

[<< Back to CodeChef](#)[questions](#)[tags](#)[users](#)[badges](#)[unanswered](#)[ask a question](#)[about](#)

CodeChef Discussion

Search Here...

☒ questions ☐ tags ☐ u:

Computing Factorials of a huge number in C/C++: A tutorial

Hello @all,

109 As in MARCH13 contest we needed to use primary school arithmetics once again, and as it is a topic that comes up quite frequently here in the forums thanks to [this](#) problem, and also, as I don't see a complete and detailed tutorial on this topic here, I decided I'd write one before my Lab class at university :P (spending free time with Codechef is always a joy!!)

59

- Some preliminary experiences with C

If we want to implement a factorial calculator efficiently, we need to know what we are dealing with in the first place...

Some experiences in computing factorials iteratively with the following code:

```
#include <stdio.h>

long long int factorial(int N)
{
    long long ans = 1;
    int i;
    for(i=1; i <= N; i++)
        ans *= i;
    return ans;
}

int main()
{
    int t;
    for(t=1; t <= 21; t++)
    {
        printf("%lld\n", factorial(t));
    }
    return 0;
}
```

will produce the following output on Ideone:

```
1
2
6
24
120
720
5040
40320
362880
3628800
39916800
479001600
6227020800
87178291200
1307674368000
20922789888000
355687428096000
6402373705728000
121645100408832000
2432902008176640000
-4249290049419214848
```

So, we can now see that even when using the long long data type, the maximum factorial we can expect to compute correctly, is only 20!

When seen by this point of view, suddenly, 100! seems as an impossible limit for someone using C/C++, and this is actually **only partially true**, such that we can say:

It is impossible to compute factorials larger than 20 when using built-in data types.

Follow this question

By Email:

Once you sign in you will be able to subscribe for any updates here

By RSS:

[Answers](#)[Answers and Comments](#)

Question tags:

[tutorial](#) **x630**[factorial](#) **x175**[bignum](#) **x87**[fctrl2](#) **x79**

question asked: 13 Mar '13, 21:12

question was seen: 212,622 times

last updated: 13 Oct '17, 02:06

Related questions

[How to print a large number](#)[small factorial ,wrong answer?can anyone point out the error?](#)[Factorials for large numbers](#)[\[closed\] FCTRL2 :getting wa](#)[My trailing zeros shell script is getting time limit exceeded , any help on this?](#)[find number fact](#)[FCTRL2 executes in my system but codechef doesn't accept](#)[!!!!!!!!!!!!!!SPOT THE ERROR IF CAN!!!!!!!!!!!!!!!!!!!!\(factorial of large no,c++\)](#)[GOVT - EDITORIAL](#)[Storing very large number](#)

You are not logged in. Please login at www.codechef.com to post your questions!



and variable precision arithmetic libraries implemented?).

The answer to these questions is surprisingly and annoyingly "basic" and "elementar", in fact, we shall travel back to our primary school years and apply what was taught to most of us when we were 6/7/8 years old.

I am talking about doing all operations by hand!!

- The underlying idea behind long operations and how to map it into a programming language

$12 \times 11 = 132$

Any programming language will tell you that. But, so will any 8 year old kid that's good with numbers. But, the kid's way of telling you such result is what we are interested in:

Here's how he would do it:

```

    12
  x 11
  ----
    12
   +12
  ----
   132

```

But, why is this idea way more interesting than simply doing it straightforwardly? It even looks harder and more error-prone... But, it has a fundamental property that we will exploit to its fullest:

The intermediate numbers involved on the intermediate calculations never exceed 81

This is because it is the largest product possible of two 1-digit numbers ($9 \times 9 = 81$), and these numbers, well, we can deal with them easily!!

The main idea now is to find a suitable data structure to store all the intermediate results and for that we can use an array:

Say `int a[200]` is array where we can store 200 1-digit numbers. (In fact, at each position we can store an integer, but we will only store 1 digit for each position.)

