

Министерство образования и науки РФ  
Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования

Тульский государственный университет

КАФЕДРА ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

## **ЧИСЛА С ПЛАВАЮЩЕЙ ТОЧКОЙ. ДЛИННЫЕ ЧИСЛА**

Лабораторная работа № 4  
по курсу «Структуры и алгоритмы обработки данных»

Вариант № 4

Выполнил:	студент группы 220601	_____	Белым А.А.
		(подпись)	
Проверил:	д. ф.-м.н, проф.каф. ИБ	_____	Двоенко С.Д.
		(подпись)	

Тула 2013

## Цель работы

Изучить основы работы с длинными числами, научиться выполнять различные арифметические действия над большими числами и числами с плавающей точкой.

## Задание

Напишите функцию вычисления факториала. В качестве тестового примера подсчитайте  $100000789787!$

## Теоретическая справка

Иногда при вычислениях приходится обрабатывать очень длинные числа, которые выходят за пределы разрядной сетки вычислительной установки. Такие задачи невозможно решить, используя базовые типы данных, встроенные в язык программирования и распознаваемые компилятором. Из-за ограничений на разрядность при обработке таких данных, будет получено или очень большое число, превышающее максимально допустимое значение стандартного типа, или точность результата, которая нас не устраивает.

Способ представления зависит от цели вычислений. Обычно, целое большое число  $N$  представляется в виде

$$X = x_0 + x_1 B + x_2 B^2 + \dots + x_n B^n,$$

дробное представляется в виде

$$X = x_0 + \frac{x_1}{B} + \frac{x_2}{B^2} + \dots + \frac{x_n}{B^n}.$$

Здесь  $B$  - основание системы счисления, в которой записывается число. Все  $x_i$  - стандартные числа (long или double, например) и  $0 \leq x_i < B$ .

Знак числа либо хранится отдельно, либо все  $x_i < 0$ , если число отрицательное.

Основание  $B$  обычно зависит от максимального размера базового типа данных на компьютере, и выбирается, исходя из следующих соображений:

1. Основание  $B$  подходит под один из базовых типов данных,

2. Основание  $B$  должно быть как можно больше, чтобы уменьшить размер представления длинного числа и увеличить скорость операций с ними, но достаточно малого размера, чтобы все операции с коэффициентами использовали базовый тип данных.

Для удобства можно выбрать  $B$  как степень 10 (вывод информации, отладка).  $B$  - степень двойки позволяет проводить быстрые операции на низком уровне. Нужно иметь в виду, что быстрый переход от основания  $B = 2^m$  к  $B = 10^n$  для больших чисел реализовать весьма непросто.

Для коэффициентов  $x_i$  естественно выбрать тип long (в Си). Но тип long обычно занимает 32 бита, а использование double (Си - обычно 64 бита) дает возможность получить большее основание.

Кроме того, некоторые процессоры (Pentium) специально оптимизированы для операций с плавающей точкой, поэтому лучше выбирать double. При этом появляется возможность заменить деление на число умножением предварительно вычисленное обратное ему.

Основной (и решающий в серьезных пакетах) недостаток выбора типа double состоит в том, что для перехода на качественно иной уровень быстродействия потребуется реализовать многие операции на низком уровне, используя внутреннее машинное представление числа. Это достаточно удобно делается только с целым типом (например, long).

