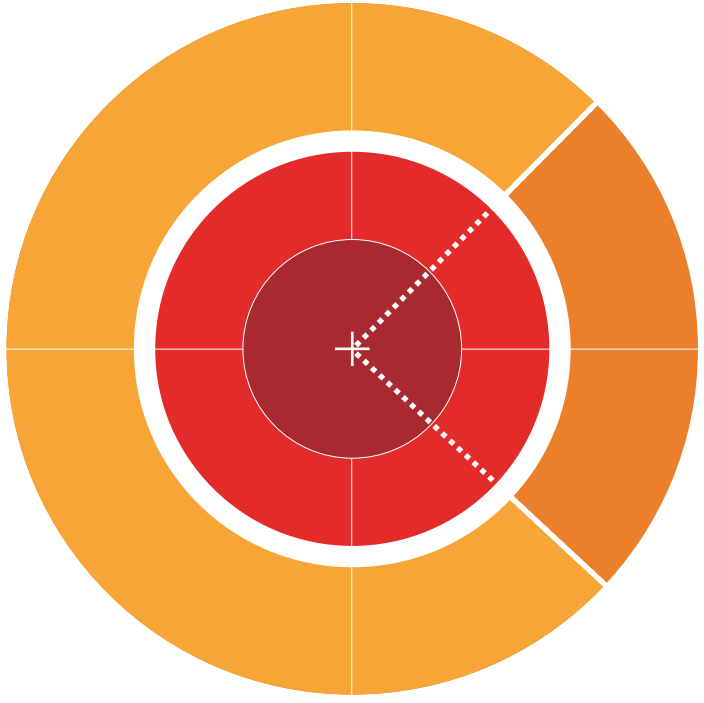


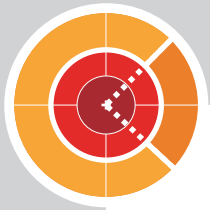
Výrazy -

(i) **l-value (l-hodnota)**:
- výraz, který odkazuje na objekt takovým způsobem, aby se daný objekt mohl "prohlížet" nebo do něj něco uložit; může být typu neúplné pole, ale nesmí být void

(ii) **r-value (r-hodnota)**:
- opak l-hodnoty, pouze na pravé straně

```
a[i + 50];
sizeof(int) * p;

a[2 * n] = b++ & 0xFF;
```

l-hodnota **r-hodnota**



Výrazy, které mohou být l-hodnotou

identifikátor
`e[k]`
`(e)`
`e.`*identifikátor*
`e->`*identifikátor*
`*e`

Operátory, které vyžadují l-hodnotové operandy

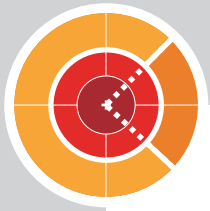
```
&
++ --
= += -= *= /= %= <<= >>= &= ^= |=
```

levý operand musí být l-hodnota
(pravý ne)

Operátory a jejich priorita

Tabulka 7-1: **C operátory v pořadí jejich priority**

Operátor	Operace	Třída	Priorita	Asociativita
jméno, literál	jednoduchý atom	primární	17	není
a[*k*]	indexování	postfix	17	zleva
f(...)	volání funkce	postfix	17	zleva
.	přímý výběr	postfix	17	zleva
->	nepřímý výběr	postfix	17	zleva
++ --	zvýšení, snížení	postfix	16	zleva
++ --	zvýšení, snížení	prefix	15	zprava
sizeof	velikost	unární	15	zprava
~	bitové not	unární	15	zprava
!	logické not	unární	15	zprava
- +	aritmetická negace, plus	unární	15	zprava
&	adresa	unární	15	zprava
*	nepřímý odkaz	unární	15	zprava
(*jméno typu*)	přetypování	unární	14	zprava
* / %	multiplikativní	binární	13	zleva
+ -	aditivní	binární	12	zleva
<< >>	posuv doleva, doprava	binární	11	zleva
< > <= >=	relace	binární	10	zleva
== !=	rovnost/nerovnost	binární	9	zleva
&	bitové and	binární	8	zleva
^	bitové xor	binární	7	zleva
|	bitové or	binární	6	zleva
&&	logické and	binární	5	zleva
||	logické or	binární	4	zleva
? :	podmíněná	ternární	3	zprava
= += -= *= /= %= <<= >>= &= ^= |=	dosazení	binární	2	zprava
,	postupné vyhodnocení	binární	1	zleva

