Name : IMON RAJ

Roll: 002010501098

Class: BCSE II

Sem: First

Session: 2021-2022

**ASSIGNMENT SET 1**

***Problem no.-*** *1*

**Problem statement:** Write a program to compute the factorial of an integer n iteratively and recursively. Check when there is overflow in the result and change the data types for accommodating higher values of inputs.

**Solution Approach:**

Factorial of a number n is given by 1.2.3….n and we have found out factorial of integers by using iterative method in a function and recursive manner in another one. After making and compiling the program we are running it with increasing value of integers. In the program we at first taken int as data type to evaluate factorial. Then we have changed it whenever needed.

**PseudoCode:**

RECURSIVE METHOD:

If n==0 or n==1 then:

Return 1

Else :

Return n\*RECURSIVE\_METHOD(n-1)

ITERATIVE METHOD:

If n==0 then:

Return 1

Fact = 1

For i from 1 to equals to n:

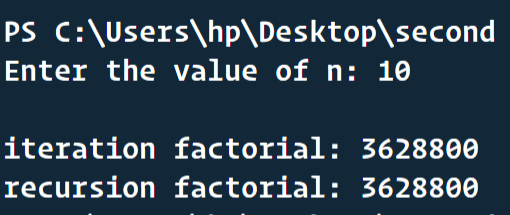
Fact = fact\*i

Return fact

**Results:**

After running the program, we found that: --

* In case of int overflow occurs after n=16 and we change data type to long
* In case of long overflow occurs after n=16 and we change data type to long long
* In case of long long overflow occurs after n=20



**Discussion:**

This program helps us to understand the range of the built in data types int, long and long long. Whenever the answer of factorial surpass the range of a particular data type, the overflow occurs.

***Problem no.- 2***

**Problem statement**: Write a program to generate the nth Fibonacci number iteratively and recursively. Check when there is

overflow in the result and change the data types for accommodating higher values of inputs. Plot the

Fibonacci number vs n graph.

**Solution Approach:**

Fibonacci sequence is a special type of sequence where each element is calculated by adding the previous two elements. We have written program to find out nth Fibonacci number in iterative and recursive method. We will check where the overflow occurs for various data types by increasing the value of n.

**PseudoCode:**

RECURSIVE METHOD:

If n==1 or n==2 then:

Return 1

Else:

Return RECURSIVE\_FIB(n-1)+ RECURSIVE\_FIB(n-2)

ITERATIVE METHOD:

If n==1 or n==2 then:

Return 1

prev = 1

before = 1

Sum = 0

For i from 3 to equals to n:

