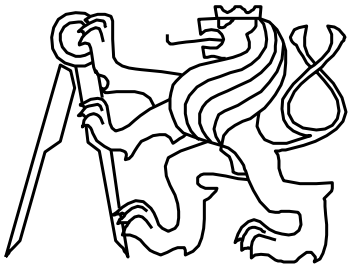
*Příprava studijního programu Informatika je podporována projektem financovaným z Evropského sociálního fondu a rozpočtu hlavního města Prahy.*
*Praha & EU: Investujeme do vaší budoucnosti*

# Racionální čísla, operátory, výrazy, knihovní funkce



**BI-PA1 Programování a algoritmizace 1, ZS 2012-2013**
**Katedra teoretické informatiky**
© Miroslav Balík
Fakulta informačních technologií
České vysoké učení technické

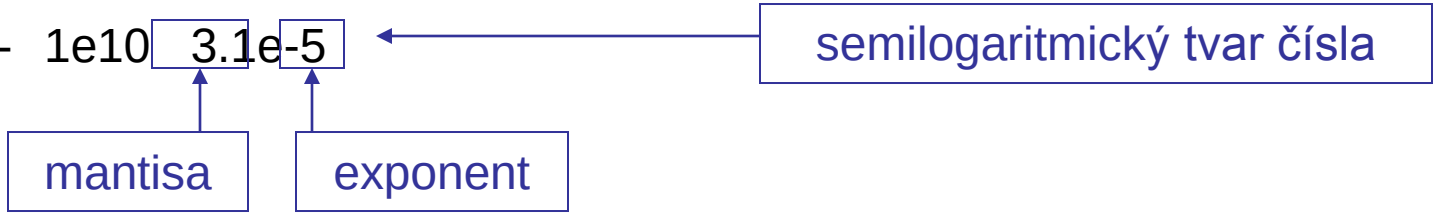
# Racionální typy

- Pro práci s racionálními čísly slouží typy
  - *float*
  - *double*
  - *long double* (nebudeme používat)
- Umožňují zobrazit aproximace racionálních čísel v určitém rozsahu a s určitou přesností
- Rozsah hodnot ani přesnost zobrazení nejsou dány normou, ale implementací jazyka; obvykle platí:

| typ | B | přesnost | rozsah hodnot |
|---|---|---|---|
| float | 4 | 7 dek. číslic | $\pm 3{,}4 \cdot 10^{-38}$ až $\pm 3{,}4 \cdot 10^{+38}$ |
| double | 8 | 15 dek. číslic | $\pm 1{,}7 \cdot 10^{-308}$ až $\pm 1{,}7 \cdot 10^{+308}$ |
| long double | 10 | 19 dek. číslic | $\pm 3{,}4 \cdot 10^{-4932}$ až $\pm 1{,}1 \cdot 10^{+4932}$ |

# Zápis racionálních čísel

- V programu i ve vstupních datech zapisujeme racionální čísla s desetinou tečkou nebo s exponentem
  - 45.31 0.25 .25 10.
  - 1e10 3.1e-5




Jsou typu *double*

- Má-li být typ čísla *float* nebo *long double*, musíme k číslu přidat *F* nebo *L*
  - 25.1F 5e-15L
- V paměti jsou racionální čísla zobrazena jako čísla v *pohyblivé řádové čárce*
- Princip tohoto zobrazení
  - při daném základu *z* (např. 16) je číslo *x* zobrazeno dvojicí *m, e,* kde
    - *m* je normalizovaná mantisa, pro kterou $z^{-1} \leq |m| < 1$ nebo $m=0$
    - *e* je exponent
    - $x = m \cdot z^e$
- Podrobněji v předmětu ...