So, we are making good progress!! We managed to understand two important things:

- Primary school arithmetic can prove very useful on big number problems;
- We can use all built-in structures and data-types to perform calculations;

Now, comes up a new question:

How can we know the length of such a huge number? We can store an array and 200 positions, but, our big number may have only 100 digits for example.

The trick is to use a variable that will save us, at each moment, the number of digits that is contained in the array. Let's call it `m`.

Also, since we want only one digit to be stored in every position of array, we need to find a way to "propagate" the carry of larger products to higher digits and sum it afterwards. Let's call the variable to hold the carry, `temp`.

1. `m` -> Variable that contains the number of digits in the array in any given moment;
2. `temp` -> Variable to hold the "carry" value resultant of multiplying digits whose product will be larger than 9. ($8 \times 9 = 72$, we would store 2 on one array position, and 7 would be the "carry" and it would be stored on a different position.)

So, now that we have an idea on how to deal with the multiplications, let's work on mapping it into a programming language.

- Coding our idea and one final detail

Now, we are ready to code our solution for the FCTRL2 problem.

However, one last remark needs to be done:

How do we store a number in the array, and why do we store it the way we do?

If after reading this tutorial you look at some of the accepted solutions in C/C++ for this problem, you will see that contestants actually stored the numbers "backwards", for example:

123 would be saved in an array, say `a`, as:

```
a = [3,2,1];
```

This is done such that when the digit by digit calculations are being performed, the "carry" can be placed on the positions of the array with higher index. This way, we are sure that carry is computed and placed correctly on the array.

Also, computing the products this way and maintaining the variable, `m`, allows us to print the result directly, by looping from `a[m-1]` until `a[0]`.

As an example, I can leave here an implementation made by [@upendra1234](#), that I took the liberty to comment for a better understanding:

```

#include<stdio.h>
int main()
{
    int t;

```

You are not logged in. Please login at www.codechef.com to post your questions!



```
scanf("%d",&t);
while(t-->0)
{
    scanf("%d",&n);
    a[0]=1; //initializes array with only 1 digit, the digit 1.
    m=1; // initializes digit counter

    temp = 0; //Initializes carry variable to 0.
    for(i=1;i<=n;i++)
    {
        for(j=0;j<m;j++)
        {
            x = a[j]*i+temp; //x contains the digit by digit product
            a[j]=x%10; //Contains the digit to store in position j
            temp = x/10; //Contains the carry value that will be stored on later indexes
        }
        while(temp>0) //while loop that will store the carry value on array.
        {
            a[m]=temp%10;
            temp = temp/10;
            m++; // increments digit counter
        }
    }

    for(i=m-1;i>=0;i--) //printing answer
        printf("%d",a[i]);
    printf("\n");
}
return 0;
}
```

I hope this tutorial can help someone to gain a better understanding of this subject and that can help some people as it is why we are here for :D

Best Regards,

Bruno Oliveira

EDIT: As per [@betlista](#) comment, it's also worth pointing out that, since we keep only a single digit at each position on the array, we could have used the data-type char instead of int. This is because internally, a char is actually an integer that only goes in the range 0 - 255 (values used to represent the ASCII codes for all the characters we are used to see). The gains would be only memory-wise.

This question is marked "community wiki".

asked 13 Mar '13, 21:12

[factorial](#) [bignum](#) [tutorial](#) [fctrl2](#)

edited 11 Nov '14, 20:28

3★ s1h33p
[329]•2•3•9



2★ kuruma
[17.6k]•72•143•209
accept rate: 8%

When digits are stored in the form of ascii code then digits 0-9 can't be treated as 0-9. Digits will be treated as 48-57. To use digit 1 as number 1 if it is stored as char then 48 should be subtracted from ascii value of 1 i.e 49. 49-48=1

[ritujain1971](#) (01 Aug '13, 12:23)