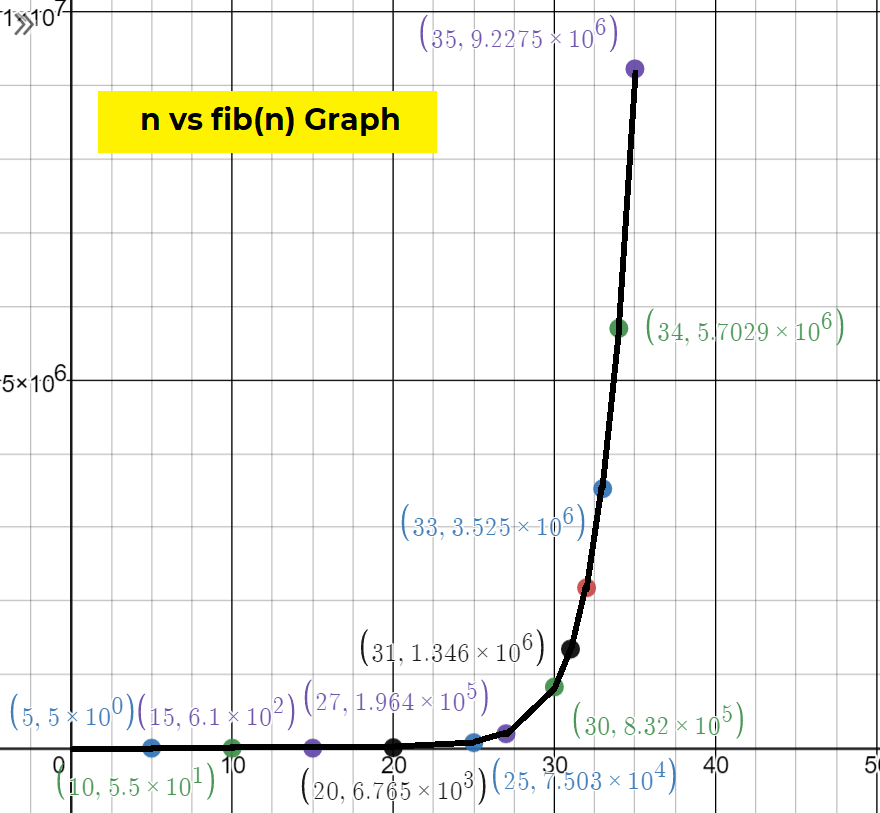
Sum = prev+before

before = prev

prev = Sum

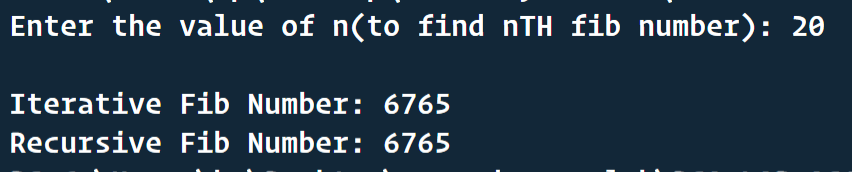
Return Sum

**Results:**

****

After running the program, we found that: --

* In case of int overflow occurs after n=46 and we change data type to long
* In case of long overflow occurs after n=46 and we change data type to long long
* In case of long long overflow occurs after n=92



**Discussion:**

This program helps us to understand the range of the built in data types int, long and long long. Whenever the answer of fibonacci surpass the range of a particular data type, the overflow occurs.

***Problem no.- 3***

**Problem statement:**  Write programs for linear search and binary search for searching integers, floating point numbers and words in arrays of respective types.

**Solution Approach:**

We will find for the key in the array in different ways on the basis of type of array (ordered and unordered). In linear search, we will go for each element and check whether it is equal to the key and return index if found and -1 if not found. This will be in case of unsorted array. In sorted array we will use binary search and we will half the array in each iteration on the basis of key value comparing with the mid-value of array.

**PseudoCode:**

Given: key , arr[]

LINEAR SEARCH:

For all arr[] elements with index i:

If arr[i]==key then:

Return i

Return -1

BINARY SEARCH:

midindex = leftindex + rightindex

if key<arr[leftindex] or key>arr[rightindex]:

