RR

Čísla s voliteľnou presnosťou s pohyblivou desatinou čiarkou

Obsah

1 Na úvod

Trieda RR je použitá k reprezentácii čísel s ľubovoľnou presnosťou s pohyblivou rádovou čiarkou. Funkcie v tomto module zaručia veľmi silnú presnosť podmienok, ktoré umožňujú jednoduchšie uvažovať o správaní sa jednotlivých programov používajúcich tieto funkcie.

Výsledky aritmetických operácií sú vždy okolo p bitov, kde p je aktuálna presnosť. Aktuálna presnosť môže byť zmenená použitím $RR :: SetPrecision()$, načítaná je pomocou $RR :: precision()$. Najmenšia presnosť, ktorá môže byť nastavená je 53 bitov. Maximálna presnosť je obmedzená veľkosťou slova procesoru.

Poznámka:

Veľké čísla sú také čísla, ktoré pozostávajú z viacej bitov ako je slovo procesoru (machine word). Napríklad 1024-bitové číslo je považované za veľké číslo. Veľkosť takéhoto čísla býva zvyčajne násobkom veľkosti slova procesoru, aby sa s ním jednoduchšie pracovalo. Vtedy môžeme povedať že veľké číslo pozostáva z niekoľkých slov procesoru.

Procesor nemá inštrukcie na priamu manipuláciu s veľkými číslami, ale poskytuje inštrukcie na manipulovanie so slovami procesoru, ktoré sú súčasťou veľkého čísla. Napríklad procesor x86-32, má veľkost slova 32 bitov. Ak požadujeme napríklad 24 bitovú presnosť, tak celý výpočet bude prebiehať s 32 bytovou presnosťou. Výsledok sa zobrazí v požadovanej presnosti(posledných 8 bytov sa nezobrazí), číže maxímálna presnosť je obmedzená veľkosťou slova stroja(procesoru).

Všetky aritmetické operácie sú zrealizované tak, že výsledný efekt by bol akože vypočítaný presne a potom zaokrúhlený na p bitov. Ak by sa číslo nachádzalo presne v strede medzi dvoma p-bitovými číslami, je použité pravidlo "round to even". To znamená, že vypočítaný výsledok bude mať relatívnu chybu nanajvýš $2^{\{-p\}}$.

Vyššie uvedené zaokrúhlovacie pravidlá je možné aplikovať na všetky aritmetické operácie v tomto module, okrem nasledovných funkcií:

- Transcendentné(abstraktné) funkcie:

  `log, exp, log10, expm1, log1p, pow, sin, cos, ComputePi`

- Funkcie *power*

- Vstup a ascii RR konverzných funkcií pri použití "e"-zápisu.

Pre tieto funkcie je stále zaručená veľmi silná presnosť: vypočítaný výsledok má relatívnu chybu menej než $2^{\{-p+1\}}$(v skutočnosti bližšie k $2^{\{-p\}}$). To znamená, že výsledky boli ako keby vypočítané presne a potom zaokrúhlené na jedno z dvoch susedných p-bitových čísel(ale nie nevyhnutne najbližšie).

Správanie všetkých funkcií v tomto module je nezávislé na celej platforme: "úplne"rovnaké výsledky by ste mali dostať na hocijakej platforme(jedinú výnimku tvorí generátor náhodných čísel).

Pre používanie tejto triedy je potrebné naimportovať knižnicu nasledovným spôsobom:

```
#include <NTL/RR.h>
```

Poznámky:

- Pretože presnosť je variabilná, číslo môže byť vypočítané s vysokou presnosťou p', potom je použité ako vstup k aritmetickým operáciám kde aktuálna presnosť je $p < p'$.

  Vyššie uvedené záruky sa stále aplikujú; obzvlášť tam, kde nie je uskutočnené žiadne zaokrúhlenie pokiaľ sa operácia vykonala.

  Napríklad: Ak x a y sú vypočítané s presnosťou 200 bitov, potom je presnosť nastavená na 100 bitov, potom $x - y$ bude vypočítaná správne na 100 bitov. Ak x a y boli zaokrúhlené pred odčítaním, rozdiel by bol potom s presnosťou len 50 bitov.