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
define m double
```

```
m fact(m n) { m i,ans=1; for(i=2;i<=n;i++) ans=ans*i; return ans; } int main() { m t,n; int i; scanf("%lf",&t); n=
(m)malloc(sizeof(m)t); for(i=0;i<t;i++) scanf("%lf",&n[i]); for(i=0;i<t;i++) printf("%n0.0lf",fact(n[i])); return 0; } getting
correct on compiler but gives wrong answer on codechef
```

2★ wasserkopf (07 Oct '16, 21:38)

25 Answers:

[oldest answers](#) [newest answers](#) [popular answers](#)

1 2 3 next »

22

Just a small tip (a got it too, I'm not the author), you do not need to have digits in your array ;-). If you want to use digits, using char array is more space efficient...

Let say you want to find the result of 98*76

```

      9 8
x    7 6
-----
      54 48
      63 56
-----
      63 110 48
      ~~~~~ (mod 10)
```

You are not logged in. Please login at www.codechef.com to post your questions!



74 63+11

=====

7 4 4 8

link

answered 13 Mar '13, 22:04



3★ betlista ♦♦
[16.9k]•49•115•225
accept rate: 11%

8 Yes, using a char array instead of an int array, since we are only storing digits and not numbers, would make more sense when talking about a memory efficient code.

On this case, I chose clarity over efficiency, as I believe that for a newbie that reads this tutorial, introducing the idea that a char is actually a very small int could be unnecessary complicated :)

2★ kuruma (14 Mar '13, 00:17)

And I took the liberty to fork the above code to find out very large powers of n. :D

6 //Code to store very large powers of 2.

```
#include<stdio.h>
int main()
{
    int t;
    int a[1000]; //array will have the capacity to store 1000 digits.
    int n,i,j,temp,m,x;

    scanf("%d",&t);
    while(t-->0)
    {
        scanf("%d",&n); // n is the power.
        a[0]=1; //initializes array with only 1 digit, the digit 1.
        m=1; // initializes digit counter
        i=2; // i is base 2
        int k=1; //k is a counter that goes from 1 to n.
        temp = 0; //Initializes carry variable to 0.
        while(k<=n)
        {
            for(j=0;j<m;j++)
            {
                x = a[j]*i+temp; //x contains the digit by digit product
                a[j]=x%10; //Contains the digit to store in position j
                temp = x/10; //Contains the carry value that will be stored on later indexes
            }
            while(temp>0) //while loop that will store the carry value on array.
            {
                a[m]=temp%10;
                temp = temp/10;
                m++; // increments digit counter
            }
            k++;
        }

        for(i=m-1;i>=0;i--) //printing answer
            printf("%d",a[i]);
        printf("\n");
    }

    return 0;
}
```

link

answered 14 Nov '13, 11:38



2★ saikatumardey
[136]•1•4•5
accept rate: 50%

Thanks a lot

2★ tushar22 (30 Oct '14, 10:46)

4 I have a better solution not just for finding factorial. Whenever the question involves computation with big numbers (yeah very big!) you can use a user defined data type **BIGINT** you need not do any code for it I'm posting it here(Even I found it somewhere :)) hope you find it useful. here's the code:

```
#include <iostream>

include <iomanip>
include <vector>
include <list>
include <string.h>
```

You are not logged in. Please login at www.codechef.com to post your questions!



```
using namespace std;

const int base = 1000000000; const int base_digits = 9;

struct bigint { vector<int> a; int sign;

    bigint() :
        sign(1) {
    }

    bigint(long long v) {
        *this = v;
    }

    bigint(const string &s) {
        read(s);
    }

    void operator=(const bigint &v) {
        sign = v.sign;
        a = v.a;
    }

    void operator=(long long v) {
        sign = 1;
        if (v < 0)
            sign = -1, v = -v;
        for (; v > 0; v = v / base)
            a.push_back(v % base);
    }

    bigint operator+(const bigint &v) const {
        if (sign == v.sign) {
            bigint res = v;

            for (int i = 0, carry = 0; i < (int) max(a.size(), v.a.size()) || carry; ++i) {
                if (i == (int) res.a.size())
                    res.a.push_back(0);
                res.a[i] += carry + (i < (int) a.size() ? a[i] : 0);
                carry = res.a[i] >= base;
                if (carry)
                    res.a[i] -= base;
            }
            return res;
        }
        return *this - (-v);
    }

