Xdouble

Čísla s dvojitou presnosťou s pohyblivou desatinou čiarkou s rozšíreným rozsahom exponentu (pre veľmi veľké čísla)

Obsah

1 Na úvod

Trieda *xdouble* je použitá k reprezentácii čísel s pohyblivou rádovou čiarkou s rovnakou presnosťou ako *'double'*, ale s rozšíreným rozsahom exponentu (ponúka niekoľko bitov naviac pre 'dĺžku' exponentu). Programovacie rozhranie pre *xdouble* je skoro identické ako *double*.

Pre používanie tejto triedy je potrebné naimportovať knižnicu nasledovným spôsobom:

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

1.1 Implementačné detaily

$Xdouble$ je dvojica (x, e)reprezentovan8 ako $mantisa/exponent$, kde x je typu *double* a e je typu *long*. Reálne čísla (x, e) sú reprezentované prostredníctvom $x * NTL\_XD\_BOUND^e$, kde

$$NTL\_XD\_BOUND = NTL\_XD\_HBOUND^2$$
$$NTL\_XD\_HBOUND = 2^{\{(max(NTL\_DOUBLE\_PRECISION, NTL\_BITS\_PER\_LONG)+4)\}}.$$

Taktiež, mantisa x vyhovuje $1/NTL\_XD\_HBOUND \leq |x| \leq NTL\_XD\_HBOUND$, okrem čísla 0, ktorá je reprezentovaná ako $(0, 0)$.

Obe NTL_XD_BOUND a NTL_XD_HBOUND sú makrá definované v $< NTL/xdouble.h >$.

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

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 Rovnosti

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

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

long compare(const xdouble& a, const xdouble& b); // vracia (a - 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
// Podporuje typ double pre xdouble na (a, b)
```

3 Rôzne funkcie

3.1 trunc

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

3.2 floor

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

3.3 ceil

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

3.4 fabs

```
xdouble fabs(const xdouble& a);
//z = |a|
```

3.5 sqrt

```
xdouble sqrt(const xdouble& a);
// z = a^{1/2} resp. z = sqrt(a);
//chyba je vyvolaná ak a < 0
```

3.6 log

```
double log(const xdouble& a);
// log(a) (vrátená hodnota je typu double!)
```

3.7 xexp

```
xdouble xexp(double a);
// exp(a) (argument je typu double!)
```

3.8 power

```
void power(xdouble& z, const xdouble& a, const ZZ& e);
xdouble power(const xdouble& a, const ZZ& e);

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

3.9 power2

```
void power2(xdouble& z, long e);
xdouble power2_xdouble(long e);
// z = 2^e, e môže byť záporné
```

3.10 MulAdd

```
void MulAdd(xdouble& z, const xdouble& a, const xdouble& b, const xdouble& c);
xdouble MulAdd(const xdouble& a, const xdouble& b, const xdouble& c);
// z = a + b*c, ale rýchlejšie
```

3.11 MulSub

```
void MulSub(xdouble& z, const xdouble& a, const xdouble& b, const xdouble& c);
xdouble MulSub(const xdouble& a, const xdouble& b, const xdouble& c);
// z = a - b*c, ale rýchlejšie
```

4 Vstup/výstup

```
<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 $xdouble :: SetOutputPrecision(p)$. Implicitná hodnota p je 10. Aktuálna hodnota p je vrátená volaním funkcie $xdouble :: OutputPrecision()$.

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