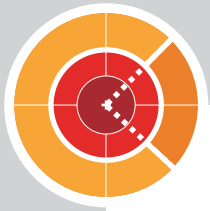


Indexování

```
char buffer[100];
char *bptr = buffer;
int i = 99;

buffer[0] = 'a';
bptr[i - 1] = bptr[0];
i[bptr] = 'a';

bptr = &buffer[6];
bptr[-4] = 'b';
bptr = NULL;
```

nekonvenční indexování (**nepoužívat** - ale kupodivu to funguje)

- index jde vždy od 0
- nejvyšší hodnota indexu je o 1 menší, než rozměr pole uvedený v deklaraci



Indexování vícerozměrných polí

```
#define SIZE 10

double matrix[SIZE][SIZE];
int i;

matrix[0][0] = 1.0;

for (i = 0; i < SIZE * SIZE; i++)
   ((double *) matrix)[i] = 0.0;
```

přetypováním na jednorozměrné pole snadno vynulujeme
(**opatrně** - tento postup může vést k chybám z nepozornosti)

Výběr složky struktury

```
struct Point { double x, y; } point,
   *ppoint;

point.x = 1.5;
point.y = 3.7;

ppoint = &point;

ppoint->x = 0.0;
(*ppoint).x = 0.0;
```

} ekvivalentní

závorky jsou nutné
(operátor výběr složky má vyšší prioritu než operátor dereference *)

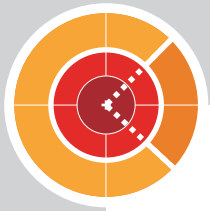
Volání funkce

- jménem a seznamem p-metrů oddělených čárkami v závorkách - nebo `()` nemá-li fce p-metry
- všechny p-metry se předávají **hodnotou**, tj. nelze je operacemi uvnitř fce změnit

```
double square(double y) {
    return y * y;
}

void swap(int *px, int *py) {
    int t = *px;
    *px = *py;
    *py = t;
}
```

návratový typ

potřebujeme-li hodnoty parametrů změnit, musí-me předat ukazatele na ně



Implicitní předání parametrů odkazem

- je-li p-metrem funkce **pole**, předává se samozřejmě odkazem, tj. ukazatel na pole => změníme-li hodnotu prvku uvnitř těla funkce, změní se natrvalo

```
void archg(double data[]) {
    data[0] = 0.0;
}

void sidefx1() {
}

void sidefx2(void) {
}
```

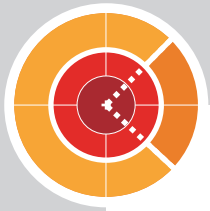
Prefixový/postfixový operátor zvýšení/snížení hodnoty

```
int i, j;

...

i++;          i = i + 1;
j--;          j = j - 1;
i = j++;      i = j; j = j + 1;
i = j--;      i = j; j = j - 1;

++i;          i = i + 1;
--j;          j = j - 1;
i = ++j;      j = j + 1; i = j;
i = --j;      j = j - 1; i = j;
```

postfix

prefix

- používat rozumně

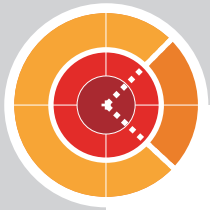


Přetypování neboli explicitní typová konverze

```
int i;
char a[] = "abcd";

i = (int) *a;
```

jaký typ z toho chceme udělat

```
#include <stdio.h>

const char *entry = "\060\000\000\000Novak\000";
struct S { int age; char name[6]; } *ps;

int main() {
   ps = (struct S *) entry;
   printf("Name: %s, aged %d", ps->name, ps->age);
   return 0;
}
```

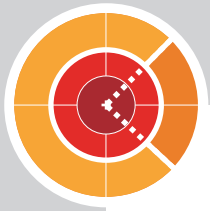


Operátor sizeof - zjištění velikosti objektu

```
struct Person { char name[32]; int age; }
   employee;
int i;

printf("%d\n", sizeof(employee));
printf("%d\n", sizeof(i));
printf("%d\n", sizeof(int));
```

```
36
4
4
```

- používá se zejména při práci s pamětí (alokace, uvolňování)
- při vývoji multiplatformních programů pro zjištění velikosti (a tedy rozsahu) základních datových typů