    bigint operator-(const bigint &v) const {
        if (sign == v.sign) {
            if (abs() >= v.abs()) {
                bigint res = *this;
                for (int i = 0, carry = 0; i < (int) v.a.size() || carry; ++i) {
                    res.a[i] -= carry + (i < (int) v.a.size() ? v.a[i] : 0);
                    carry = res.a[i] < 0;
                    if (carry)
                        res.a[i] += base;
                }
                res.trim();
                return res;
            }
            return -(v - *this);
        }
        return *this + (-v);
    }

    void operator*=(int v) {
        if (v < 0)
            sign = -sign, v = -v;
        for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i) {
            if (i == (int) a.size())
                a.push_back(0);
            long long cur = a[i] * (long long) v + carry;
            carry = (int) (cur / base);
            a[i] = (int) (cur % base);
            //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"(cur), "c"(base));
        }
        trim();
    }
};
```



```

bigint operator*(int v) const {
    bigint res = *this;
    res *= v;
    return res;
}

friend pair<bigint, bigint> divmod(const bigint &a1, const bigint &b1) {
    int norm = base / (b1.a.back() + 1);
    bigint a = a1.abs() * norm;
    bigint b = b1.abs() * norm;
    bigint q, r;
    q.a.resize(a.a.size());

    for (int i = a.a.size() - 1; i >= 0; i--) {
        r *= base;
        r += a.a[i];
        int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
        int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() - 1];
        int d = ((long long) base * s1 + s2) / b.a.back();
        r -= b * d;
        while (r < 0)
            r += b, --d;
        q.a[i] = d;
    }

    q.sign = a1.sign * b1.sign;
    r.sign = a1.sign;
    q.trim();
    r.trim();
    return make_pair(q, r / norm);
}

bigint operator/(const bigint &v) const {
    return divmod(*this, v).first;
}

bigint operator%(const bigint &v) const {
    return divmod(*this, v).second;
}

void operator/=(int v) {
    if (v < 0)
        sign = -sign, v = -v;
    for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i) {
        long long cur = a[i] + rem * (long long) base;
        a[i] = (int) (cur / v);
        rem = (int) (cur % v);
    }
    trim();
}

bigint operator/(int v) const {
    bigint res = *this;
    res /= v;
    return res;
}

int operator%(int v) const {
    if (v < 0)
        v = -v;
    int m = 0;
    for (int i = a.size() - 1; i >= 0; --i)
        m = (a[i] + m * (long long) base) % v;
    return m * sign;
}

void operator+=(const bigint &v) {
    *this = *this + v;
}

void operator-=(const bigint &v) {
    *this = *this - v;
}

void operator*=(const bigint &v) {
    *this = *this * v;
}

void operator/=(const bigint &v) {

```

You are not logged in. Please login at www.codechef.com to post your questions!



```

bool operator<(const bigint &v) const {
    if (sign != v.sign)
        return sign < v.sign;
    if (a.size() != v.a.size())
        return a.size() * sign < v.a.size() * v.sign;
    for (int i = a.size() - 1; i >= 0; i--)
        if (a[i] != v.a[i])
            return a[i] * sign < v.a[i] * v.sign;
    return false;
}

bool operator>(const bigint &v) const {
    return v < *this;
}

bool operator<=(const bigint &v) const {
    return !(v < *this);
}

bool operator>=(const bigint &v) const {
    return !(*this < v);
}

bool operator==(const bigint &v) const {
    return !(*this < v) && !(v < *this);
}

bool operator!=(const bigint &v) const {
    return *this < v || v < *this;
}

void trim() {
    while (!a.empty() && !a.back())
        a.pop_back();
    if (a.empty())
        sign = 1;
}

bool isZero() const {
    return a.empty() || (a.size() == 1 && !a[0]);
}

bigint operator-() const {
    bigint res = *this;
    res.sign = -sign;
    return res;
}

bigint abs() const {
    bigint res = *this;
    res.sign *= res.sign;
    return res;
}

long long longValue() const {
    long long res = 0;
    for (int i = a.size() - 1; i >= 0; i--)
        res = res * base + a[i];
    return res * sign;
}

