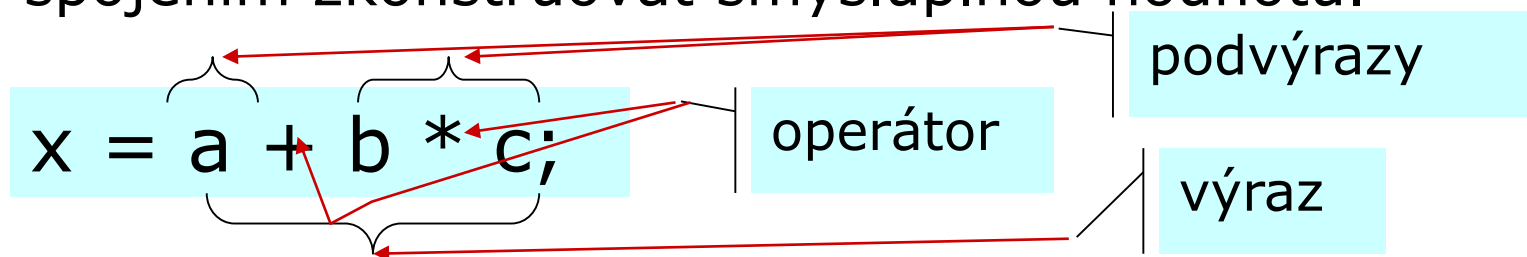
- **Výraz** – konstrukce jazyka, která má hodnotu (nějakého datového typu).
- **Operand** – je částí výrazu na kterou je aplikován jeden z **legálních** operátorů.
  - Má také hodnotu konkrétního datového typu.
  - Někdy se operandům říká podvýrazy, protože operandem může být i složitější výraz, např.  $(a+b)*(c+d)$ .





# Operátory a výrazy

- ❑ **Operátor** – určuje, jakým způsobem se z operandů získá hodnota výrazu.
  - Pořadí vyhodnocování podvýrazů ve výrazu je dáno prioritami jednotlivých operátorů nebo závorkami, pokud byly použity.
  - Datové typy operandů určují, které operátory je možné v daném okamžiku použít.
  - Na konkrétním typovém systému jazyka záleží, zda se aplikace operátoru na různé datové typy vyhodnotí jako chybná konstrukce nebo zda je možno tímto spojením zkonstruovat smysluplnou hodnotu.





# Operátory a výrazy

## □ Výraz x příkaz

### ■ Rozdíl

- výraz nabývá dále použitelné hodnoty,
- primárním úkolem příkazu je vykonat nějaký kód.

### ■ Pokud je výsledkem kódu v příkazu nějaká hodnota a nejde o výraz s přiřazením, hodnota se zahodí (nepoužije, bude se ignorovat).

### ■ V jazyce C se z výrazu stane příkaz tím, že jej ukončíme středníkem (samozřejmě v místě, kde to dává smysl - uprostřed podmínky cyklu to samozřejmě nejde).

### ■ Samotný středník (;) představuje prázdný příkaz (null statement), který se někdy používá například v cyklech.

`x = 64 // výraz s přiřazením`

`x = 64; // příkaz`

`printf("Hello world!"); // příkaz`

`a + b; // příkaz – výsledek výrazu se "zahodí"/nepoužije`

"mrtvý kód"



# Operátory a výrazy

## □ Operátor přiřazení, L-hodnota a P-hodnota

- Operátor přiřazení (=) kopíruje hodnotu výrazu na své pravé straně do proměnné na levé straně.
- V této souvislosti se mluví o L-hodnotě a P-hodnotě (anglicky L-value a R-value).
- **L-hodnota** – je objekt v paměti, kterému lze přiřadit hodnotu.
  - proměnná, prvek pole či struktury nebo paměť odkazovaná přes ukazatel.
- **P-hodnota** – výraz, který má vždy hodnotu a který vystupuje na pravé straně přiřazovacího operátoru.