# Výpis racionálních čísel

- Pro dekadický výpis racionálních čísel funkcí *printf* slouží konverze:
  - *%f* s desetinnou tečkou (bez exponentu)
  - *%e* v semilogaritmickém tvaru (s exponentem)
  - *%g* podle velikosti čísla buď s, nebo bez exponentu

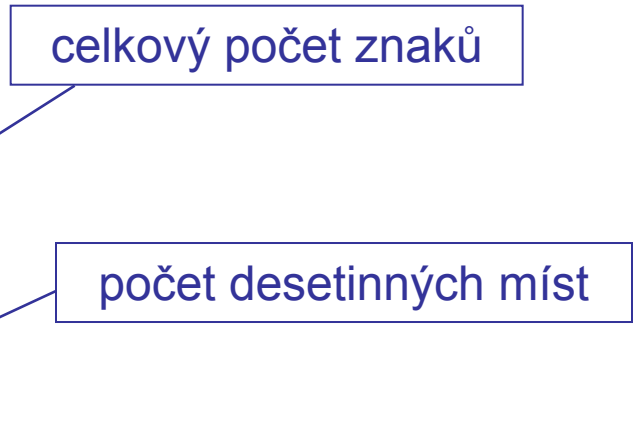
```
/* prog3-1a.c */
#include <stdio.h>
int main(void) {
    float x, y;
    x = 41.25;
    y = 12.1e10;
    printf("x = %f\n", x); printf("y = %e\n", y);
    printf("x = %g\n", x); printf("y = %g\n", y);
    return 0;
}
```

```
x = 41.250000
y = 1.210000e+11
x = 41.2500
y = 1.21000e+11
```

# Výpis racionálních čísel

- Lze zadat celkový počet vypsaných znaků a/nebo počet desetinných míst

```
/* prog3-1b.c */
#include <stdio.h>
int main(void) {
float x, y;
x = 41.25;
y = 12.1e10;
printf("x = %20f\n", x); printf("y = %20e\n", y);
printf("x = %20.3f\n", x);printf("y = %20.3e\n", y);
printf("x = %.3f\n", x); printf("y = %.3e\n", y);
return 0;
}
```




```
x =            41.250000
y =      1.210000e+11
x =               41.250
y =           1.210e+11
x = 41.250
y = 1.210e+11
```

# Vstup racionálních čísel

- Pomocí funkce *scanf* a jakékoliv koverze *%f, %e* a *%g*
- Pozor: přečtené číslo se správně uloží pouze do proměnné typu *float*!!!

```
/* prog3-1c.c */
#include <stdio.h>
#include <stdlib.h>
int main(void) {
float x, y;
printf("zadej dve racionalni cisla: ");
scanf("%f%e", &x, &y);
printf("x = %f\n", x);
printf("y = %e\n", y);
printf("x = %g\n", x);
printf("y = %g\n", y);
return 0;
}
```

# Konverze při přiřazení

- Všechny číselné typy (celočíselné a racionální) jsou *kompatibilní vzhledem k přiřazení*
- Znamená to, že v přiřazovacím příkazu

  $x = v$;

  může proměnná $x$ a hodnota $v$ být libovolného číselného typu. Přiřazovaná hodnota se vždy konvertuje na typ proměnné.
- Následující tabulka udává 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 |

- Příklady: prog3-2a.c, prog3-2b.c

# Výrazy

- Výraz předepisuje výpočet hodnoty určitého typu
- Výraz může obsahovat:
  - proměnné
  - konstanty
  - volání funkcí
  - binární operátory
  - unární operátory
  - závorky
- Pořadí operací předepsaných výrazem je dáno:
  - prioritou operátorů
  - asociativitou operátorů
- Příklad:

| výraz | pořadí operací | zdůvodnění |
|---|---|---|
| x + y * z | x + (y * z) | *** má vyšší prioritu než** |
| x + y + z | (x + y) + z | **+ je asociativní zleva** |

# Aritmetické operátory

- Podle priority (od nejvyšší po nejnižší):
  - unární + a –
    - např. +*x*, -*x*
  - binární *, /, % (násobení, dělení, zbytek po dělení)
    - např. *x/y, x*y, x%z*
  - binární +, - (sčítání, odčítání)
    - např. *x+y, x-y*
- Dělení (/) pro celočíselné operandy je celočíselné dělení (celá část podílu)
- Zbytek po dělení (%) je definován pouze pro celočíselné typy takto:

  $x = (x / y) * y + x \% y$

# Aritmetické operátory – příklad

```
/* prog3-3a.c */
#include <stdio.h>
#include <stdlib.h>
int main(void) { int i,j;
    printf("zadejte dve cela cisla (i,j): ");
    scanf("%d%d", &i, &j);
    printf("-i = %d\n", -i); printf("+j = %d\n", +j);
    printf("i+j = %d\n", i+j);
    printf("i-j = %d\n", i-j);
    printf("i*j = %d\n", i*j);
    printf("i/j = %d\n", i/j);
    printf("i%%j = %d\n", i%j);
    return 0;
}
```

chceme-li v řetězci *formát* zadat znak %, musíme napsat dva znaky %

# Aritmetické operátory – příklad II

```
/* prog3-3b.c */
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    int i = 13, j = 4;
    printf(" i j i/j i%%j\n");
    printf("%4d%4d%6d%6d\n", i, j, i/j, i%j);
    printf("%4d%4d%6d%6d\n", -i, j, -i/j, -i%j);
    printf("%4d%4d%6d%6d\n", i, -j, i/-j, i%-j);
    printf("%4d%4d%6d%6d\n", -i, -j, -i/-j, -i%-j);
    return 0;
}
```

```
  i    j    i/j   i%j
 13    4     3     1
-13    4    -3    -1
 13   -4    -3     1
-13   -4     3    -1
```

# Aritmetické operace s operandy různých typů

- Operandy aritmetických operátorů mohou být různých číselných typů
- Před provedením operace se provedou tyto konverze (běžné aritmetické konverze – usual aritmetic conversions):
  - je-li jeden operand typu *long double,* druhý se převede na *long double* a operace se provede v typu *long double*
  - jinak, je-li jeden operand typu *double,* druhý se převede na *double* a operace se provede v typu *double*
  - jinak, je-li jeden operand typu *float,* druhý se převede na *float* a operace se provede v typu *float*
  - jinak, operand celočíselného typu, který je „menší“ než *int* resp. *unsigned int*, se převede na *int* resp. *unsigned int* (celočíselné roztažení, integer promotion) a operace se provede celočíselně ve „větším“ z typů operandů (přesná pravidla najdeme v normě jazyka)
- Příklad: prog3-3c.c

# Relační operátory

- Hodnoty všech číselných typů jsou uspořádané a lze je porovnávat relačními operátory
- Relační operátory ( priorita je menší než priorita aritmetických operátorů ):

  >, <, >=, <= ( větší než, menší než, větší nebo rovno, menší nebo rovno )

  ==, != ( rovná se, nerovná se )
- Výsledek relační operace je typu *int*: 1, když relace označená operátorem platí, 0 v opačném případě
- Před vyhodnocením operace se provedou běžné aritmetické konverze
- Příklad: prog3-3d.c

# Logické operátory

- Operandy číselného typu
- Výsledek typu *int* (0 nebo 1)
- Priorita je menší než priorita relačních operátorů
- Operandy se vyhodnocují zleva, druhý operand se nevyhodnocuje, je-li výsledek dán prvním operandem
- Unární:
  - ! **logická negace**
    - **výsledek 1, má-li operand hodnotu 0, jinak je výsledek 0**
- Binární:
  - && **logický součin (konjunkce)**
    - **výsledek 1, jsou-li oba operandy nenulové, jinak 0**
  - || **logický součet (disjunkce)**
    - **výsledek 1, je-li alespoň jeden operand nenulový, jinak 0**
- Příklady: prog3-4a.c, prog3-4b.c, prog3-4c.c, prog3-4d

# Bitové operátory

- Operandy jen celočíselné, provádějí se běžné aritmetické konverze
- Unární:
  - ~ **negace všech bitů**
- Binární:
  - & **logický součin všech bitů**
  - | **logický součet všech bitů**
  - ^ **logické *xor* všech bitů**
  - << **posun bitové reprezentace levého operandu vlevo**
  - >> **posun bitové reprezentace levého operandu vpravo**

  U operací posunu pravý operand udává délku posunu v bitech
- Pozor, plete se & a &&, | a || :

```
int x = 1, y = 2;
x & y == 0
x && y == 1
```

- Příklad: prog3-5a.c

# Inkrementace a dekrementace

Unární operace s vedlejším efektem (změní hodnotu operandu)