return -1

do

{

if(key == arr[midindex]){

return midindex

} else if (key > arr[midindex]) {

leftindex = midindex

} else {

rightindex = midindex

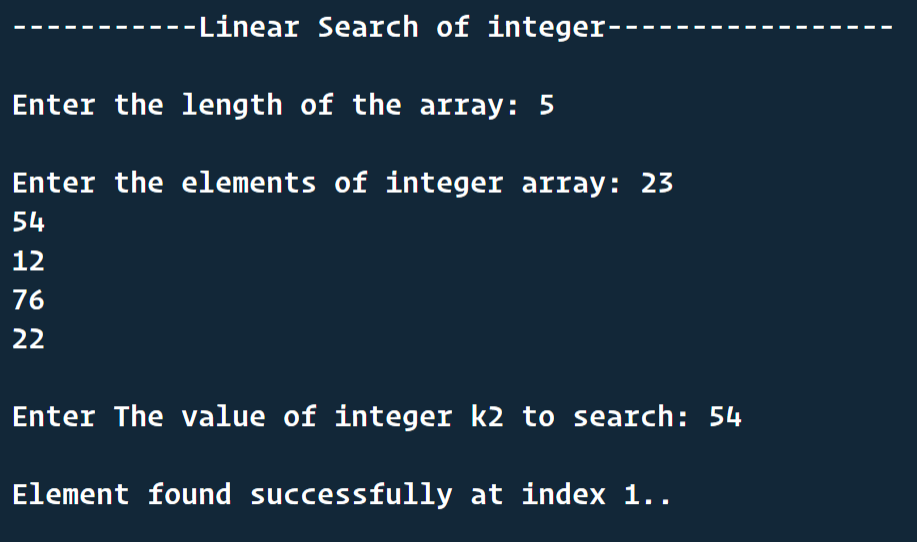
}

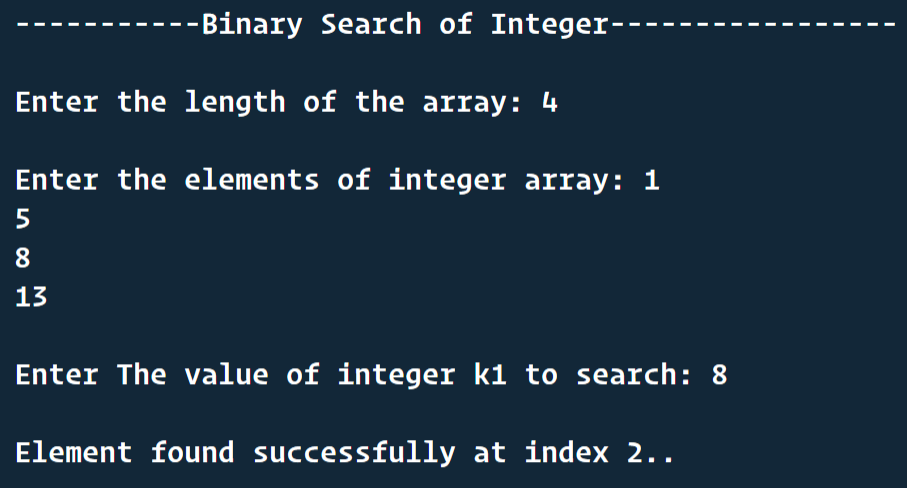
midindex = (leftindex+rightindex)/2

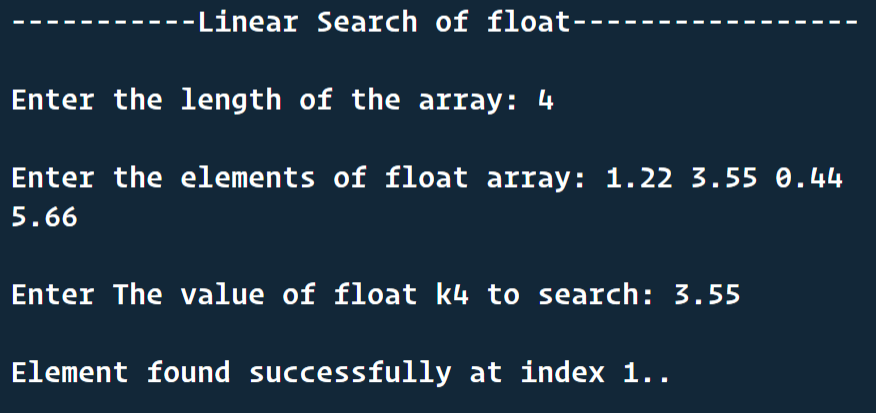
} while (midindex != leftindex)