L-hodnota

```
float x = -c / b;  
pole[i] = 25;  
*cislo = 4;  
souradnice->x = 17;
```

```
(a + b) = 68; // nelze!  
faktorial(5) = 120; // nelze
```

P-hodnota



# Operátory a výrazy

## □ Přířazení

- V jazyce C zavedeno jako operátor (=).
- V praxi to znamená, že operace přiřazení má vlastní hodnotu, kterou lze použít dále ve složitějším výrazu.
- Může se tedy vyskytovat i v P-výrazech.

používat opatrně

*Příklad:*

$x1 = x2 = -c/b;$

P-hodnota



# Operátory a výrazy

- Jazyk C ještě poskytuje celou kolekci rozšířených přiřazovacích operátorů, které zjednodušují zápis často používaných výrazů typu:

L-hodnota = L-hodnota *operátor* výraz;

- V jazyce C jsou definovány tyto rozšířené přiřazovací operátory:

`+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=`

*Příklad:*

`x *= a + b;` je totéž jako `x = x * (a + b);`

`x *! = c + b;`

// syntaktická chyba – mezi operátorem a = nesmí být mezera



# Aritmetické operátory

---

- V jazyce C definovány nad číselnými datovými typy.
- Tři skupiny:
  - unární,
  - binární aritmetické operátory,
  - speciální unární operátory.



# Aritmetické operátory

[HePa13]

## ■ unární,

Unární plus	Unární mínus
+	-

- Ke změně znaménka výrazu.
- Lze použít jak s celočíselnými tak i s racionálními typy.

*Příklad:*

*i = -5;*



# Aritmetické operátory

[HePa13]

## ■ binární aritmetické operátory,

sčítání	odčítání	násobení	dělení	modulo
+	-	*	/	%

### ○ Operátor modulo (%)

- zbytek po celočíselném dělení
- jen s celočíselnými typy.

Pozor! V jazyce C může operace modulo produkovat i záporné hodnoty ( $-2\%4 == -2$ ).

*Příklad:*

```
x = a * b;
```





# Aritmetické operátory

---

- Operátory (+ - \* /) fungují kontextově podle datového typu operandů.

`int op int ...` celočíselné dělení – výsledkem je `int`

`int op float ...` dělení v pohyblivé řádové čárce – výsledkem je `float`

`float op int ...` dělení v pohyblivé řádové čárce – výsledkem je `float`

`float op float ...` dělení v pohyblivé řádové čárce – výsledkem je `float`

- Multiplikativní operátory (\*, /, %) mají vyšší prioritu než aditivní operátory (+, -), takže matematické výrazy můžeme zapisovat přirozeným způsobem:

`x = 10 + 2*3; // výsledkem je 16`



## ■ speciální unární operátory,

Unární přičtení jedničky	Unární odečtení jedničky
++	--

- Oba existují jako předpony (prefix) a jako přípony (suffix). Obě verze se liší svým významem

## L-hodnota ++

- inkrementace po použití hodnoty
- nejprve vrátí původní hodnotu (pokud je použit ve výrazu) a poté přičte k L-hodnotě jedničku

```
i++; // inkrementace proměnné i
```



## ++ L-hodnota

- inkrementace před použitím hodnoty.
- nejprve přičte jedničku a teprve potom vrátí hodnotu výsledku (pokud je použit ve výrazu),
- tyto operátory lze použít výhradně na L-hodnotu, protože do ní ukládají výsledek.