- Operátor riadenia a kopírovací konštruktor tvorí presné kópie ich výstupov → oni niesú nikdy zaokrúhlené. Toto je zmena v sémantikách z verzie 2.0 a už skorej v úlohách a kópiách, ktoré zaokrúhľujú ich výstupy. Toto bolo považované za dizajnovú chybu a bola odstránená.

- Ak chcete vynútiť zaokrúhlenie s aktuálnou presnosťou, urobíte to najľahším spôsobom - s prevodným programom *RRdoRR*:

  ```
    conv(x, a);
  or
    x = to_RR(a);
  ```

  Nasledujúci program zaokrúhli aktuálnu presnosť a výsledok uloží do x. Zápis:

  ```
    x = a + 0;
  or
    x = a*1;
  ```

- *RR* je reprezentovený ako mantisa/exponent dvojica (x, e), kde x je typu *ZZ* a e je typu *long*. Reálne čísla sú zastúpené prostredníctvom (x, e) ako $x * 2^e$. Nula je vždy zastúpená ako $(0, 0)$. Pre všetky zvyšné čísla, je x vždy nepárne.

2 Konvertovanie

Kompletný súbor prevodových podprogramov medzi RR a inými typmi je zdokumentovaný v súbore "conversions.txt". Zmena z hociakého typu do RR sa výsledok vždy približuje k aktuálnej presnosti. Základné operácie taktiež podporujú názor o "propagáciách", tak, aby propagovali RR typu *double*. Napríklad, môžme napísať $x = y + 1.5$; kde x a y sú RR. Mali by sme si byť vedomí, že tieto propagácie sú vždy zrealizované používaním prevodového programu *double* do RR.

Veľkosť invarianty: $max(NumBits(x), |e|) < 2^{(NTL\_BITS\_PER\_LONG-4)}$.

3 Aritmetické operácie

3.1 Operátory súčtu a rozdielu

Na sčítanie a odčítanie môžme použiť klasické operátory: $+$, $-$, $++$, $--$, $+=$, $-=$. K dispozícii sú aj **procedurálne verzie:**

```
void add(RR& z, const RR& a, const RR& b); // z = a+b
void sub(RR& z, const RR& a, const RR& b); // z = a-b
void negate(RR& z, const RR& a); // z = -a

// Podporuje typ double pre RR na (a, b).
```

3.2 Operátor násobenia

Na násobenie môžme použiť klasické operátory: $*$ a $*=$. K dispozícii sú aj **procedurálne verzie:**

```
void mul(RR& z, const RR& a, const RR& b); // z = a*b

void sqr(RR& z, const RR& a); // z = a * a
RR sqr(const RR& a);

// Podporuje typ double pre RR na (a, b).
```

3.3 Operátor delenia

Na delenie môžme použiť klasické operátory: / a /=. K dispozícii sú aj **procedurálne verzie:**

```
void div(RR& z, const RR& a, const RR& b); z = a/b

void inv(RR& z, const RR& a); // z = 1 / a
RR inv(const RR& a);

// Podporuje typ double pre RR na (a, b).
```

3.4 Rovnosti

Na porovnávanie môžme použiť klasické operátory: ==, ! =, <=, >=, <, >. Ďalej existujú funkcie *IsZero*, a *IsOne*. Obe fungujú pre všetky dátové typy, ktoré v NTL slúžia na reprezentáciu *RR*:

```
long IsZero(const RR& a);
//ak je na vstupe 0 vráti 1, inak 0
long IsOne(const RR& a);
//ak je na vstupe 1 vráti 1, inak 0

long sign(const RR& a);
//vracia znaky:
//ak je číslo kladné vráti +1
//ak je číslo záporné vráti -1
//ak je číslo nula vráti 0

long compare(const RR& a, const RR& b);
//vracia (a-b)
//ak je (a-b) kladné vráti +1
//ak je (a-b) záporné vráti -1
//ak je (a-b) nulový vráti 0
// Podporujú typ double pre RR na (a, b)
```

