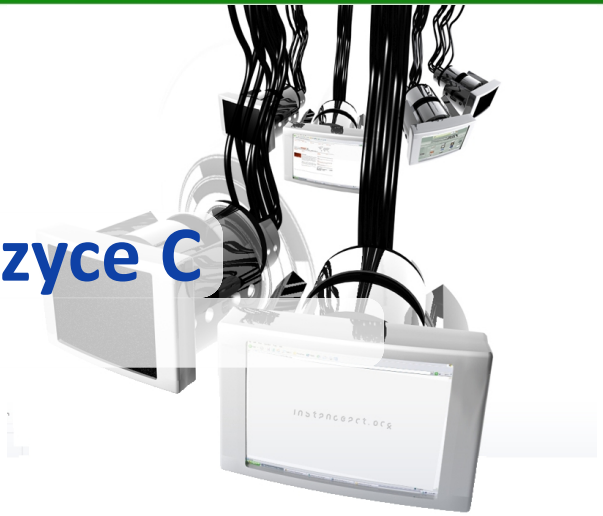

Programování v jazyce C

4 Výrazy a operátory

- Koncepce L-value a R-value
- Operátory a jejich priorita
- Indexování a výběr složky
- Explicitní typová konverze
- Podmíněný výraz

Výrazy
Pojmy L-value a R-value

V jazyce C se ve výrazech rozlišují dvě abstraktní entity:

L-value *(L-hodnota)* – výraz, který odkazuje na nějaký objekt, který fyzicky existuje v paměti počítače, lze do něj např. uložit nějakou hodnotu (proměnná, pole, prvek pole, ...).

R-value *(P-hodnota)* – opak L-value, obvykle na pravé straně např. přiřazovacího výrazu, má hodnotu, ale nemusí nutně existovat v paměti (např. konstanta, výsledek operace).

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

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

Výrazy
Výrazy, které mohou být L-value

- <identifikátor> – proměnná
- `e[k]` – prvek pole
- `(e)` – uzávorkovaný výraz (za urč. okolností)
- `e.`<identifikátor> – složka struktury nebo unionu
- `e->`<identifikátor> – složka struktury odkázané pointerem
- `*e` – dereference pointeru

```
x = 5;
a[x] = 10;
(*(s->p)) = 0;
s.w = 200;
s->p = &x;
*q = 1;
```

Operátory
Tabulka priorit operátorů v jazyce C

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

Operátory ve výrazech
Operátory, které vyžadují L-value operandy

- referenční operátor (získání adresy) – **&**
- operátory inkrementace a dekrementace – **++**, **--**
- přiřazovací a dosazovací operátory –
  =, **+=**, **-=**, ***=**, **/=**, **%=**, **<<=**, **>>=**, **&=**, **^=**, **|=**

Levý operand musí být L-value (pravý nikoliv)!

```
a <<= 2;
p &= 0xFF;
b += a++;
```

Dosazovací operátor upravuje hodnotu objektu ⇒ levý operand musí být instancí nějakého objektu v paměti, aby bylo výsledek kam uložit.

Operátor přetypování

Tzv. explicitní typová konverze

```
unsigned int i, j;
char a[] = "ABCD";

i = (unsigned int) *a;
j = *((unsigned int *) a);
```

identifikátor typu, který má překladač užít při práci s *a

i == 0x41

j == 0x44434241

```
#include <stdio.h>

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

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

Operátor indexování
Indexování polí

- Index pole vždy od 0, nejvyšší hodnota indexu je o 1 menší, než rozměr pole uvedený v deklaraci.
- Překladač dodržení rozsahu (většinou) **nekontroluje**.

```
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;
```

alternativní způsob indexování (**nepoužívat!** – ale kupodivu to funguje)

Operátor indexování
Indexování vícerozměrných polí

- Pole je v paměti uloženo lineárně, tj. vícerozměrný index se musí přepočítat na tzv. ***offset*** = vzdálenost prvku od počátku (***báze***) pole. To zajišťuje ***mapovací funkce***...

```
#define SIZE 10

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

for (i = 0; i < SIZE; i++)
  for (j = 0; j < SIZE; j++)
    matrix[i][j] = 0.0;

matrix[0][0] = 1.0;
```

báze
bázový typ
offset

matrix +
+ (i * SIZE + j) * sizeof(double)

Operátor indexování
Indexování vícerozměrných polí – urychlení

- V některých případech (např. nulování pole) lze průchod vícerozměrným polem urychlit přetypováním na jednorozměrné pole ⇒ zjednodušení výpočtu mapovací funkce.

- **POZOR:** Tato technika může vést k chybám z nepozornosti...

```
#define SIZE 10

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

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

Operátory výběru složky struktury
Přímý a nepřímý výběr

```
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;
```

přímý výběr

nepřímý výběr

ekvivalentní

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

- Smyslem operátoru nepřímého výběru je usnadnit zápis výběru složky struktury, na kterou ukazuje ukazatel (běžný stav v C).