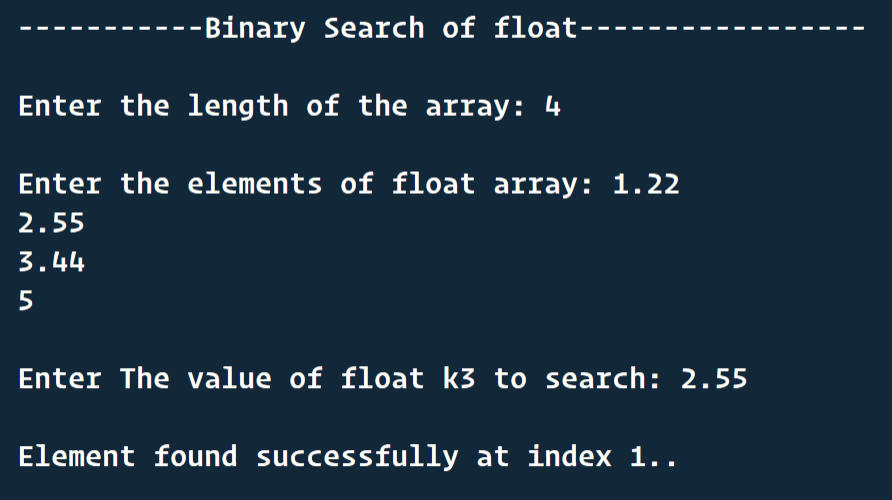
Return -1

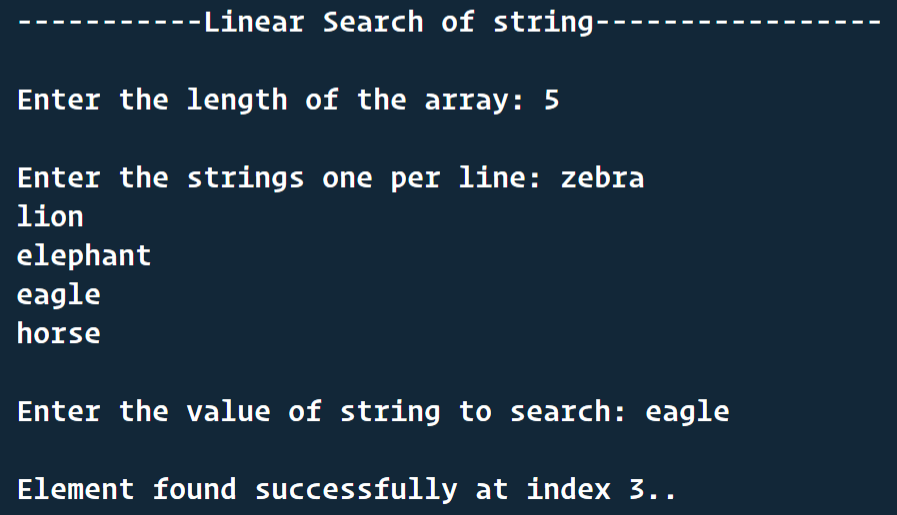
**Results:**

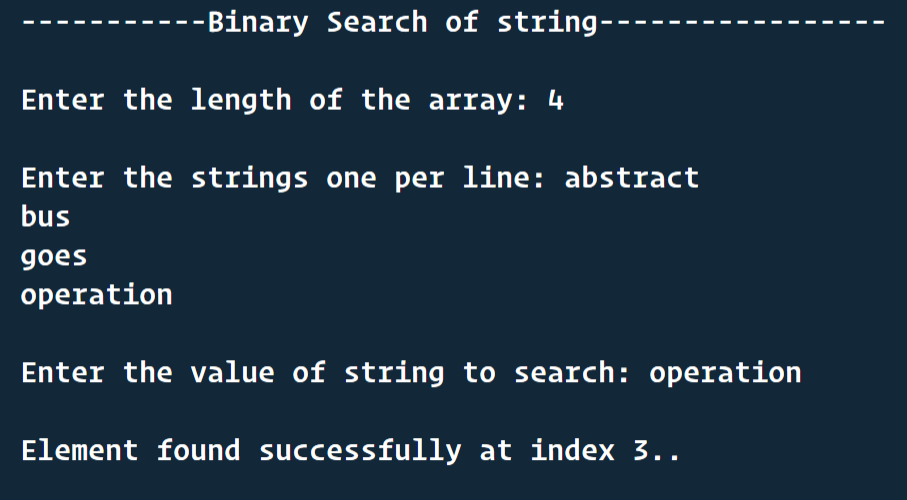












**Discussion:**

In case of binary search the array must be sorted and it is lot more faster than linear search and linear search doesn’t need any criteria, the array may be unsorted and it checks each element of array. These two search methods are applied in case of float, int and string arrays.

***Problem no.- 4***

**Problem statement:**  Write a program to generate 1,00,000 random integers between 1 and 1,00,000 without repetitions and

store them in a file in character mode one number per line. Study and use the functions in C related to

random numbers.

**Solution Approach:**

Here, we will have an integer array of each element from 1 to 100K. We will shuffle this array by generating random index values and finally the shuffled array will be printed in the file. To generate random integers we will use rand() function of C language.

**PseudoCode:**

Random\_index function:

frac1 = rand()/32768.000

frac2 = rand()/32768.000

val = (frac1\*frac2)\*99999

index = floor(val)

return index

Shuffle\_array(arr[]) function:

For k from 0 until(not equal) 100000:

i = Random\_index()

j = Random\_index()

swap(arr[i], arr[j])

