Definice proměnných

- deklarace = specifikace parametrů a chování identifikátorů
- definice = deklarace s rezervací paměti kompilátorem pro daný identifikátor
- ilustrace:

	definice	deklarace
proměnná	`int a;`	`extern int a;`

funkce	`int mocnina(int x)`
	`{`
	`  return x * x;`
	`}`
		`int mocnina(int x);`

- definice

  proměnné: `int a, b; double x;`

- definice s inicializací

```
long x = 10, aaa, y = -33;
```

- pouze na začátku bloku

```
int main(int argc, char* argv[])
{
	int a, b;
	double u;
	printf("ahoj");
	long x = 10, z, y = -33;
	if (x > 5)
	{
		unsigned short int pomoc;
		// dalsi cinnost
	}
}
```

OK

error

OK

Operátory

- dělení:
  - aritmetické, logické, relační, bitové, ostatní
  - unární, binární, ternární
- Priorita (Operator Precedence) = přednost vyhodnocení ve výrazech
- Asociativita (Operator Associativity) = směr vyhodnocení

(priorita od nejvyšší k nejnižší)

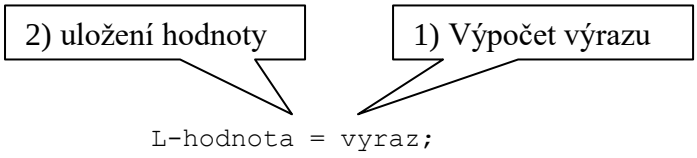
Operátor	Popis	Asoc.
`()`	volání funkce, priorita vyhodnocení	
`[]`	prvky pole	
`jméno.člen`	přístup k prvkům struktury	
`pointer->člen`	přístup k prvkům struktury	
`++ --`	in(de)krementační – postfix	
`++ --`	in(de)krementační – prefix	←
`+ -`	unární +, –	
`! ~`	Logické, bitové NOT	
`(typ) výraz`	přetypování	
`*`	Pointer – dereference	
`&`	Pointer – reference (adresový op.)	
`sizeof (výraz)`	velikost výrazu v B	
`* / %`	binární *, /, % (násobení, dělení, dělení modulo)	
`+ -`	binární +, – (součet, rozdíl)	
`<< >>`	bitový posun vlevo, vpravo	
`< <=`	menší, menší nebo rovno	
`> >=`	větší, větší nebo rovno	
`== !=`	rovno, nerovno	
`&`	bitové AND	
`^`	bitové XOR	
`|`	bitové OR	
`&&`	logické AND	
`||`	logické OR	
`?:`	ternární (podmíněný) op.	←
`=`	přiřazení	←
`+= -=`	složené přiřazení	
`*= /=`		
`%= &=`		
`^= |=`		
`<<= >>=`		
`,`	čárka (oddělení výrazů)	

Poznámky

- nevyznačená asociativita →
- typ `/` dle operandů – alespoň jeden reálný → reálné
- závorkovat!!!

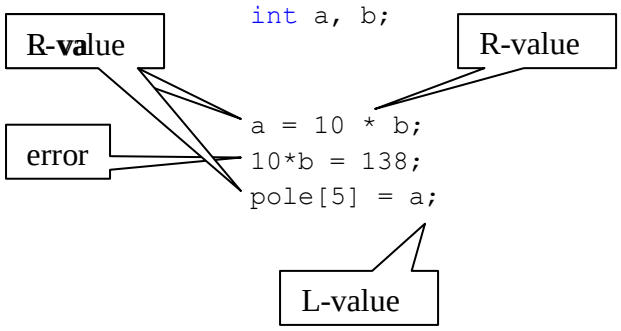
Přiřazení

- (Assignment)



L-hodnota

- L-hodnota
  - (L-Value), L = left, locator
  - = výraz s výsledkem odpovídajícím adrese v paměti
  - vpravo i vlevo od =
- R–hodnota
  - (R-Value)
  - vše, co není L-value
  - pouze vpravo od =
- příklady:



Poznámky

- může být součástí výrazů
  - příklad I

```
double a, b, c = 10;
a = 10*(b = c);   // b = 10, a = 100
```

  - příklad II – čtení ze souboru

```
while ((znak = getc(soubor)) != EOF)
{
// ...
```

- vícenásobné přiřazení

```
a = b = 10 * x;   // a = (b = (10 * x))
```

- složené přiřazení

```
a += 10 * x;      // a = a + (10 * x)
```

Přetypování

- (casting)
- = přepočet formátu čísla při operacích s nestejnými datovými typy

```
int a;
double b=2;
a = b;
```

implicitní

- = automatická konverze z nižších typů na vyšší, např.:

```
char → int → double
```