friend bigint gcd(const bigint &a, const bigint &b) {
    return b.isZero() ? a : gcd(b, a % b);
}

friend bigint lcm(const bigint &a, const bigint &b) {
    return a / gcd(a, b) * b;
}

void read(const string &s) {
    sign = 1;
    a.clear();
    int pos = 0;
    while (pos < (int) s.size() && (s[pos] == '-' || s[pos] == '+')) {
        if (s[pos] == '-')
            sign = -sign;
        ++pos;
    }
    for (int i = s.size() - 1; i >= pos; i -= base_digits) {
        int x = 0;

```

You are not logged in. Please login at www.codechef.com to post your questions!



```

        a.push_back(x);
    }
    trim();
}

friend istream& operator>>(istream &stream, bigint &v) {
    string s;
    stream >> s;
    v.read(s);
    return stream;
}

friend ostream& operator<<(ostream &stream, const bigint &v) {
    if (v.sign == -1)
        stream << '-';
    stream << (v.a.empty() ? 0 : v.a.back());
    for (int i = (int) v.a.size() - 2; i >= 0; --i)
        stream << setw(base_digits) << setfill('0') << v.a[i];
    return stream;
}

static vector<int> convert_base(const vector<int> &a, int old_digits, int new_digits) {
    vector<long long> p(max(old_digits, new_digits) + 1);
    p[0] = 1;
    for (int i = 1; i < (int) p.size(); i++)
        p[i] = p[i - 1] * 10;
    vector<int> res;
    long long cur = 0;
    int cur_digits = 0;
    for (int i = 0; i < (int) a.size(); i++) {
        cur += a[i] * p[cur_digits];
        cur_digits += old_digits;
        while (cur_digits >= new_digits) {
            res.push_back(int(cur % p[new_digits]));
            cur /= p[new_digits];
            cur_digits -= new_digits;
        }
    }
    res.push_back((int) cur);
    while (!res.empty() && !res.back())
        res.pop_back();
    return res;
}

typedef vector<long long> vll;

static vll karatsubaMultiply(const vll &a, const vll &b) {
    int n = a.size();
    vll res(n + n);
    if (n <= 32) {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                res[i + j] += a[i] * b[j];
        return res;
    }

    int k = n >> 1;
    vll a1(a.begin(), a.begin() + k);
    vll a2(a.begin() + k, a.end());
    vll b1(b.begin(), b.begin() + k);
    vll b2(b.begin() + k, b.end());

    vll a1b1 = karatsubaMultiply(a1, b1);
    vll a2b2 = karatsubaMultiply(a2, b2);

    for (int i = 0; i < k; i++)
        a2[i] += a1[i];
    for (int i = 0; i < k; i++)
        b2[i] += b1[i];

    vll r = karatsubaMultiply(a2, b2);
    for (int i = 0; i < (int) a1b1.size(); i++)
        r[i] -= a1b1[i];
    for (int i = 0; i < (int) a2b2.size(); i++)
        r[i] -= a2b2[i];
}

```




You are not logged in. Please login at www.codechef.com to post your questions!

```

    for (int i = 0; i < (int) a1b1.size(); i++)
        res[i] += a1b1[i];
    for (int i = 0; i < (int) a2b2.size(); i++)
        res[i + n] += a2b2[i];
    return res;
}

bigint operator*(const bigint &v) const {
    vector<int> a6 = convert_base(this->a, base_digits, 6);
    vector<int> b6 = convert_base(v.a, base_digits, 6);
    vll a(a6.begin(), a6.end());
    vll b(b6.begin(), b6.end());
    while (a.size() < b.size())
        a.push_back(0);
    while (b.size() < a.size())
        b.push_back(0);
    while (a.size() & (a.size() - 1))
        a.push_back(0), b.push_back(0);
    vll c = karatsubaMultiply(a, b);
    bigint res;
    res.sign = sign * v.sign;
    for (int i = 0, carry = 0; i < (int) c.size(); i++) {
        long long cur = c[i] + carry;
        res.a.push_back((int) (cur % 1000000));
        carry = (int) (cur / 1000000);
    }
    res.a = convert_base(res.a, 6, base_digits);
    res.trim();
    return res;
}
};