Unární operátory

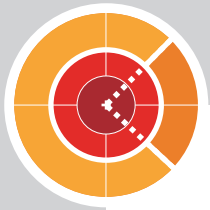
unární minus a plus	`+, -`	
logická negace	`!`	
bitová negace	`~`	
adresový (referenční) operátor	`&`	**vzájemně inverzní** (`*&x == x`)
dereferenční operátor	`*`	

```
int i = 0, j = -5;
int *ip;

i = -j; i = ~j;
ip = &i; j = *ip;

if (!i) { ... }
```

podmínka je splněna, je-li **i rovno 0**. (`i == 0` zn. False, `i != 0` zn. True)

Binární operátory

Operátor	Operandy	Výsledek
* /	aritmetické	aritmetický
%	celočíselné	celočíselný
+	aritmetické	aritmetický
	ukazatel a celočíselný	ukazatel
-	aritmetické	aritmetický
	ukazatel - celočíselný	ukazatel
	ukazatel - ukazatel	celočíselný
<< >>	celočíselné	celočíselný
< <= >= >	aritmetické nebo ukazatele	0 nebo 1
== !=	aritmetické nebo ukazatele	0 nebo 1
&	celočíselné	celočíselný
^	celočíselné	celočíselný
|	celočíselné	celočíselný

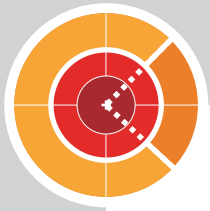


Operátory posunu bitů

```
int i = 0x8000  ..00 1000 0000 0000 0000
int j;

j = i << 2;     ..10 0000 0000 0000 0000
j = j >> 3;     ..00 0100 0000 0000 0000
j >>= 41;       ..00 0000 0000 0010 0000
```

```
j = j >> 41;
```

- je-li počet bitových pozic, o které se má operand posunout, **větší** než počet bitů daného datového typu, bere se hodnota daná celočíselným zbytkem po vydělení počtu pozic počtem bitů (v příkladu `41 % 32 == 9`, tj. posun o 9 vpravo)
- je-li počet bitových pozic **záporný**, závisí výsledek na překladači a je nepředvídatelný



Bitové operátory

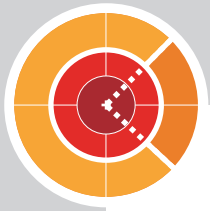
a	b	a & b	a ^ b	a | b
		AND	XOR	OR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	1

- **pozor na záměnu** s logickými operátory `&&` a `||`:

```
int i = 2;
int j = 4;

if (i & j) {...}
if (i && j) {...}
```

tato podmínka **neproběhne** (`a & b == 0`, tj. nepravda)

tato podmínka naopak **proběhne**, protože a i b jsou současně nenulové (`a && b == 1`, tj. pravda)

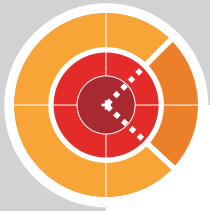


Užití bitových operátorů

```
int onebits(int i) {
    int j, count = 0;

    for (j = 0; j < sizeof(i) * 8; j++)
        if (i >> j & 1) count++;

    return count;
}
```

test, je-li 0. bit (LSB)
nastaven na 1

- funkce počítá, kolik je v parametru typu **`int`** jedničkových bitů
- operátorem **`sizeof`** je zajištěna přenositelnost mezi platformami s různým počtem bitů u typu **`int`**



Relační operátory - častý zdroj problémů

- **pozor** na **relace se smíšenými typy**

```
if (-1 < (unsigned) 0) {...}
```

tato podmínka **není!** v důsledku typových konverzí **splněna**

- operátory == a != mají stejnou prioritu (menší než <, <=, >, >=) a jsou asociativní zleva, tj. **pozor**

```
int x = 7, y = 7;

if (x == y == 7) {...}
```

tato podmínka **není! splněna**, protože v důsledku levé asociativity se výraz vyhodnocuje jako

```
(x == y) == 7
```

1

- nelze porovnávat struktury a uniony (složitý postup)

Logické výrazy a operátory

- v ANSI C neexistuje logický typ (Boolean), pracuje se s celočíselnými typy: **0 - nepravda, vše ostatní pravda**

```
if (a && b) {...}
if (a || b) {...}
```

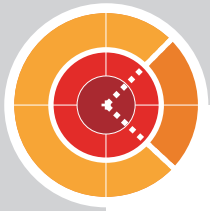
logické A (AND)

logické NEBO (OR)