28++ ... nelze

--(a + b) ... nelze

i = i++; // POZOR! Nedefinovaná hodnota – není jasné, v jakém pořadí se bude modifikovat hodnota proměnné i



# Logické a relační operátory

---

- ❑ Pracují s logickými hodnotami.
- ❑ V jazyce C produkují výsledky typu **int**
  - C interpretuje hodnotu 0 jako nepravdu,
  - nenulovou hodnotu jako pravdu.
- ❑ ISO C99 – datový typ **bool** (**false**, **true**)
  - kompatibilní s typem **int**
  - nutno dovézt rozhraní `<stdbool.h>`
  - identifikátor **true** je definován jako konstanta s hodnotou 1
    - Nikdy nedělejte toto: `if (vyraz == true)`



# Logické a relační operátory [HePa13]

Logický součin	Logický součet	Logická negace
&&		!

Rovnost	Nerovnost	Menší než	Menší nebo rovno	Větší než	Větší nebo rovno
==	!=	<	<=	>	>=

## Zkrácené vyhodnocování výrazů

- ❑ Jazyk C používá zkrácené vyhodnocování (short circuit) logických výrazů.



# Logické a relační operátory

- ❑ Logické výrazy se vyhodnocují zleva doprava a jakmile je jistý výsledek, vyhodnocování se ukončí.

```
if (x != 0 && y / x < z)
```

- ❑ Zde nedojde k dělení nulou, protože pokud  $x$  je rovno nule, po vyhodnocení podvýrazu  $x \neq 0$ , je jasný výsledek celého výrazu a druhá část už se nebude vyhodnocovat.
- ❑ V jazyce C je priorita aritmetických a relačních operátorů vyšší než priorita logických operátorů  $\Rightarrow$  logické výrazy není třeba přehnaně závorkovat.



# Bitové operátory

[HePa13]

- ❑ Pro práci s čísly na úrovni jednotlivých bitů.
- ❑ Jazyk C poskytuje šest bitových operátorů.
- ❑ Tyto operátory mají odlišnou funkci od logických operátorů!

Bitový součin (AND)	Bitový součet (OR)	Bitová negace (NOT)	Bitový exkluzivní součet (XOR)	Bitový posuv vlevo	Bitový posuv vpravo
&		~	^	<<	>>



# Bitové operátory

---

- ❑ Výsledkem je číselná hodnota!
- ❑ Výsledkem není logická hodnota!

*Příklady:*

`0xF0 && 0x88` // logický součin – výsledkem je true (1)

`0xF0 & 0x88` // bitový součin – výsledkem je 0x80





# Adresové operátory

[HePa13]

- Ve spojení s ukazateli.

Referenční operátor	Dereferenční operátor
<i>&amp;jmeno</i>	<i>*ukazatel</i>

- Referenční operátor - vrací adresu proměnné.
- Dereferenční operátor - vrací hodnotu paměťového místa, odkazovaného ukazatelem.

Příklady:

```
int *uCislo = &cislo;
```

```
// (int *) typ ukazatel na int, ne dereferenční operátor
```

```
int hodnota = *uCislo; // nyní jde o dereferenční operátor
```



## ***(typ)výraz***

- ❑ Pro explicitní změnu datového typu výrazu.
- ❑ Programátor přebírá zodpovědnost za chování typového systému jazyka.
- ❑ Nevhodné použití explicitních konverzí způsobuje problémy!
- ❑ Programátor by si měl být vědom, jaké dopady bude mít tato násilná změna datového typu.



# Operátor přetypování

---

- ❑ Často používán s ukazateli, zejména s obecným ukazatelem (`void *`).

*Příklady:*

`(char)ordinální_hodnota` ... získání znaku z ordinální hodnoty

`(int)výraz_float` ... oříznutí desetinné části (Pozor! Problémy s velkými čísly. Raději `trunc()`, `round()`, ...)

`(float)výraz_int` ... převod na racionální typ



# Operátor sizeof

---

[HePa13]

**sizeof(typ)**

**sizeof(výraz)**

- ❑ Vrací počet bajtů, které zabírá konkrétní datový typ nebo výraz.
- ❑ Norma jazyka nezaručuje přesnou velikost číselných datových typů!



# Operátor sizeof

- Operátor vrací velikost v bajtech.

