# "EFFECTIVE USE OF PROGRAMMING LANGUAGES"

**AKULA .V.S. SIVA RAMA RAO, M.Tech(CSE).,M.Phil(CS), Associate Professor**
**KANAKAM SIVA RAM PRASAD.,Asst. Prof.**
**L.V.SAMBA SIVA RAO Asst. Prof.**

**Sasi Institute of Technology and Engineering**
**TADEPALLIGUDEM-534101, W.G.Dist(AP)**

---

**Problem Statement:** The objective of the case study is to dispel the limitations myths of students about the Programming languages by "**EFFECTIVE USE OF PROGRAMMING LANGUAGES**".



Myths about programming Languages:

I would like to explain "**EFFECTIVE USE OF PROGRAMMING LANGUAGES**" with an example of finding 'Factorial' for given large number.

For example, the factorial for the numbers 20, 100, 1000, we expect more then 15, 150, 1500 digits numbers respectively (of course those are 19, 158, and 2,568 Digits numbers respectively).

How can we store this much large number in a variable of any 'data type' of any language ?

- ✓ 'C' language Supports 'Unsigned long int' data type of 32 bits variable, which can store upto 4,294,967,295 integer number(maximum of 10 digits) only.

- ✓ Java language Supports 'long' data type of 64 bits variable and it can store upto 9,223,372,036,854,775,807 integer number(maximum of 19 digits) only.

**Scenario :**

In many situations of our real life, there is a need to represent more than '19 digits' numbers.

Examples :

- ✓ Combinations and Permutations problems require to calculate factorial of 21 or above (ie In case Java 20!= 2432902008176640000 which is 19 digits number, 21! is 51090942171709440000 which is 20 digits number.
  So students have myth that C, C++ or Java etc. can't find factorial for '21').
- ✓ Finding sum of Deposits or loans of all braches of Andhra Bank.
- ✓ Total salaries of State or Union Government employees
- ✓ Government or Private Organizations' Financial Budgets and Balance Sheets.
- ✓ Stock market data analysis
- ✓ Calculations regarding the population of the world.
- ✓ Scientific and R & D organizations require to use very large numbers.

In that situation shall we blame developers of 'C', 'C++' or 'Java' because of above misconception ?

"Blaming them is blaming us".

In such situation "**The Empowerment of Software**" comes into picture !

**Solution :** The following program illustrates how to "Empower Your Software" by finding 1000!

Source Code :

```
/*****************************Finding factorial for given integer*************************************
                    It finds Factorial value upto 30,000 digits number
*****************************************************************************************************/
#include<stdio.h>
#include<conio.h>
#define size 30000
void main()
{
//Declaration begin
unsigned int un_int_n,un_int_k;
// To store single digit in array 'unsigned short int' takes only one byte
unsigned short int a[size];
int int_rem,int_pos=size-1,int_p=size-1,int_i,int_j;
//Declaration ends
//Initialization of array with zeros
for(int_i=0;int_i<size;int_i++)
a[int_i]=0;
clrscr();
// Reading number for which factorial to be found
printf("\nEnter an integer to find factorial ");
scanf("%d",&un_int_n);
int_i=1;
//Initially store 1 in right side of array
a[size-1]=1;
// Each iteration find product, and loop will execute for n times ie 1 X 2 X 3....n
        while(int_i<=un_int_n)
        {
          int_j=size-1;
          //multiply j location value  with iterative value
```

```
        un_int_k=a[int_j]*int_i;



        //repeats loop until product k is >0, where p is current position of most significant digit
            while(un_int_k>0 || int_p<=int_j)
            {
                //extracting last digit of k
                int_rem=un_int_k%10;
                a[int_j]=int_rem;
                //decrement the position p
                int_pos=--int_j;
                //eleminating last digit
                un_int_k=un_int_k/10;
                //adding quotient to next digit
                un_int_k=a[int_j]*int_i+un_int_k;
            }
        // store position in p
        int_p=int_pos;
        //increment the iterative value
        int_i++;
  }
//diplay the factorial digit by digit from the array
printf("\nFactorial %d is\n",un_int_n);
for(int_i=int_p+1;int_i<size;int_i++)
printf("%d",a[int_i]);
getch();
}
```