int main() { int t; cin>>t; bigint fact=1; for(int i=1;i<=t;i++) fact*=i; cout<<fact<<endl; return 0; }

```

link

answered 30 Oct '14, 13:48



2★ anichavan20
[93]•1•4
accept rate: 0%

1 source - anudeep's blog.

4★ pranjalaranjan (21 Nov '14, 11:44)

What about last ten non zero digit of 10^{14} factorial ??

2

link

answered 17 Aug '13, 17:46



2★ may3
[53]•1•3•6
accept rate: 0%

very very helpful post.

2

--Thanks

link

answered 14 Jul '14, 02:08



cftc
[26]•2
accept rate: 0%

A lot of...lot of thanks to u :)

2

link

answered 13 Aug '14, 17:37



3★ shubham201
[51]•3
accept rate: 8%

Many many thanks 2 @codechef

2

link

answered 02 Sep '14, 12:42



1★ ganesluit
[26]•1•2
accept rate: 0%

Hell yeah!!! ..I use python and VOILA!!!!!!

1

link

answered 30 Sep '14, 00:22



3★ unlucy7735
[30]
accept rate: 0%

You are not logged in. Please login at www.codechef.com to post your questions!



0

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace BigMultiplier
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] s1 = new int[1];
            int[] s2 = new int[1];

            s1[0]=1;
            Program p = new Program();

            int[] s3 = p.doit(s1, s2);
            Console.WriteLine("Enter the Number for Which Factorial to be Found (below
1000)");
            int limit = Convert.ToInt32(Console.ReadLine());

            for (int i = 1; i <= limit; i++)
            {
                s3 = p.return_array(i);
                s1 = p.doit(s1, s3);
            }
            int sum = 0;
            for (int j = s1.Length-1; j >=0; j--)
            {
                Console.Write(s1[j]);
                sum = sum + s1[j];
            }
            Console.Write("sum = "+sum);
            Console.WriteLine();
            Console.ReadLine();
        }

        int[] return_array(int num)
        {
            int[] num_arr = new int[1]; ;

            if (num <= 9)
            {
                num_arr = new int[1];
                num_arr[0] = num;
            }
            else if (num <= 99)
            {
                num_arr = new int[2];
                num_arr[1] = num % 10;
                num = num / 10;
                num_arr[0] = num;
            }
            else if (num <= 999)
            {
                num_arr = new int[3];
                num_arr[2] = num % 10;
                num = num / 10;
                num_arr[1] = num % 10;
                num = num / 10;
                num_arr[0] = num % 10;
            }
            else if (num <= 9999)
            {
                num_arr = new int[4];
                num_arr[3] = num % 10;
                num = num / 10;
                num_arr[2] = num % 10;
                num = num / 10;
                num_arr[1] = num % 10;
                num = num / 10;
                num_arr[0] = num % 10;
            }
        }
    }
}
```