For k from 0 until(not equal) 100000:

i = Random\_index()

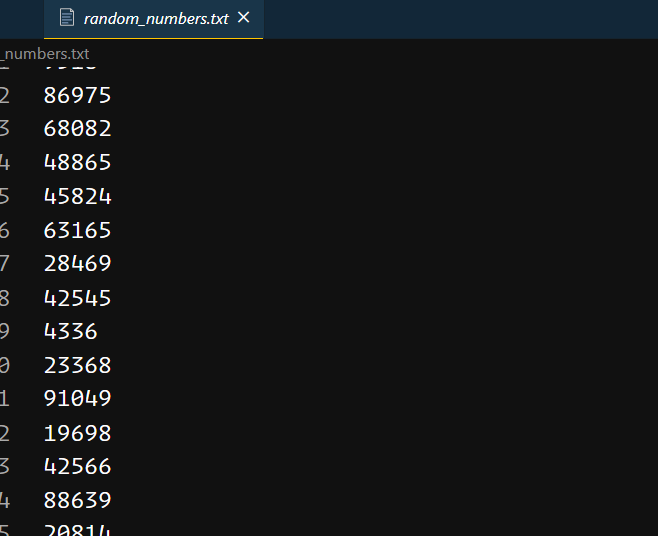
j = Random\_index()

swap(arr[99999-i], arr[99999-j])

Return arr[]

**Results:**

We have used two for loops to make the shuffle more random, one focusing elements in start side and another in last side. Finally we can see the complete random numbers in the txt file.



**Discussion:**

To randomize we have depended on the rand() function but in reality nothing can be completely random generated by a computer. They are actually pseudorandom values. In generation of pseudo random values a seed is needed . We have passed the current time as seed and after that rand() function is used randomize values.

***Problem no.- 5***

**Problem statement:**  Write a program to generate 1,00,000 random strings of capital letters of length 10 each, without

repetitions and store them in a file in character mode one string per line.

**Solution Approach:**

We know the ASCII values of A to Z is 65 to 90. 26 alphabets are there. So, we have used rand() function and done modulo operation with 26 and this will generate values between 0 to 26 and adding random value with 65 gives us 65-90 random values. Using loops and random ASCII values we have generated 100K random strings and stored them in text file.