## Текст программы

Далее представлен текст программы на языке C++, реализующей вычисление факториала в формате длинных чисел.

```
#include <iostream>
#include <cstdio>
#include <sstream>
#include <iomanip>
typedef unsigned long long uint;

const uint SIZE=(sizeof(uint))*8,
        SIZE2=SIZE/2,
        POW2=uint(1)<<SIZE2;
const uint MAX_SIZE=10000;
using namespace std;
inline void addc(uint& a,const uint& b,uint& carry){
    a+=b;
    if(a<b)
        ++carry;
```

```

}

void long_len_correct(uint *src,uint &len){
    for(uint *ptr=src+len-1;!( *ptr );ptr--,len--);
}
void long_clear(uint *src,uint len){
    for(;len;src++,len--)
        *src=0;
}

void mult1(uint a,uint b,uint *res,uint& car){
    uint r0=(a%POW2)*(b%POW2),
        r01=(a%POW2)*(b>>SIZE2),
        r10=(a>>SIZE2)*(b%POW2),
        r1=(a>>SIZE2)*(b>>SIZE2);
    uint car0=0;
    addc(res[0],r0,car0);
    addc(res[0],(r01%POW2)<<SIZE2,car0);
    addc(res[0],(r10%POW2)<<SIZE2,car0);

    addc(res[1],car0,car);
    addc(res[1],r1,car);
    addc(res[1],r01>>SIZE2,car);
    addc(res[1],r10>>SIZE2,car);
}

void long_mult(uint *a, uint la,uint *b, uint lb, uint *res,uint &len){
    uint i,j,car0=0,car1=0,car2=0,maxi=la+lb-1;
    long_clear(res,la+lb);
    for(i=0;i<=maxi;i++){
        addc(res[i],car0,car1);
        for(j=0;j<=i;j++){
            if(j>=la)
                break;
            if((i-j)>=lb)
                continue;
            mult1(a[j],b[i-j],res+i,car2);
        }
        car0=car1;car1=car2;car2=0;
    }
    len=maxi+1;
    if(car0){
        len=maxi+1;
        res[len]=car0;
        len++;
    }
    if(car1){
        len=maxi+2;
        res[len]=car1;
        len++;
    }
    if(car2){
        len=maxi+3;
        res[len]=car2;
        len++;
    }
    long_len_correct(res,len);
}

void print_long(uint* num,uint len){
    for(uint *ptr=num+len-1;len;--ptr,--len){
        cout<<hex<<setw(sizeof(uint)*2)<<setfill('0')<<*ptr;
    }
    cout<<endl;
}

```

```

void long_inc(uint *res,uint &len){
    uint car=0,car0=0;uint i;
    addc(*res,1,car0);
    res++;
    for(i=len-1;i&&car0;i--,res++){
        addc(*res,car0,car);
        car0=car;car=0;
    }
    if(!i&&car0){
        len++;
        *res=car0;
    }
}

bool long_le(uint *a,uint la,uint *b,uint lb){
    if(la<lb)
        return true;
    else if(la>lb)
        return false;
    else{
        uint *p1=a+la-1,*p2=b+lb-1;
        for(;la&&*p1==*p2;la--,p1--,p2--);
        return !la||*p1<*p2;
    }
}

void long_fact(uint* src, uint lsrc,uint *res0,uint &lres){
    uint *res=res0,*t;
    uint *buf1=new uint[MAX_SIZE],
        *buf2=new uint[MAX_SIZE];
    uint lb1=1,lb2=1;
    long_clear(buf1,MAX_SIZE); long_clear(buf2,MAX_SIZE);
    buf1[0]=1;buf2[0]=2;
    while(long_le(buf2,lb2,src,lsrc)){
        long_clear(res,lres);
        long_mult(buf1,lb1,buf2,lb2,res,lres);
        lb1=lres;
        t=buf1;buf1=res;res=t;
        long_inc(buf2,lb2);
    }
    if(buf1!=res0)
        long_copy(buf1,lres,res);
    else{
        t=buf1;buf1=res;res=t;
    }
    delete buf1;
    delete buf2;
}

void long_from_string(const string &str,uint *res,uint &len){
    uint block_size=SIZE/4;
    int i; len=0;
    for(i=str.length()-block_size;i>=0;i-=block_size){
        istringstream istr(str.substr(i,block_size));
        istr>>hex>>*res;
        len++;res++;
    }
    if((i+block_size)>0){
        istringstream istr(str.substr(0,i+block_size));
        istr>>hex>>*res;
        len++;res++;
    }
}

void input_long(uint *res,uint &len){

```

```

    string s;
    cin>>s;
    long_clear(res,len);
    long_from_string(s,res,len);
}

int main()
{
    uint base[MAX_SIZE],res[MAX_SIZE],len,base_len=MAX_SIZE;
    cout<<"Программа вычисляет N!."<<endl;
    cout<<"Введите N."<<endl;
    input_long(base,base_len);
    long_fact(base,base_len,res,len);
    cout<<"N! ="<<endl;
    print_long(res,len);
    return 0;
}

```

### Тестовый пример

В связи с тем, что факториал очень быстро возрастает при росте своего аргумента, расчет значения  $100000789787!$  предъявляет слишком высокие требования к памяти.