**Result : The above program produced following 2,568 Digits number as 1000!**

Enter an integer to find factorial 1000

Factorial 1000 is

402387260077093773543702433923003985719374864210714632543799910429938
512398629020592044208486969404800479988610197196058631666872994808558
901323829669944590997424504087073759918823627727188732519779505950995
276120874975462497043601418278094646496291056393887437886487337119181
045825783647849977012476632889835955735432513185323958463075557409114
262417474349347553428646576611667797396668820291207379143853719588249
808126867838374559731746136085379534524221586593201928090878297308431
392844403281231558611036976801357304216168747609675871348312025478589
320767169132448426236131412508780208000261683151027341827977704784635
868170164365024153691398281264810213092761244896359928705114964975419
909342221566832572080821333186116811553615836546984046708975602900950
537616475847728421889679646244945160765353408198901385442487984959953
319101723355556602139450399736280750137837615307127761926849034352625
200015888535147331611702103968175921510907788019393178114194545257223
865541461062892187960223838971476088506276862967146674697562911234082
439208160153780889893964518263243671616762179168909779911903754031274
622289988005195444414282012187361745992642956581746628302955570299024
324153181617210465832036786906117260158783520751516284225540265170483
304226143974286933061690897968482590125458327168226458066526769958652
682272807075781391858178889652208164348344825993266043367660176999612
831860788386150279465955131156552036093988180612138558600301435694527
224206344631797460594682573103790084024432438465657245014402821885252
470935190620929023136493273497565513958720559654228749774011413346962
715422845862377387538230483865688976461927383814900140767310446640259
899490222221765904339901886018566526485061799702356193897017860040811

889729918311021171229845901641921068884387121855646124960798722908519
296819372388642614839657382291123125024186649353143970137428531926649
875337218940694281434118520158014123344828015051399694290153483077644
569099073152433278288269864602789864321139083506217095002597389863554
277196742822248757586765752344220207573630569498825087968928162753848
863396909959826280956121450994871701244516461260379029309120889086942
028510640182154399457156805941872748998094254742173582401063677404595
741785160829230135358081840096996372524230560855903700624271243416909
004153690105933983835777939410970027753472000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000



**Logic behind the program :**

**Initial array :**

| **0** | **1** | **.....29,999** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**1st iteration : a[29999] X 1 = 1**

| **0** | **1** | **.....29,999** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | .... | .... | .... | .... | .... | 1 |

**2nd iteration : a[29999] X 2 = 2**

| **0** | **1** | **2.....29,999** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | .... | .... | .... | .... | .... | 2 |

**3rd iteration : a[29999] X 3 = 6**

| **0** | **1** | **2.....29,999** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | **....** | **....** | **....** | **....** | **....** | 6 |

**4th iteration : a[29999] X 4 =24, store 4 in a[29999] and add carry 2 to the previous location ie 29998**



| **0** | **1** | **..... 29,998** | **29,999** | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | **....** | **....** | **....** | **....** | 2 | 4 |

**5th iteration : a[29999] X 5 = 20, store 0 in 29999 and a[29998] X 5 + 2 carry = 12, store 2 in 29,998, add carry 1 to location 29,997**

| **0** | **1** | **2..... 29,997** | **29,998** | **29,999** | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | .... | .... | .... | 1 | 2 | 0 |

**6th iteration : a[29999] X 6 = 0, store 0 in 29999, a[29998] X 6 = 12, store 2 in 29998th location, a[29997] X 6 = 6 and add carry 1 to the location 29997**

| 0 | 1 | ..... | 29,997 | 29,998 | 29,999 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 .... .... .... 7 | 2 | 0 |

.
.
.

**1000th iteration : The above produced each digit of factorial value will be stored in one location of array ie from 27,432th location to 29,999**

**After 1000th iteration array will look like this :**

| 0 | 1 | 2 ..... | 27,432 | 27,433 | 27,434 ..... | 29,998 | 29,999 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 ..... | 4 | 0 | 2 .... | 0 | 0 |

▲ Starting of Factorial Digit

▲ End of Factorial Digit



**Conclusion :** By using this simple program, I wish to share my idea, ie how to **"Empower the Software".** This program can find up to 30,000 (Thirty Thousand) Digits factorial value and this program can be implemented more efficiently by using Dynamic Data Structure(Linked Lists).

**Learning Outcomes :** Students can come out from the myths of limitations about Programming Languages and **"Enrich their Software Products".**