acm.mipt.ru

олимпиады по программированию на Физтехе

Раздел «Алгоритмы» . LongArithmeticsCPP :

Реализация длинной арифметики на C++

Длинная арифметика: поддерживает сложение, вычитание, умножение, деление столбиком, отрицательные и дробные числа. В дальнейшем, возможно, добавлю оптимизацию, и другой хитрый метод длинного деления с помощью вычисления 1/x. Сообщения о багах а также любая критика приветствуются :)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>


// Количество цифр после запятой
#define NEG_ACC 30
// Количество цифр до запятой
#define POS_ACC 30
// Макрос, выцепляющий i-ю цифру (положительные i -
// цифры после запятой, отрицательные - до)
#define DIGIT(longnum, i) ( (longnum).data[(i)+NEG_ACC] )
// Собственно тип длинного числа
struct LongNum{
   LongNum(){
      // при инициализации записываем ноль
      flush();
   }
   void flush(){
      // функция обнуления числа
      int i;
      for(i=0;i<NEG_ACC+POS_ACC;i++)
         data[i]=0;
      sign = 1;
   }

   // функция, выцепляющая i-ю цифру
   char inline digit(int i) const {return data[i+NEG_ACC];}
   // массив, в котором хранятся цифры
   char data[NEG_ACC+POS_ACC];
   // знак: +1 или -1
   signed char sign;
};

// вводит с клавиатуры длинное число
struct LongNum LNinput()
{
   char s[NEG_ACC+POS_ACC+10];
   int i,j,k;
   struct LongNum res;
   cin >> s;

   // проверяем на унарный минус
   if(s[0]=='-'){
      res.sign = -1;
      i=1;
   }else{
      res.sign = 1;
      i=0;
   }

   // считываем по цифирке; особое внимание - точке
```

```
   // поскольку мы пока не знаем в какой разряд должна попасть
   // первая цифра, то пихаем все начиная со старшего разряда
   j = POS_ACC-1;
   for(;i<strlen(s) && s[i]!='.';i++){
      DIGIT(res, j--) = s[i]-'0';
   }
   // а теперь, когда мы встретили либо конец строки, либо
   // десятичную запятую, сдвигаем все цифры на нужные разряды ...
   j++;
   for(k=j;k<POS_ACC;k++)
      DIGIT(res, k-j) = DIGIT(res,k);
   // ... и очищаем загаженные старшие разряды
   for(k=POS_ACC;k-j<POS_ACC;k++)
      DIGIT(res, k-j) = 0;
   // если встретили точку, то дочитываем дробные разряды
   if(s[i]=='.'){
      j=-1;
      for(i++;i<strlen(s);i++)
         DIGIT(res, j--) = s[i]-'0';
   }
   return res;
}

// выводит на экран длинную чиселку
void LNoutput(struct LongNum ln)
{
   if(ln.sign == -1)cout << '-';
   // существует два типа цифр 0 (ноль):
   // ведущие нули, которые не следует выводить
   // и нули в середине числа, которые нужно вывести
   // (замыкающие нули - это другая история)
   // переменная flag определяет встретили ли мы какую-либо
   // значащую цифру при движении от старших разрядов к младшим
   // если у нас были значащие цифры и мы наткнулись на ноль, то
   // его надо вывести, т.к. он в середине числа
   int i, j, flag=0;
   for(i=POS_ACC-1;i>=0;i--)
      if( flag || ln.digit(i)!=0 ){
         flag = 1;
         cout << (char)(ln.digit(i)+'0');
      }
   // если значащих цифр нет, надо хоть что-то вывести :)
   if(!flag)cout << '0';
   // проверяем, есть ли дробная часть у числа, и запоминаем
   // в j ее конец ...
   j = 0;
   for(i=-NEG_ACC;i<0;i++)
      if(ln.digit(i)!=0){
         j = i;
         break;
      }
   // ... если есть, то выводим
   if(j != 0){
      cout << '.';
      for(i=-1; i>=j; i--)
         cout << (char)(ln.digit(i)+'0');
   }
}

// переводит короткое число в длинное
struct LongNum toLN(int n)
{
   struct LongNum res;
   if(n<0){
      res.sign = -1;
      n = -n;
   } else res.sign = 1;
   int i=0,p10=1;
   // вычисляем степень 10, соответствующую максимальному
```