4 Transcendentné funkcie

4.1 Exponenciálne funkcie

4.1.1 exp

```
void exp(RR& res, const RR& x);  // e^x
RR exp(const RR& x);
```

4.1.2 expm1

```
void expm1(RR& res, const RR&  x);
RR expm1(const RR& x);
// e^(x)-1; exp(x)-1 je presnejší, keď |x| je malé
```

4.1.3 pow

```
void pow(RR& res, const RR& x, const RR& y);  // x^y
RR pow(const RR& x, const RR& y);
```

4.2 Logaritmické funkcie

4.2.1 log

```
void log(RR& res, const RR& x); // log(x) (natural log)
RR log(const RR& x);
```

4.2.2 log10

```
void log10(RR& res, const RR& x); // log(x)/log(10)
RR log10(const RR& x);
```

4.2.3 log1p

```
void log1p(RR& res, const RR& x);
RR log1p(const RR& x);
// log(1 + x); log(1 + x) je presnejší, keď |x| je malé
```

4.3 Goniometrické funkcie

4.3.1 sin

```
void sin(RR& res, const RR& x);  // sin(x); obmedzenie: |x| < 2^1000
RR sin(const RR& x);
```

4.3.2 cos

```
void cos(RR& res, const RR& x);  // cos(x); obmedzenie: |x| < 2^1000
RR cos(const RR& x);
```

4.4 ComputePi

```
void ComputePi(RR& pi); // aproximuje pi do aktuálnej presnosti
RR ComputePi_RR();
```

5 Zaokrúhlenie na celú hodnotu

/* RR výstup ***/**

5.1 trunc

```
void trunc(RR& z, const RR& a);
RR trunc(const RR& a);
// z = a, nezaokrúhluje - zobrazi celé číslo bez desatiných miest
```

5.2 floor

```
void floor(RR& z, const RR& a);
RR floor(const RR& a);
// z = a, zaokrúhluje nadol - zobrazi celé číslo bez desatiných miest
```

5.3 ceil

```
void ceil(RR& z, const RR& a);
RR ceil(const RR& a);
// z = a, zaokrúhluje nahor - zobrazi celé číslo bez desatiných miest
```

5.4 round

```
void round(RR& z, const RR& a);
RR round(const RR& a);
// z = a, zaokrúhluje k najbližšiemu celému číslu
```

/* ZZ výstup ***/**

5.5 TruncToZZ

```
void TruncToZZ(ZZ& z, const RR& a);
ZZ TruncToZZ(const RR& a);
// z = a, nezaokrúhluje - zobrazi celé číslo bez desatiných miest
```

5.6 FloorToZZ

```
void FloorToZZ(ZZ& z, const RR& a);
ZZ FloorToZZ(const RR& a);
// z = a, zaokrúhluje nadol - zobrazi celé číslo bez desatiných miest
// rovnako ako konverzia z RR do ZZ
```

5.7 CeilToZZ

```
void CeilToZZ(ZZ& z, const RR& a);
ZZ CeilToZZ(const ZZ& a);
// z = a, zaokrúhluje nahor - zobrazi celé číslo bez desatiných miest
```

5.8 RoundToZZ

```
void RoundToZZ(ZZ& z, const RR& a);
ZZ RoundToZZ(const RR& a);
// z = a, zaokrúhluje k najbližšiemu celému číslu
// väzby sú zaokrúhlené do rovného(celého) čísla
```

6 Rôzne funkcie

6.1 MakeRR

```
void MakeRR(RR& z, const ZZ& a,  long e);
RR MakeRR(const ZZ& a,  long e);
// z = a*2^e, zaokrúhlené na aktuálnu presnosť
```

6.2 random

```
void random(RR& z);
RR random_RR();
// z = pseudonáhodné číslo v rozsahu [0,1].
```