**PseudoCode:**

For i from 0 to 100000(not include):

For j from 0 to 10(not include):

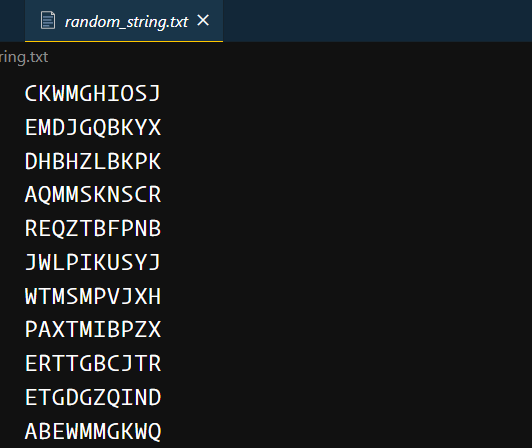
C = to\_character(65+rand()%26) // randomly generates a letter from A to Z

Print\_into\_file(C)

Print\_into\_file(‘\n’)

**Results:**

After the program runs, we can see the text file that 100K random 10-length strings have been generated one per line.



**Discussion:**

To randomize we have depended on the rand() function but in reality nothing can be completely random generated by a computer. They are actually pseudorandom values. In generation of pseudo random values a seed is needed . We have passed the current time as seed and after that rand() function is used randomize values.

***Problem no.- 6***

**Problem statement:**  Store the names of your classmates according to roll numbers in a text file one name per line. Write a program to find out from the file, the smallest and largest names and their lengths in number of characters. Write a function to sort the names alphabetically and store in a second file.

**Solution Approach:**

In this problem a lager name means that its lexicographical value is greater. We will use the string library functions to compare two names. For finding the largest and smallest name we will consider the first name of the list to be largest as well as smallest and compare it to every name and update the value if needed and finally we will get the largest and smallest name.

For sorting the name alphabetically we will use bubble sort method. The all name comparisons will be based on STRCMP function.

Name1>name2 🡺 strcmp(name1, name2) > 0

**PseudoCode:**

FINDING LARGEST AND SMALLEST NAME:

**//let’s say we are considering all the names of the text file as name\_array[]**

largest\_name = name\_array[0]

smallest\_name = name\_array[0]

for all names in **name\_array[]** with index **i**:

if (**name\_array[i] > largest\_name**) then:

largest\_name = name\_array[i]

else if (**name\_array[i] < smallest\_name**) then:

smallest\_name = name\_array[i]

length\_of\_largest\_name = strlen(largest\_name)

length\_of\_smallest \_name = strlen(smallest\_name)

SORTING THE NAMES ALPHABETICALLY:

total\_number\_of\_names = length(name\_array)

for i from 0 to total\_number\_of\_names:

for j from i+1 to total\_number\_of\_names:

if(strcmp(name\_array[i], name\_array[j]) > 0) then:

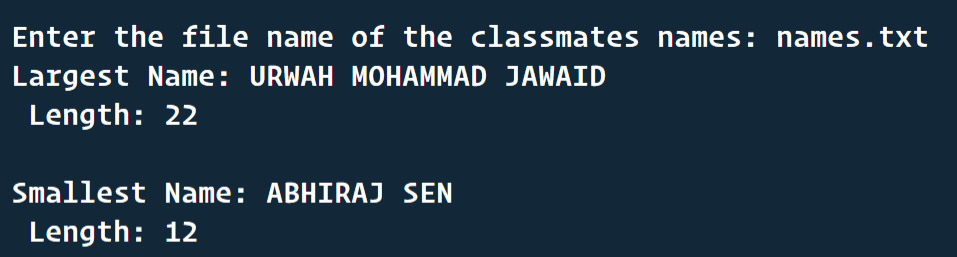
strcpy(temp, name\_array[i]);

strcpy(name\_array[i], name\_array[j]);

strcpy(name\_array[j], temp)

**Results:**

After the program runs, we can see the largest name to be URWAH MOHAMMAD JAWAID and smallest to be ABHIRAJ SEN. And the sorted.txt file stores the names in alphabetical order.