```
    // разряду короткого целого
    while((p10*10)/10==p10){
        p10*=10;
        i++;
    }
    // поразрядно выделяем цифры короткого целого
    for(;i>=0;i--){
        DIGIT(res, i) = n/p10;
        n %= p10;
        p10 /= 10;
    }
    return res;
}

// сравнение ПО МОДУЛЮ двух длинных чисел
// возвращает 1 если |a|>|b|; -1 если |a|<|b|
// и возвращает 0 если они равны
int LNabscmp(const struct LongNum &a, const struct LongNum &b)
{
    int i;
    // сравниваем по разрядам, начиная со старших
    for(i=POS_ACC-1; i>=-NEG_ACC; i--){
        if(a.digit(i)==b.digit(i))continue;
        if(a.digit(i)>b.digit(i))return 1;
        if(a.digit(i)<b.digit(i))return -1;
    }
    return 0;
}

// сравнение двух длинных чисел
// возвращает 1 если a>b ; -1 если a<b
// и возвращает 0 если они равны
int LNcmp(const struct LongNum &a, const struct LongNum &b)
{
    if(a.sign == 1 && b.sign == -1)return 1;
    if(a.sign == -1 && b.sign == 1)return -1;
    if(a.sign == 1 && b.sign == 1)return LNabscmp(a,b);
    if(a.sign == -1 && b.sign == -1)return -LNabscmp(a,b);
    return 0;//never executed
}

// возвращает ln-subtractment
// описана ниже
void LNsubLN(struct LongNum &ln, struct LongNum subtractment);

// возвращает ln+addment
// результат сохраняется в ln
void LNaddLN(struct LongNum &ln, struct LongNum addment)
{
    int i,p;
    // если числа одного знака
    if(ln.sign == addment.sign){
        // в p сохраняем перенос единицы при сложении столбиком
        p=0;
        // поразрядно складываем
        for(i=-NEG_ACC;i<POS_ACC;i++){
            p += ln.digit(i) + addment.digit(i);
            DIGIT(ln, i) = p%10;
            p/=10;
        }
    }else{
        // сюда попадаем если числа разных знаков
        // если ln<0 то переставим их местами
        if(ln.sign == -1){
            struct LongNum dummy = ln;
            ln = addment;
            addment = dummy;
        }
        int cmp = LNabscmp(ln,addment);
```

```
      if(cmp == 0)
         // если числа равны по модулю и при этом
         // разных знаков, то их сумма - ноль
         ln.flush();
      else if(cmp > 0){
         addment.sign = 1;
         LNsubLN(ln, addment);
      }else{ // cmp < 0
         addment.sign = 1;
         LNsubLN(addment, ln);
         ln = addment;
         ln.sign = -1;
      }
   }
}

// возвращает ln-subtractment
// результат сохраняется в ln
void LNsubLN(struct LongNum &ln, struct LongNum subtractment)
{
   // пробуем сделать так:   a-(-b) == a+b
   if(subtractment.sign == -1){
      subtractment.sign = 1;
      LNaddLN(ln, subtractment);
      return;
   }

   // здесь пробуем так:   (-a)-b == -(a+b)
   if(ln.sign == -1){
      ln.sign = 1;
      LNaddLN(ln, subtractment);
      ln.sign = -ln.sign;
      return;
   }

   // здесь ln.sign == subtractment.sign == 1
   int cmp = LNabscmp(ln, subtractment);
   if(cmp == 0){
      ln.flush();
      return;
   }

   // тут если a>0, b>0, a<b, то a-b == -(b-a)
   // причем b-a>0
   if(cmp < 0){
      struct LongNum dummy = ln;
      ln = subtractment;
      subtractment = dummy;
      LNsubLN(ln, subtractment);
      ln.sign = -ln.sign;
      return;
   }

   // а вот здесь честно вычитаем по разрядам
   int i, p=0;
   for(i=-NEG_ACC; i<POS_ACC; i++){
      p = ln.digit(i)-subtractment.digit(i)-p;
      DIGIT(ln, i) = (p+10)%10;
      if(p<0)p=1;else p=0;
   }
}

// умножает ln на multiplier
// результат сохраняется в ln
void LNmulLN(struct LongNum &ln, struct LongNum multiplier)
{
   ln.sign *= multiplier.sign;
   int i, j, k, p;
   struct LongNum res;
```