- Lze zapsat prefixově nebo postfixově
- Inkrementace: **++**
- Dekrementace: **−−**
- Prefix: ++*X* --*X*

  inkrementuje (dekrementuje) *X*, hodnotou je změněné *X*
- Postfix: *X*++ *X*--

  inkrementuje (dekrementuje) *X*, hodnotou je *X* před změnou
- Příklady:

| před | operace | po |
|---|---|---|
| – x == 1 | y = ++x | x == 2, y == 2 |
| – x == 1 | y = x++ | x == 2, y == 1 |
| – x == 1 | y = --x | x == 0, y == 0 |
| – x == 1 | y = x-- | x == 0, y == 1 |

- Příklad: prog3-6a.c

# Podmíněný výraz

- Výsledek podmíněného výrazu je závislý na hodnotě podmínky
- Syntaxe:

  *Podmínka* ? *Výraz1* : *Výraz2*

  kde *Výraz1* a *Výraz2* jsou výrazy číselných typů
- Je-li *Podmínka* splněna (nenulová hodnota), je výsledkem hodnota *Výrazu1*, jinak je výsledkem hodnota *Výrazu2*
- Vyhodnocuje se podmínka a pak jen příslušný výraz
- Příklad:
  - max = a > b ? a : b;
  - p = r * (x < 0 ? -1 : x == 0 ? 0 : 1);
- Příklad: prog3-7a.c

# Operátor přiřazení, výrazový příkaz

- Operátor přiřazení (=) patří mezi binární operátory, má vedlejší efekt
- Výraz

  *x* = *h*

  se vyhodnotí tak, že hodnota *h* se (po případné konverzi) přiřadí proměnné *x* a výsledkem je nová hodnota proměnné *x*
- Z přiřazovacího výrazu uděláme přiřazovací příkaz tím, že ho zakončíme znakem středník

  *x* = *y*  přiřazovací výraz

  x = y;  přiřazovací příkaz
- Z jakéhokoliv výrazu uděláme příkaz tím, že ho zakončíme středníkem
- Příklad:

  *x*++;  --y;

# Operátor přiřazení

- Přiřazení lze řetězit, je asociativní zprava

```
/* prog3-8a.c */
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    int i;
    float f;
    f = i = 1.98; /* a co teď?*/
    printf("i = %d\n", i);
    printf("f = %f\n", f);
    system("PAUSE");
    return 0;
}
```

- Co program vypíše?

# Složené přiřazení

- Operátory složeného přiřazení (přiřazení spojeno s operací):
  - += -= *= /= %= &= |= ^= <<= >>=
- Význam:

  *X op= Y* je zkratkou za *X = X op (Y)*

```
/* prog3-9a.c */
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    int i = 10,
        j = 5,
        k = 0;
    k += i+j;
    printf("i = %d\n", i);
    printf("j = %d\n", j);
    printf("k = %d\n", k);
    system("PAUSE");
    return 0;
}
```

# Knihovní funkce

- Při numerických výpočtech někdy potřebujeme matematické funkce jako např. *sin*, *cos,* √ *apod.*
- Jejich deklarace jsou v souboru *math.h*
- Většina (záleží na implementaci) je uváděna ve třech variantách:

```
#include <math.h>
double sin(double x); // sin ... double
float sinf(float x); // sinf ... float
long double sinl(long double x); // sinl ... long
```

# Knihovní funkce – přehled základních

```
double sin(double X); //sin X
double cos(double X); //cos X
double tan(double X); //tg X
double asin(double X); //arcsin X
double acos(double X); //arccos X
double atan(double X); //arctg X
double atan2(double X, double Y); // arctg (X/Y)
double sinh(double X); //sinh X
double cosh(double X); //cosh X
double tanh(double X); //tgh X
double exp(double X); //eX
double log(double X); //ln X
double log10(double X); //log10 X
double pow(double X, double Y); //XY
double sqrt(double X); //X1/2
double fabs(double X); //|X| pro celá čísla je abs
```

# Knihovní funkce

- Příklad použití:

```
/* prog3-10a.c */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main(void) {
    float x, y, z;
    printf("zadejte delky odvesen: ");
    scanf("%f%f", &x, &y);
    z = sqrt(x*x+y*y);
    printf("delka prepony je %f\n", z);
    system("PAUSE");
return 0;
}
```

- S dalšími funkcemi se seznámíme na cvičení