*Příklady:*

`sizeof(int)` // 2, 4 nebo 8 – záleží na procesoru

`sizeof(promenna)` // rozměr datového typu proměnné

- Zvláštní význam má u polí – vrací součet velikostí jeho jednotlivých položek.
  - Ale ne u polí tvořených přes ukazatel!

*Příklad:*

```
int pole[10];
```

```
sizeof(pole) // == 10*sizeof(int)
```

```
           // == 40 – pokud sizeof(int) == 4
```



# Podmíněný operátor (ternární)

- ❑ Pro vytváření výrazů, jejichž výsledek závisí na vstupní podmínce.
- ❑ Syntaxe:  
*podmínka ? varianta\_true : varianta\_false*
- ❑ Pokud je podmínka pravdivá, má výraz hodnotu *varianta\_true*, jinak má hodnotu *varianta\_false*.

*Příklad:*

```
int max = (a > b)? a : b; // maximum z a, b
```

!! Co když se a rovná b?



# Podmíněný operátor (ternární)

---

## Doporučení:

- ☐ Používat pouze ve výrazech.
- ☐ Nezneužívat jako náhradu příkazu if.
- ☐ Nevnořovat ternární operátory do sebe – vede k nesrozumitelnému kódu.



# Operátor volání funkce

---

**jméno\_funkce()**

**jméno\_funkce(parametry)**

- ❑ V jazyce C je pro zavolání funkce nutné použít operátor volání funkce, a to i pro funkce, které nemají žádné parametry.
- ❑ Identifikátor funkce bez závorek má význam ukazatele na tuto funkci.

*Příklad:*

```
getchar();  
printf("HELLO.");
```





# Přístupové operátory

[HePa13]

Indexování	Přístup k prvku struktury	Přístup k prvku struktury přes ukazatel na strukturu
<b>[]</b>	<b>.</b>	<b>-&gt;</b>

*Příklad:*

```
pole[i+2] = pole[i+1]+pole[i];  
souradnice.x = 10;  
souradnice.y = 20;
```

(\*ukazatel).prvek    ekvivalentní s    ukazatel->prvek



# Operátor čárka

*Výraz, výraz*

- ❑ Vyhodnocuje se zleva doprava a celkový výraz nabývá hodnoty nejpravějšího podvýrazu.
- ❑ Používá se například v příkazu `for`.

!! zde to není  
operátor přiřazení

*Příklad:*

`for (int i = 0, j = 10; i != j; i++, j--) // i++, j-- - použití operátoru čárka`

`int vysledek = (a+b, c+d); // výsledkem je c + d, a+b se zahodí`  
`vysledek = a+b , c+d; // výsledkem je a+b, c+d je mrtvý kód`

**Zásada:**

priorita "=" > priorita ","

raději nepoužívat, smysluplně se dá použít jen v minimálním počtu případů.



# Priorita operátorů

---

- ❑ Udává pořadí vyhodnocení podvýrazů.
- ❑ Vhodně zavedená hierarchie priorit operátorů dovoluje značně redukovat počet závorek ve výrazech.
- ❑ Pomocí závorek lze měnit pořadí vyhodnocení výrazu podle potřeby.
- ❑ Ve všech jazycích je přesně specifikována priorita všech operátorů, (v některých jsou však priority navrženy neprakticky – viz Pascal).
- ❑ Platí pravidlo:  
"Nejsi-li si jistý prioritou operátorů, závorkuj."