- Příklad:

```
int a;
double b=2;
b = a;      // OK
```

explicitní

- = ruční konverze v vyšších typů na nižší

```
(novy_typ)vyraz
```

- příklad:

```
int a;
float b=2.87;
a = (int)b; // OK, a = 2, ztráta informace
```

Poznámky

- vyšší priorita než binární operátory

```
(int)a + b        // pretypovani a
```

- přetypovat lze cokoli

```
x = k * (double)(a + b)
x = (double)30;
```

- konverze `signed` ↔ `unsigned` – bitově totéž, ale různá interpretace čísla (2D vs. bin)

```
unsigned char u = 255;
signed char i;
i = u;            // i = -1
```

Logické výrazy, proměnné

- logická hodnota reprezentována `int`:
  - 0 „false“
  - nenulová hodnota „true“

- Příklad:

```
int damSiPivo = 0;      // = zatim si nedam
damSiPivo = 1;          // = pivo si dam
```

- logické výrazy
  - s operátory `&&`, `||`, `!`
  - výsledek „true“ → kompilátor vrací `1`
  - výsledek „false“ → kompilátor vrací `0`
- reálné typy → `0.0` = „false“

```
double a=1.256, b=0.0, c;
c = a && b;       // c = 0.0
c = a || 45.6;    // c = 1.0
c = !a;           // b = 0.0
```

  - logické operátory a reálné typy přímo nepoužívat!!!

Bitové operace

- Bitové operátory definovány pouze pro celočíselné typy

AND, OR, XOR

- Příklad

```
unsigned char A = 189;  //     1011 1101
unsigned char B = 90;   //     0101 1010
unsigned char C,D,E;
C = A | B;              // C = 1111 1111 = 255
D = A & B;              // D = 0001 1000 = 24
E = A ^ B;              // E = 1110 0111 = 231
```

Posuvy

- posun vlevo: `operand << n`
  - bitově posune `operand` o `n` pozic vlevo, zprava jsou doplňovány 0
  - příklad:

```
unsigned char A = 5, X;
X = A << 2; // X = 20
//  5 = 00000101
// 20 = 00010100
```

  - posun u 1 pozici vlevo = násobení operandu 2
- posun vpravo: `operand >> n`
  - bitově posune `operand` o `n` pozic vpravo, zleva jsou doplňovány 0
  - příklad:

```
unsigned char A = 5, X;
X = A >> 1; // X = 2
// 5 = 00000101
// 2 = 00000010
```

Negace

- ~ – převrátí hodnoty všech bitů v operandu

```
C = ~0x0F;  // C = 0xF0
```

Poznámky

- Nutné znát vnitřní reprezentaci datových typů!!!
- pouze pro celá čísla
- logické vs. bitové operátory

```
unsigned char A = 189;  //     1011 1101
unsigned char B = 90;   //     0101 1010
unsigned char C;
C = A | B;              // C = 1111 1111 = 255
C = A || B;             // C = 1
C = A & B;              // C = 0001 1000 = 24
C = A && B;             // C = 1
C = ~B;                 // C = 1010 0101 = 165
C = !B;                 // C = 0
```

Příklad

- Napište kód, který uloží (na 32 b platformě) čtyři čísla `char` do jednoho čísla `int`

```
char b1, b2, b3, b4;
unsigned int cil;
cil = b1;
cil |= (unsigned int)b2 << 8;
cil |= (unsigned int)b3 << 16;
cil |= (unsigned int)b4 << 24;
```

Ternární operátor

- podmíněný operátor

```
BV ? vyraz1 : vyraz2
```

- sémantika

```
if (BV) {
      vyraz1;
} else {
      vyraz2;
}
```

- je to výraz, nikoli příkaz!!!
- Př.: převod čísla na absolutní hodnotu

```
int prom = 10;
prom = (prom >= 0) ? prom : -prom;
```

- využití
  - jako příkaz

```
int a = 5, b = 10, c = 1;
c = (a > b) ? a : b;
(c == 1) ? a++ : b++;
```

  - jako výraz

```
printf("%d", (a == b) ? a : b);
```

Uživatelské datové typy I

- syntaxe: `typedef StaryTyp NovyTyp;`
- náhrada jmen vestavěných (základních) typů

```
typedef unsigned int uint;
uint Prom = 10;

void Funkce(uint Arg1, double Arg2)
{
      // ...
```