Operátor `sizeof`
Zjištění velikosti objektu

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

printf("%d\n", sizeof(employee));   36
printf("%d\n", sizeof(i));           4
printf("%d\n", sizeof(int));         4
```

- Možno předat buď typ nebo instanci typu (objekt), vrací velikost objektu v **bytech**.
- **POZOR: V případě polí záleží na způsobu vzniku** (statická vs dynamická alokace pole)**!**
- Používá se zejména při práci s pamětí (alokace)
- Při vývoji multiplatformních programů pro zjištění velikosti (a tedy rozsahu) základních datových typů, zejm. **`int`**.

Operátory inkrementace a dekrementace
Prefixové/postfixové

```
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**, nadužití vede ke snížení čitelnosti a někdy také k dalším problémům (nejednoznačné určení, kdy má být operace provedena).

Unární operátory
tj. operátory s jediným operandem

unární plus a mínus	`+, -`
logická negace	`!`
bitová negace	`~`
adresový (referenční) operátor	`&`
dereferenční operátor	`*`

} **vzájemně inverzní** (`*&x` ≡ `x`)

```
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
tj. operátory se dvěma operandy

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
SHL (*Shift Left*) a SHR (*Shift Right*)

```
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 obecně **nepředvídatelný**.

Bitové operátory
AND, OR a XOR

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)

Bitové operátory
Příklad použití

```
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 (technika maskování operátorem **`&`** – AND).
- Operátorem **`sizeof`** je zajištěna přenositelnost mezi platformami s různým počtem bitů u datového typu **`int`**.

Relační operátory
==, !=, <, >, <=, >=

- **POZOR: U relací se smíšenými typy dochází k implicitní typové konverzi** – to může způsobit neočekávané chování:

```
if (-1 < (unsigned) 0) { … }
```

Tato podmínka **není!** v důsledku typových konverzí **splněna**.

- Operátory **==** a **!=** mají stejnou prioritu (nižší 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 operátory a výrazy
za nepřítomnosti logického datového typu

- V ANSI C **neexistuje** logický datový typ (bool, boolean)
  ⇒ pracuje se s celočíselnými typy: **0** ≡ **nepravda**, **jakákoliv jiná hodnota** ≡ **pravda**.

```
if (a && b) { … }
```
◄····· logické A (AND)

```
if (a || b) { … }
```
◄····· logické NEBO (OR)

- Výsledek operací **`&&`** a **`||`** je vždy typu **`int`**.
- Vždy se provádí **neúplné vyhodnocování** logických výrazů.

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 <podm> ? <pos> : <neg>
Ternární rozhodovací operátor

- Ternární operátor (3 operandy), asociativní **zprava.**
- Výsledná hodnota je dána druhým operandem tehdy, je-li první operand nenulový, jinak je dána třetím operandem.

```
i > 100 ? printf("%i velké\n", i) :
    printf("%i malé\n", i);
```

logický výraz
(podmínka)

je-li pravdivý

není-li pravdivý

- Rozhodovací operátory mohou být vnořené do sebe (to ale významně snižuje čitelnost kódu).

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

Přiřazovací výraz a přiřazovací operátory
V některých zdrojích též zvané dosazovací

- 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;
i &= a >= 0 ? b : c;
```

```
x *= (y = z)
NE (x *= y) = z
a = (b = c + 7)
```

Čárkovaný výraz
Postupné vykonání (tj. vrácení) operandů

- Binární operátor zavedený kvůli usnadnění zápisu cyklu **`for`**, asociativní **zleva** (nemá moc smysl), zapisuje se čárkou.
- Hodnota čárkovaného výrazu je dána **pravým operandem**, pokud nějakou hodnotu vrací i levý operand, je přepsána.

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

závorky jsou **nutné!**

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

- Jediné smysluplné použití čárkovaného výrazu je v cyklu **for**, jinde (např. v parametrech funkce) může být přímo v rozporu se syntaxí jazyka C.

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

Výraz volání funkce
a získání adresy funkce

```
wait_for_keypress();
c = getch();
v = compute(a, b, get_mode("input.dat"));
```

- Funkce se volá svým **jménem** a seznamem parametrů v závorkách – nemá-li parametry, je seznam prázdný, tj. … **()** .
- Parametry se funkcím předávají **výhradně hodnotou**, tj. případné modifikace hodnot parametrů uvnitř funkce se neprojeví navenek, neboť funkce pracuje s lokálními kopiemi parametrů (uloženými v zásobníku).
- Návratovou hodnotu lze ignorovat. Volání může být vnořené.

- Je-li třeba získat adresu funkce, uvede se pouze její jméno:

```
p = wait_for_keypress;
```

Zde může být referenční operátor **&**.