*Příklad:*

$X = 3 + 2 * 10;$  // výsledek 23

$x = (3 + 2) * 10;$  // výsledek 50



# Priorita operátorů

[HePa13]

Priorita	Operátory	Asociativita	skupina op.
1.	() [] -> .	zleva doprava	primární
2.	! ~ ++ -- + - (typ) * & <b>sizeof</b>	zprava doleva	unární
3.	* / %	zleva doprava	multiplikativní
4.	+ -	zleva doprava	aditivní
5.	<< >>	zleva doprava	posuny
6.	< <= > >=	zleva doprava	relační
7.	== !=	zleva doprava	relační
8.	&	zleva doprava	bitový součin
9.	^	zleva doprava	exkl. bit. souči.
10.		zleva doprava	bitový součet
11.	&&	zleva doprava	logický součin
12.		zleva doprava	logický součet
13.	?:	zprava doleva	ternární podm.
14.	= += -= *= /= %= >>= <<= &=  = ^=	zprava doleva	přiřazení
15.	,	zleva doprava	op. čárka



# Priorita operátorů

[HePa13]

Priorita	Operátory	Asociativita	skupina op.
1.	() [] -> .	zleva doprava	primární
2.	! ~ ++ -- + - (typ) * & sizeof	<b>zprava doleva</b>	unární
3.	* / %	zleva doprava	multiplikativní
4.	+ -	zleva doprava	aditivní
5.	<< >>	zleva doprava	posuny
6.	< <= > >=	zleva doprava	relační
7.	== !=	zleva doprava	relační
8.	&	zleva doprava	bitový součin
9.	^	zleva doprava	exkl. bit. souči.
10.		zleva doprava	bitový součet
11.	&&	zleva doprava	logický součin
12.		zleva doprava	logický součet
13.	?:	<b>zprava doleva</b>	ternární podm.
14.	= += -= *= /= %= >>= <<= &=  = ^=	<b>zprava doleva</b>	přiřazení
15.	,	zleva doprava	op. čárka

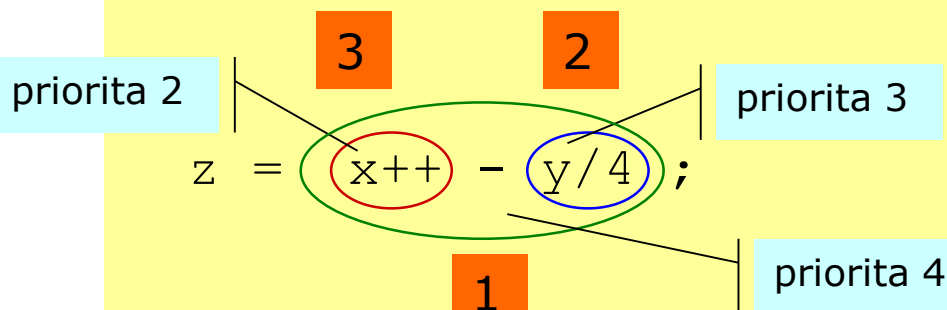


# Priorita operátorů

- Čím nižší číslo skupiny, tím vyšší priorita

*Příklad:*

```
#include <stdio.h>
int main(void)
{
    signed int z, x=3, y=9;
```



```
printf("%d \n", z);
return 0;
}
```



# Principy vyšších programovacích jazyků

---





# Kontrolní otázky

---

1. Uvedte druhy operátorů.
2. K čemu slouží operátor *sizeof*?
3. Jak probíhá vyhodnocení výrazů?





# Úkoly k procvičení

---

Zadání úkolů:

□ Zapište v jazyce C:

- podmínky:  $x \in \langle 0, 1 \rangle$ ,  $x \notin (5, 10 \rangle$ ,  $x \in \{2, 5, 8\}$ ,  $x \notin \{1, 5\}$ ,  $x \neq \pm 1$ ,  $c$  je dělitelné 5,  $b$  je sudé,  $x \geq 1 \wedge x$  je liché,  $c$  je dělitelné 2 nebo 3.
- negaci výrazů (bez použití operátoru (!)):  $A > 0$ ,  $A + B \geq C$ ,  $X * Z = Z * A$ ,  $(A > 0) \wedge (B > 0)$ ,  $(A = 0) \vee (B = 0)$ ,  $\neg(A > B)$ ,