You are not logged in. Please login at www.codechef.com to post your questions!



```

    {
        num_arr = new int[5];
        num_arr[4] = num % 10;
        num = num / 10;
        num_arr[3] = num % 10;
        num = num / 10;
        num_arr[2] = num % 10;
        num = num / 10;
        num_arr[1] = num % 10;
        num = num / 10;
        num_arr[0] = num % 10;
    }
    return num_arr;
}

int[] doit(int[] num1_int, int[] num2_int)
{
    //      String num1_string, num2_string;
    //      Console.WriteLine("Enter the Number 1");
    //      num1_string = Console.ReadLine();
    //      Console.WriteLine("Enter the Number 2");
    //      num2_string = Console.ReadLine();

    //      int[] num1_int = new int[num1_string.Length];
    //      int[] num2_int = new int[num2_string.Length];

    //      for (int j = 0; j < num1_string.Length; j++)
    //      {
    //          num1_int[j] = num1_string[j]-48;
    //          Console.Write(" " + num1_int[j]);
    //      }

    //      for(int j=0;j<num2_string.Length;j++)
    //      {
    //          num2_int[j] = num2_string[j]-48;
    //          Console.Write(" " + num2_int[j]);
    //      }

    int[, ] num3_int = new int[num2_int.Length, (num1_int.Length + 1)];
    int i,k,temp=0;

    //Multiplication on Individual Digits Done and the Values are there in the
    Individual Cells of the Array
    for(i=0;i<num2_int.Length;i++)
    {
        for (k = 0; k < num1_int.Length; k++)
        {
            int mul = (num1_int[k] * num2_int[i])+temp;
            num3_int[i, k] = mul % 10;
            temp = mul / 10;
            if (k == (num1_int.Length - 1))
            {
                num3_int[i, k+1] = temp;
            }
            //          Console.Write(" " + num3_int[i, k]);
        }
        //          Console.Write(" " + num3_int[i, k]);
        temp=0;
        //          Console.WriteLine();
    }

    //      temp = 0;

    //      Console.ReadLine();
    int[] result_int = new int[num1_int.Length + num2_int.Length];

    //      int[, ] num3_int = new int[num2_string.Length, (num1_string.Length + 1)];

    double result=0;

    for(i=0;i<num2_int.Length;i++)
        for (k = 0; k < (num1_int.Length + 1); k++)
        {
            //          Console.Write(" i = " + i + " k=" + k+" ");
            //          Console.Write(num3_int[i, k]);

```



You are not logged in. Please login at www.codechef.com to post your questions!

```

// Console.WriteLine("10^k = " + Math.Pow(10,k));

(num3_int[i,k]*Math.Pow(10,k)*Math.Pow(10,i));
// Console.WriteLine("10^k = " + Math.Pow(10,k));

    }

    int[,] num4_int= new int[num2_int.Length,num1_int.Length + num2_int.Length];
    for (i = 0; i < num2_int.Length; i++)
    {
        for (k = 0; k < (num1_int.Length + 1); k++)
        {
            //for (int l = 0; l < 0; l++)
            {
                num4_int[i,k + i] = num3_int[i,k];
            }
        }
    }

    for (i = 0; i < num2_int.Length; i++)
    {
        for (k = 0; k < (num1_int.Length + num2_int.Length); k++)
        {
            Console.Write(" " + num4_int[i, k]);
        }
        Console.WriteLine();
    }

    int[] re_int = new int[num1_int.Length + num2_int.Length];
    temp=0;
    for (i = 0; i < re_int.Length; i++)
    {
        re_int[i] = 0;
        for (k = 0; k < num2_int.Length; k++)
        {
            re_int[i] = num4_int[k, i] + re_int[i];
        }
        int t = re_int[i] + temp;
        re_int[i]=t%10;
        temp=t/10;
        if(i==(re_int.Length-1))
        {
            //Need to check
        }
    }

    Console.WriteLine("Final Result - Reversed Order ");
    for(i=0;i<re_int.Length;i++)
    Console.Write(" "+re_int[i]);

    re_int[i+k]=;

    Console.WriteLine("Final Result - Correct Order ");
    for(i=re_int.Length-1;i>=0;i--)
    Console.Write(" " + re_int[i]);

    Console.WriteLine(result);
    Console.ReadLine();
    return re_int;
}
}
}

```

link

edited 13 Mar '14, 18:36

3★ betlista ♦♦
[16.9k] ●49●115●225

answered 29 Mar '13, 11:01

gopaltirupur
[30]●1
accept rate: 0%

1 that return_array method is really ugly!!!

3★ betlista ♦♦ (13 Mar '14, 18:37)

THANK YOU SO MUCH TO DESCRIBE ME

0

link

answered 13 Dec '13, 01:26

guptanitin
[29]
accept rate: 0%

You are not logged in. Please login at www.codechef.com to post your questions!



[hide preview]

☐ community wiki:

Preview

reCAPTCHA V1 IS SHUTDOWN
Direct site owners to g.co/recaptcha/upgrade

Type the text



Post Your Answer