Для того, чтобы оценить размер числа в битах, можно использовать  $\log_2$ . Для расчета логарифма факториала примем во внимание гамма-функцию

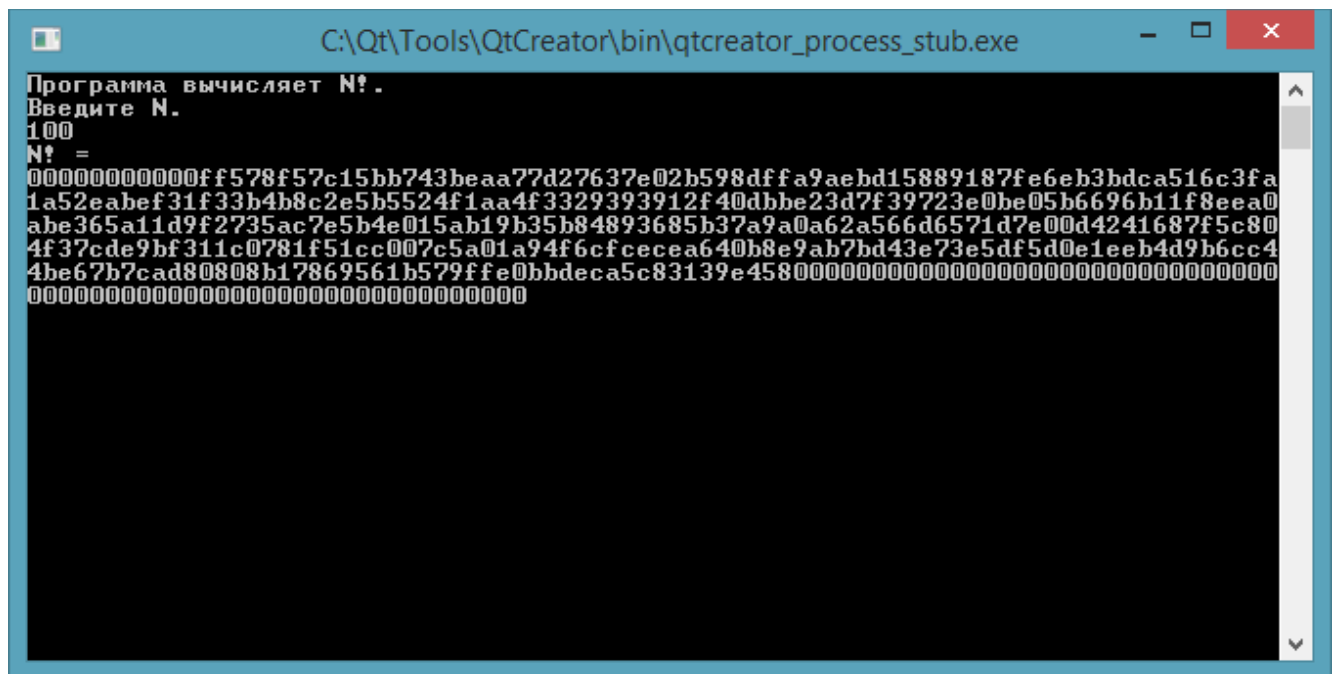
$\Gamma(n)=(n-1)!$ , а также то, что  $\ln(\Gamma(n)) \approx (n-\frac{1}{2})\ln(n) - n + \frac{1}{2}\ln(2\pi)$ . Тогда

$\ln(100000789787!) = \ln(\Gamma(100000789788)) \approx 2432863606379.6533$ ,

$\log_2(100000789787!) = \frac{\ln(100000789787!)}{\ln(2)} \approx 3509880260083.165$  бит. Если поделить

количество бит на 8, получим требуемый размер в байтах, а после этого еще на  $2^{30}$  - в гигабайтах. Итого получается, что для хранения подобного числа требуется около 408.6 гигабайт, что многократно превосходит объемы оперативной памяти персонального компьютера.

На рисунке 1 представлен пример работы программы, реализующей вычисление факториала в формате длинных чисел.



```
C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
Программа вычисляет N!.
Введите N.
100
N! =
000000000000ff578f57c15bb743beaa77d27637e02b598dfffa9aebd15889187fe6eb3bdca516c3fa
1a52eabef31f33b4b8c2e5b5524f1aa4f3329393912f40dbbe23d7f39723e0be05b6696b11f8eea0
abe365a11d9f2735ac7e5b4e015ab19b35b84893685b37a9a0a62a566d6571d7e00d4241687f5c80
4f37cde9bf311c0781f51cc007c5a01a94f6cfcecea640b8e9ab7bd43e73e5df5d0e1eeb4d9b6cc4
4be67b7cad80808b17869561b579ffe0bbdeca5c83139e45800000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Рисунок 1— Пример работы программы вычисления факториала в формате длинных чисел

### Вывод

В данной работе я познакомился с длинными числами и основными операциями над ними. Была написана программа, реализующая вычисление факториала в формате длинных чисел.