6.3 SqrRoot

```
void SqrRoot(RR& z, const RR& a); // z = sqrt(a);
RR SqrRoot(const RR& a);
RR sqrt(const RR& a);
```

6.4 abs

```
void abs(RR& z, const RR& a); // z = |a|
RR fabs(const RR& a);
RR abs(const RR& a);
```

6.5 power

```
void power(RR& z, const RR& a, long e); // z = a^e, e môže byť záporné
RR power(const RR& a, long e);
```

6.6 power2

```
void power2(RR& z, long e); // z = 2^e, e môže byť záporné
RR power2_RR(long e);
```

6.7 clear

```
void clear(RR& z);  // z = 0
void set(RR& z);  // z = 1
```

6.8 swap

```
void swap(RR& a, RR& b);  // vymení 'a' a 'b' (presunom smerníkov)
```

7 Špeciálne programy s jednoznačne presným parametrom

Programy zoberú explicitne presný parameter p. Hodnota p môže byť nejaké kladné číslo. Všetky výsledky sú vypočítané "presne"na p bitov presnosti, bez ohľadu na aktuálnu presnosť (nastaví sa prostredníctvom $RR :: SetPrecision$). Poskytnuté programy sú určené pre uľahčenie a pre situácie, kde môže byť výpočet s presnosťou menšou ako 53 bitov.

7.1 AddPrec

```
void AddPrec(RR& z, const RR& a, const RR& b, long p); // z = a + b
RR AddPrec(const RR& a, const RR& b, long p);
```

7.2 SubPrec

```
void SubPrec(RR& z, const RR& a, const RR& b, long p); // z = a - b
RR SubPrec(const RR& a, const RR& b, long p);
```

7.3 NegatePrec

```
void NegatePrec(RR& z, const RR& a, long p); // z = -a
RR NegatePrec(const RR& a, long p);
```

7.4 AbsPrec

```
void AbsPrec(RR& z, const RR& a, long p); // z = |a|
RR AbsPrec(const RR& a, long p);
```

7.5 MulPrec

```
void MulPrec(RR& z, const RR& a, const RR& b, long p); // z = a*b
RR MulPrec(const RR& a, const RR& b, long p);
```

7.6 SqrPrec

```
void SqrPrec(RR& z, const RR& a, long p); // z = a*a
RR SqrPrec(const RR& a, long p);
```

7.7 DivPrec

```
void DivPrec(RR& z, const RR& a, const RR& b, long p);  // z = a/b
RR DivPrec(const RR& a, const RR& b, long p);
```

7.8 InvPrec

```
void InvPrec(RR& z, const RR& a, long p);  // z = 1/a
RR DivPrec(const RR& a, long p);
```

7.9 SqrRootPrec

```
void SqrRootPrec(RR& z, const RR& a, long p); // z = sqrt(a)
RR SqrRootPrec(const RR& a, long p);
```

7.10 TruncPrec

```
void TruncPrec(RR& z, const RR& a, long p);
RR TruncPrec(const RR& a, long p);
// z = a, nezaokrúhluje - zobrazi celé číslo
```

7.11 FloorPrec

```
void FloorPrec(RR& z, const RR& a, long p);
RR FloorPrec(const RR& a, long p);
// z = a, zaokrúhluje nadol - zobrazi celé číslo
// a počet desatiných miest zodpovedajúci aktuálnej presnosti
```

7.12 CeilPrec

```
void CeilPrec(RR& z, const RR& a, long p);
RR CeilPrec(const RR& a, long p);
// z = a, zaokrúhluje nahor - zobrazi celé číslo
// a počet desatiných miest zodpovedajúci aktuálnej presnosti
```

7.13 RoundPrec

```
void RoundPrec(RR& z, const RR& a, long p);
RR RoundPrec(const RR& a, long p);
// z = a, zaokrúhluje k najbližšiemu celému číslu
// väzby sú zaokrúhlené do rovného(celého) čísla
```

7.14 InputPrec

```
void InputPrec(RR& z, istream& s, long p); // číta 'z' z 's'
RR InputPrec(istream& s, long p);
```

7.15 MakeRRPrec

```
void MakeRRPrec(RR& z, const ZZ& a, long e, long p); // z = a*2^e
RR MakeRRPrec(const ZZ& a, long e, long p);
```