```
   // честно умножаем столбиком, и пока кидаем все в res
   for(i=-NEG_ACC; i<POS_ACC; i++){
      // перебираем только такие j, что разряд, куда
      // помещается произведение i-й цифры первого числа на
      // j-ю второго был в пределах NEG_ACC..POS_ACC
      j=-NEG_ACC-i-1;
      if(j<-NEG_ACC)j=-NEG_ACC;
      for(; i+j<POS_ACC && j<POS_ACC; j++){
         p = ln.digit(i) * multiplier.digit(j);
         k = i+j;
         // распихиваем p по разрядам (это нужно если p>=10)
         while(p>0 && k<POS_ACC){
            if(k>= -NEG_ACC && k<POS_ACC)
               DIGIT(res, k) += p%10;
            p /= 10;
            if(DIGIT(res,k) >= 10){
               p += DIGIT(res,k)/10;
               DIGIT(res,k) %= 10;
            }
            k++;
         }
      }
   }
   res.sign = ln.sign;
   ln = res;
}

// сдвигает ln на displacement десятичных разрядов влево
// может сдвигать вправо, если displacement<0
void LNshlLN(struct LongNum &ln, int displacement)
{
   int i;
   struct LongNum res;
   for(i=-NEG_ACC; i<POS_ACC; i++)
      if(
         i-displacement >= -NEG_ACC &&
         i-displacement < POS_ACC
      )
         DIGIT(res, i) = ln.digit(i-displacement);
   ln = res;
}

// возвращает в ln значение ln/dividor
void LNdivLN(
   struct LongNum &ln, struct LongNum divider)
{
   struct LongNum res, zero;
   res.flush(); zero.flush();
   int res_sign = ln.sign * divider.sign;
   ln.sign = divider.sign = 1;
   int p_ln, p_divider;
   // если кто-то нуль, то нечего считать
   if(LNcmp(ln, zero) == 0) return;
   if(LNcmp(divider, zero) == 0){
      // делить на ноль ни в коем случае нельзя!
      ln.flush();
      return;
   }
   // найдем порядки чисел
   int i, j;
   for(i=POS_ACC-1;i>=-NEG_ACC;i--)
      if(ln.digit(i) != 0){
         p_ln = i;
         break;
      }
   for(i=POS_ACC-1;i>=-NEG_ACC;i--)
      if(divider.digit(i) != 0){
         p_divider = i;
```

```
            break;
        }
    // вычислим как надо сдвинуть делитель, чтобы его порядок
    // совпал с делимым
    i = p_ln-p_divider;
    if(i>POS_ACC) i = POS_ACC-1-p_divider;
    // сдвигаем...
    LNshlLN(divider, i);
    struct LongNum tempnum;
    // по разрядам вычисляем частное
    for(;i>-NEG_ACC;i--){
        tempnum.flush();
        // подбираем такое j и tempnum == j*divider,
        // где divider уже сдвинут на i разрядов
        // чтобы j*divider <= ln и (j+1)*divider > ln
        // в таком случае j - это цифра i-го разряда частного
        j=0;
        do{
            LNaddLN(tempnum, divider);
            j++;
        }while(LNabscmp(tempnum, ln) <= 0);
        LNsubLN(tempnum, divider);
        j--;
        // ок, подобрали j, теперь вычитаем из ln tempnum
        // (как при делении столбиком)
        LNsubLN(ln, tempnum);
        DIGIT(res, i) = j;

        // сдвигаем вправо на один разряд для вычисления следущей цифры
        LNshlLN(divider, -1);
    }
    res.sign = res_sign;
    ln = res;
}



// небольшая программа, демонстрирующая правильность работы
int main()
{
    struct LongNum ln1, ln2, lnsum, lndif, lnmul, lndiv;
    ln1 = toLN(11);
    ln2 = toLN(1);
    LNdivLN(ln1, toLN(10));
    LNdivLN(ln2, toLN(10));
    LNoutput(ln1);
    cout << endl;
    //int n;
    //cin >> n;
    //ln2 = toLN(n);
    LNoutput(ln2);
    cout << endl;
    lnsum = ln1;
    LNaddLN(lnsum, ln2);
    lndif = ln1;
    LNsubLN(lndif, ln2);
    cout << "a+b=";
    LNoutput(lnsum);
    cout << " a-b=";
    LNoutput(lndif);
    cout << endl << "a*b=";
    lnmul = ln1;
    LNmulLN(lnmul, ln2);
    LNoutput(lnmul);
    cout << endl << "a/b=";
    lndiv = ln1;
    LNdivLN(lndiv, ln2);
    LNoutput(lndiv);
    cout << endl;
```

```
    return 0;
}
```

-- AntonMalykh - 18 Mar 2004