**Discussion:**

This problem and program help us to understand sorting technique and string library functions’ use in comparing two strings and also file handling process.

***Problem no.- 7***

**Problem statement:**  Take a four-digit prime number P. Generate a series of large integers L and for each member Li compute the remainder Ri after dividing Li by P. Tabulate Li and Ri. Repeat for seven other four digit prime numbers keeping Li fixed.

**Solution Approach:**

In this problem , I will take the prime numbers as fixed values. But using the random generator function I will generate the large integers and

Then tabulate the large integers and primes with the remainder value as R = L%P.

**PseudoCode:**

FOR THE GENERATION OF LARGE INTEGER:

**long long large = (rand() % 9)+1;** // To maintain the large integer length first digit should be 1 to 9

**for (int i = 0; i < 11; i++)**

**{**

**large = large \* 10 + (rand() % 10);** // generating digits 0-9 and appending to large int

**}**

**return large;**

FOR THE TABULATION

**int numberOfLargeInts = 5;**

**long long remainder, large;**

**for (i = 0; i < numberOfLargeInts; i++)**

**{**

**large = generateLargeInt();** // FUNCTION TO GENERATE LARGE INTEGER MENTIONED BEFORE

**print(large);**

**for (j = 0; j < 8; j++)**

**{**

**remainder = large % primes[j];** // Finding the remainder

**print(remainder);**

**}**

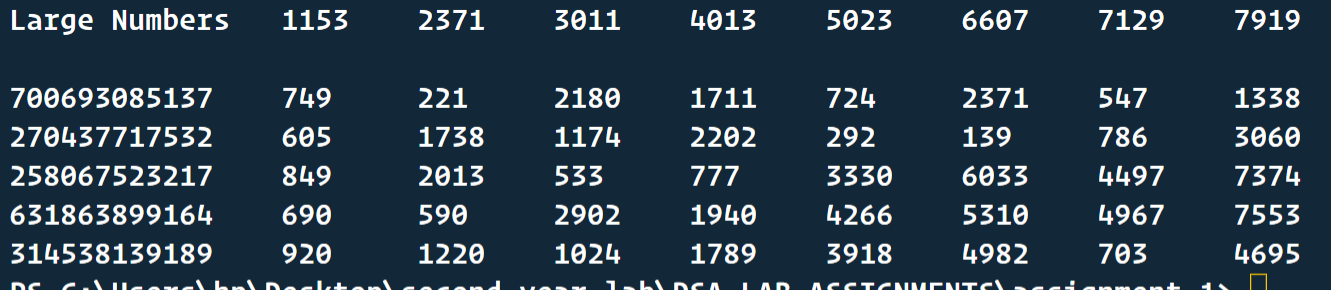
**print(NEW\_LINE);**

**large = 0;**

**}**

**Results:**

After program execution we can see a tabulated matrix of remainder values for various large integers and prime numbers.



**Discussion:**

This problem and program help us to understand random integer generation of a particular length by putting no zero in the first digit.

***Problem no.- 8***

**Problem statement:**  Convert your Name and Surname into large integers by juxtaposing integer ASCII codes for alphabet.

Print the corresponding converted integer. Cut the large integers into two halves and add the two halves.

Compute the remainder after dividing the by the prime numbers P in problem 7.

**Solution Approach:**

We will take our name as input in a string then taking each character of name and storing it into a number’s array. On the basis of the number(digit) array length I will divide my name’s converted long int value into two parts and add them digitwise. Finally, we will divide the result with the primes of the previous question and see the remainder.