8 Konverzie

8.1 ConvPrec_RR

```
void ConvPrec(RR& z, const RR& a, long p); // z = a
RR ConvPrec(const RR& a, long p);
```

8.2 ConvPrec_ZZ

```
void ConvPrec(RR& z, const ZZ& a, long p); // z = a
RR ConvPrec(const ZZ& a, long p);
```

8.3 ConvPrec_long

```
void ConvPrec(RR& z, long a, long p); // z = a
RR ConvPrec(long a, long p);
```

8.4 ConvPrec_int

```
void ConvPrec(RR& z, int a, long p); // z = a
RR ConvPrec(int a, long p);
```

8.5 ConvPrec_unsigned_long

```
void ConvPrec(RR& z, unsigned long a, long p); // z = a
RR ConvPrec(unsigned long a, long p);
```

8.6 ConvPrec_unsigned_int

```
void ConvPrec(RR& z, unsigned int a, long p); // z = a
RR ConvPrec(unsigned int a, long p);
```

8.7 ConvPrec_double

```
void ConvPrec(RR& z, double a, long p); // z = a
RR ConvPrec(double a, long p);
```

8.8 ConvPrec_xdouble

```
void ConvPrec(RR& z, const xdouble& a, long p); // z = a
RR ConvPrec(const xdouble& a, long p);
```

8.9 ConvPrec_quad_float

```
void ConvPrec(RR& z, const quad_float& a, long p); // z = a
RR ConvPrec(const quad_float& a, long p);
```

8.10 ConvPrec_char

```
void ConvPrec(RR& z, const char *s, long p); // číta 'z' z 's'
RR ConvPrec(const char *s, long p);
```

Poznámky ku kompatibilite

1. Pred verziou 5.3, dokummentácia naznačila že za určitých okolností, hodnota aktuálnej presnosti môže byť priamo nastavená nastavením premennej $RR :: prec$. Takéto použitie je teraz považované za zastaralé. Ak chcete vykonávať výpočty používajúce presnosť menšiu ako 53 bitov, mali by ste používať špecializované programy napr.: AddPrec, SubPrec, atď., zdokumentované vyššie.

2. Program *RoundToPrecision* je zastaralý, ale pre spätnú kompatibilitu je stále deklarovaný(v procedurálnej i funkčnej forme). Ekvivalentná funkcia je *ConvPrec*.

3. Vo verzii 2.0 a starších, operátor riadenia a kopírovací konštruktor pre triedu *RR* zaokrúhľuje výstupy k aktuálnej presnosti. Toto už nieje ten istý prípad: ich výstupy sú teraz presné kópie ich vstupov, bez ohľadu na aktuálnu presnosť.

9 Vstup/výstup

Vstupná syntax:

```
<číslo>: [ "-" ] <číslo bez znamienka>
<číslo bez znamienka>: <číslo s bodkou> [ <e-časť> ] | <e-časť>
<číslo s bodkou>: <číslice> | <číslice> "." <číslice> | "." <číslice> | <číslice> "."
<číslice>: <číslica> <číslice> | <číslica>
<číslica>: "0" | ... | "9"
<e-časť>: ( "E" | "e" ) [ "+" | "-" ] <číslica>
```

Príklady platného vstupu:
17 | 1.5 | 0.5 | .5 | 5. | -.5 | e10 | e-10 | e+10 | 1.5e10 | .5e10 | .5E10

Poznámky:

- Počet desatiných miest presnosti, ktoré sú používané pre výstup, si môžeme urobyť pomocou čísla $p >= 1$ volaním štandarnej funkcie $RR :: SetOutputPrecision(p)$. Implicitná hodnota p je 10. Aktuálna hodnota p je vrátená volaním funkcie $RR :: OutputPrecision(p)$.

- `istream& operator >> (istream& s, RR& x);`
  `ostream& operator << (ostream& s, const RR& a);`