- výsledek operací **`&&`** a **`||`** je vždy **`int`**
- vždy se provádí **neúplné vyhodnocování**

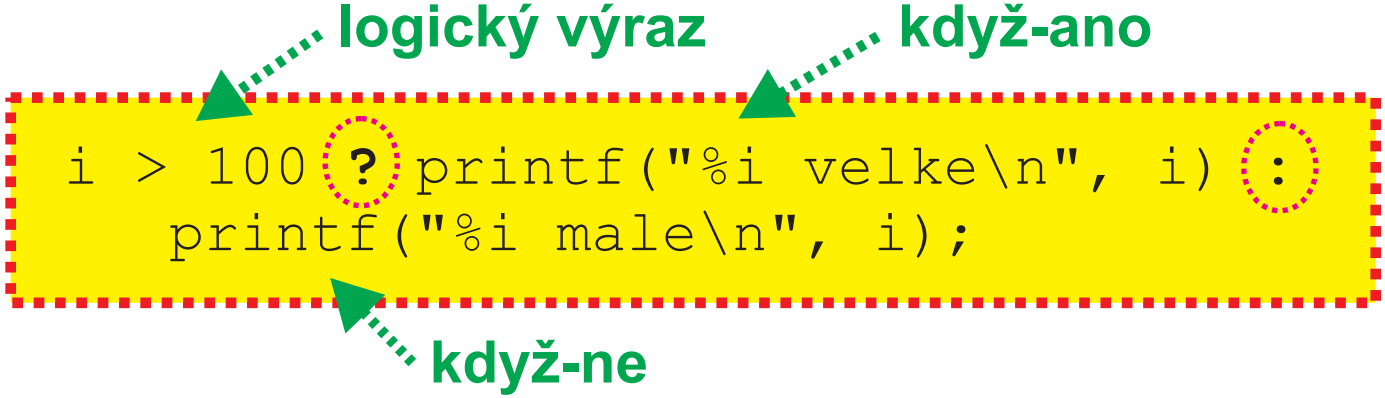
a	b	a&&b	Vyhodnocuje se b?	a||b	Vyhodnocuje se b?
1	0	0	ano	1	ne
0	34.5	0	ne	1	ano
1	"Ahoj\n"	1	ano	1	ne
'\0'	0	0	ne	0	ano
&x	y=2	1	ano	1	ne

- **pozor na záměnu s bitovými operátory**

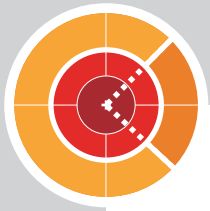


Podmíněný výraz

- ternární operátor, asociativní **zprava**
- výsledná hodnota je dána druhým operandem tehdy, je-li první operand nenulový, jinak je dána třetím operandem




```
int signum(int x) {
   return x > 0 ? 1 : x < 0 ? -1 : 0;
}
```



Dosazovací výraz a dosazovací operátory

- binární operátory, asociativní **zprava**, mohou být **složené**
- při **skládání** **pozor**: snižuje čitelnost, matoucí

Dosazovací operátor	Levý operand	Pravý operand
*= /=	aritmetický	aritmetický
%=	celočíselný	celočíselný
+= -=	aritmetický	aritmetický
+= -=	ukazatel	celočíselný
<<= >>=	celočíselný	celočíselný
&=	celočíselný	celočíselný
^=	celočíselný	celočíselný
|=	celočíselný	celočíselný

```
x *= y = z;
a = b = c + 7;
```

```
x *= (y = z)
```

NE `(x *= y) = z`

```
a = (b = c + 7)
```

Čárkovaný výraz - postupné provedení výrazů

- binární operátor, asociativní **zleva** (nemá příliš smysl), operandy jsou odděleny čárkou
- hodnota čárkovaného výrazu je dána **pravým operandem**, pokud nějakou hodnotu vrací i levý operand, je potlačena

```
r = (a, b, c++, d > 5);
```

```
a;
b;
c++;
r = (d > 5);
```

závorky jsou **nutné**

- jediné smysluplné použití čárkovaného výrazu je v cyklu **for**, jinde (např. v p-metrech fce) může být přímo v rozporu se syntaxí jazyka C

```
for (x = 0, y = N;
   x < N && y > 0; x++, y--) {...}
```