Quad float

Čísla so štvornásobnou presnosťou s pohyblivou desatinou čiarkou

Obsah

1 Na úvod **2**

2 Aritmetické operácie **3**
2.1 Operátory súčtu a rozdielu 3
2.2 Operátor násobenia 3
2.3 Operátor delenia 3
2.4 Rovnosti 3

3 Rôzne funkcie **4**
3.1 floor 4
3.2 ceil 4
3.3 trunc 4
3.4 fabs 4
3.5 exp 4
3.6 log 4
3.7 power 4
3.8 power2 4
3.9 ldexp 4
3.10 IsFinite 5
3.11 random 5

4 Vstup/výstup **5**

1 Na úvod

Trieda $quad\_float$ je použitá k reprezentácii čísel so štvornásovbnou presnosťou s pohyblivou desatinou čiarkou. A tak, so štandardom IEEE o pohyblivej rádovej čiarke, by ste mali dostať ekvivalent presnosti okolo 106 bitov (ale v skutočnosti je to o truchu menej).

Rozhranie Vám umožňuje pracovať so štvornásobnou presnosťou, v podstate ako by to boli typy s "normálnou"pohyblivou rádovou čiarkou(floating point).

Pre používanie tejto triedy je potrebné naimportovať knižnicu nasledovným spôsobom:

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

Funkcia pre nastavenie výstupnej presnosti na požadovaný počet desatiných miest má tvar:

```
quad_float::SetOutputPrecision(x);
// kde x je číslo predstavujúce počet desatiných miest
```

Viac implementačných detailov nájdete nižšie.

2 Aritmetické operácie

2.1 Operátory súčtu a rozdielu

Na sčítanie a odčítanie môžme použiť klasické operátory: +, −, ++, −−, + =, − =.

2.2 Operátor násobenia

Na násobenie môžme použiť klasické operátory: ∗ a ∗ =.

2.3 Operátor delenia

Na delenie môžme použiť klasické operátory: / a / =.

2.4 Rovnosti

Na porovnávanie môžme použiť klasické operátory: ==, ! =, <=, >=, <, >. Ďalej existujú:

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

long compare(const quad_float& x, const quad_float& y);
//vracia (x-y)
//ak je (x-y) kladné vráti +1
//ak je (x-y) záporné vráti -1
//ak je (x-y) nulový vráti 0
// Podporuje typ double pre quad_float na (x, y)
```

K predchádzjúcim operáciam neexistujú funkcie v procedurálnej forme.

3 Rôzne funkcie

```
quad_float sqrt(const quad_float& x);
//x = sqrt(a);
```

3.1 floor

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

3.2 ceil

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

3.3 trunc

```
quad_float trunc(const quad_float& x);
// x = a, nezaokrúhluje - zobrazi celé číslo bez desatiných miest
```

3.4 fabs

```
quad_float fabs(const quad_float& x);
// x = |a|
```

3.5 exp

```
quad_float exp(const quad_float& x);
// e^x
```

3.6 log

```
quad_float log(const quad_float& x);
log(x)
```

3.7 power

```
void power(quad_float& x, const quad_float& a, long e); // x = a^e
quad_float power(const quad_float& a, long e);
```

3.8 power2

```
void power2(quad_float& x, long e); // x = 2^e
quad_float power2_quad_float(long e);
```

3.9 ldexp

```
quad_float ldexp(const quad_float& x, long e);  // vracia x*2^e
```

3.10 IsFinite

```
long IsFinite(quad_float *x); // kontrola ak x je "ohraničená"
//ukazovateľ(pointer) je používaný pre kompatibilitu s IsFinite(double*)
```

3.11 random

```
void random(quad_float& x);
quad_float random_quad_float();
// generuje náhodné quad_float x s intervalu 0 <= x <= 1
```

4 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 $quad\_float :: SetOutputPrecision(p)$. Implicitná hodnota p je 10. Aktuálna hodnota p je vrátená volaním funkcie $quad\_float :: OutputPrecision(p)$.
- `istream& operator >> (istream& s, quad_float& x);`
  `ostream& operator << (ostream& s, const quad_float& x);`