**C Code:**

#include <stdio.h>

#include <string.h>

#include <math.h>

#define Max 30

int valid(char s[])

{

int i, size = strlen(s);

for (i = 0; i < size; i++)

{

if (!((s[i] >= 65 && s[i] <= 90) || s[i] == 32))

return 0;

}

return 1;

}

int main()

{

char name[Max];

int nm[Max];

int sum[Max];

char \*ptr;

int j, i, x, k = 0;

ptr = name;

do

{

printf("Type your name(In capital Letter): \n");

scanf("%[^\n]s", name);

if (!valid(name))

printf("Enter your name only in capital letter.\n");

else

break;

} while (1);

for (i = 0; \*ptr != '\0'; ptr++, i++)

{

if (name[i] == 32)

continue;

else

{

x = name[i];

nm[k++] = x / 10;

nm[k++] = x % 10;

}

}

nm[k] = '\0';

printf("Converted long integer is : ");

for (j = 0; j < k; j++)

{

printf("%d", nm[j]);

}

printf("\n");

printf("Long integer is now divided into two integer:\n");

printf("First Half: ");

for (i = 0; i < k / 2; i++)

{

printf("%d", nm[i]);

}

printf("\nAnd second Half: ");

for (i = k / 2; i < k; i++)

{

printf("%d", nm[i]);

}

printf("\nAnd summed so that Result after Summation is :\n");

int p, r = 0;

for (i = k / 2 - 1, j = k - 1, p = 0; i >= 0; i--, j--)

{

sum[p++] = (nm[i] + nm[j] + r) % 10;

r = (nm[i] + nm[j] + r) / 10;

}

sum[p++] = r;

sum[p] = '\0';

long int y = 0;

for (i = 0; i < p; i++)

{

y += sum[i] \* pow(10, i);

}

printf("%d\n", y);

int primes[8] = {1153,

2371,

3011,

4013,

5023,

6607,

7129,

7919};

for (i = 0; i < 8; i++)

{

long int d = y % primes[i];

printf("\nIf it is divided by Prime %d, ", primes[i]);

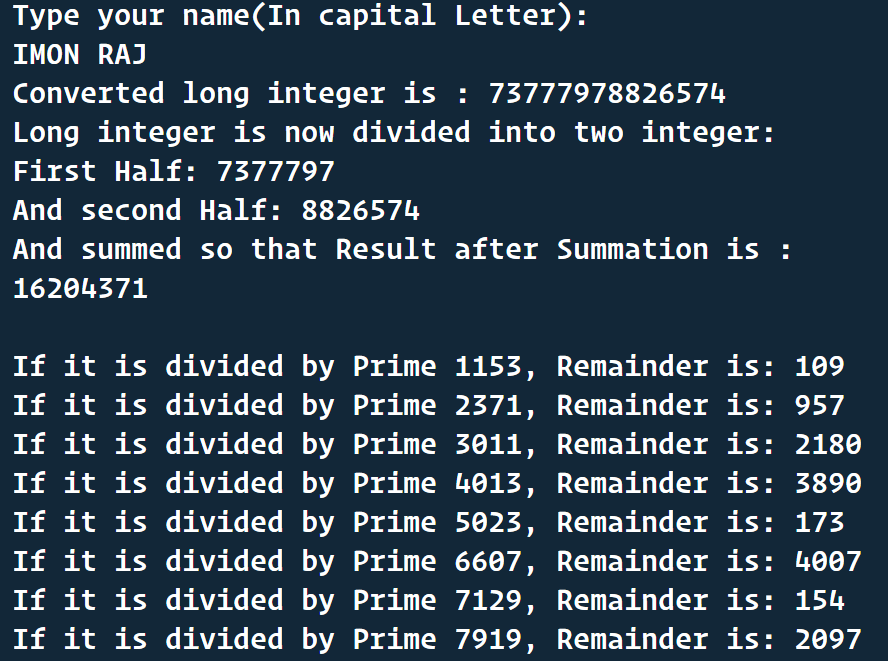
printf("Remainder is: %ld", d);

}

return 0;

}

**Results:**



**Discussion:**

This program helps to understand ascii arithmetic and how long integers are stored in array.