

**ASSIGNMENT 1**



**NAME: ROHIT SADHU ROLL NO.: 002010501074  
CLASS: BCSE II SEM: FIRST  
SESSION: 2021-2022**



**Problem 1.**

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:**

To find the factorial of a number iteratively we can use either a for loop or a while loop to iterate through the values from 1 to N(where N is the no. whose factorial is desired). For the recursive solution we will be making function calls to the same function until the base case is encountered. If our function is named “Func” and takes one parameter which is the input then we can write it like:

Func(n) = Func(n-1) \* n , where Func(k) = k! , for all natural k.

Also, we need to specify a base case for the recursion like when N -> 1 it will return 1.

**Structured Pseudocode:**

By using for loops-

Input N

For (loop from 1 to N)

prod = “Multiply all nos. from 1 to N as we proceed through the loop”

Print prod

By using recursion-

Func (No. N)

Base case: if N is 0 or 1 then return 1

return N \* Func(N-1)

Then we call the function passing N as argument.

**Results:**

Tested the codes on my machine where I used different data types and observed the overflow conditions:

1. For int data type overflow occurs at N = 17
2. For unsigned int data type overflow occurs at N = 34
3. For long long int data type overflow occurs at N = 40
4. For unsigned long long int data type overflow occurs at N = 66

**Discussion:**

The factorial of a number N is the product of all numbers from 1 to N including N. There is a special case where we assume factorial of 0 as 1. This is done to avoid confusion and calculation errors.

**Problem 2.**

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:**

To find the Nth Fibonacci of a number iteratively we can use either a for loop or a while loop to iterate through the values till N(where N is the no. such that Nth Fibonacci is our desired output). For the recursive solution we will be making function calls to the same function until the base case is encountered. If our function is named “Func” and takes one parameter which is the input then we can write it like:

Func(n) = Func(n-1) + Func(n-2) , where Func(k) = k – th Fibonacci , for all natural k.

Also, we need to specify a base case for the recursion like when N is 0 or 1 it will return 0 and 1 respectively.

**Structured Pseudocode:**

By using for loops-

Input N

For (loop till N)

res = “Sum the two preceeding nos.”

Print res

By using recursion-

Func (No. N)

Base case: if N is 0 or 1 then return N

return Func(N-1) + Func(N-2)

Then we call the function passing N as argument.

**Results:**

Tested the codes on my machine where I used different data types and observed the overflow conditions:

1. For int data type overflow occurs at N = 49
2. For unsigned int data type overflow occurs at N = 60
3. For long long int data type overflow occurs at N = 104
4. For unsigned long long int data type overflow occurs at N = 141

**Discussion:**

The Fibonacci series is special series where the N-th term of the series is equal to the sum of the (N-1)-th and (N-2)-th terms of the series. We take the first two terms of the series to be 0 and 1 and continue the series further.

**Problem 3.**

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

**Solution Approach:**

To solve this problem we can use arrays of different types storing different information and then finally search them by implementing the linear or binary search algorithms.

**Structured Pseudocode:**

Input an array of N nos.

Input element E to be searched.

Linear Search: Traverse through each element to find

For(loop from 1 to N)

If(E is equal to element at index i of array)

Then, break loop

Binary Search: Finding in a sorted array.

We keep three position indicators start, mid and end

start = index 0, mid = index N/2, end = index N-1

while(start<=end)

if(E = mid element)

then, break loop

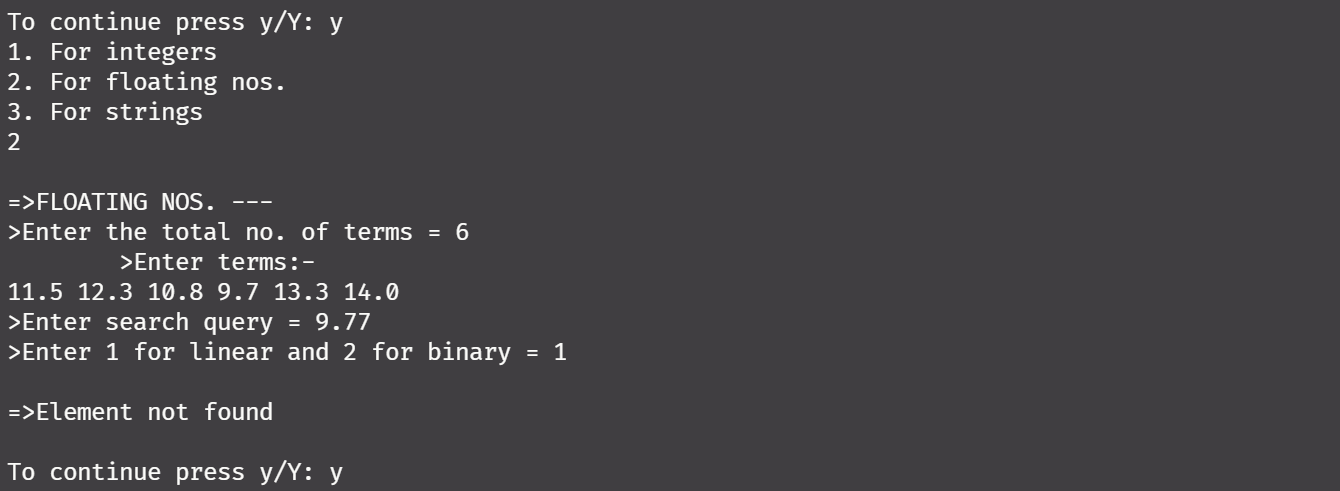
else if(E < mid element)

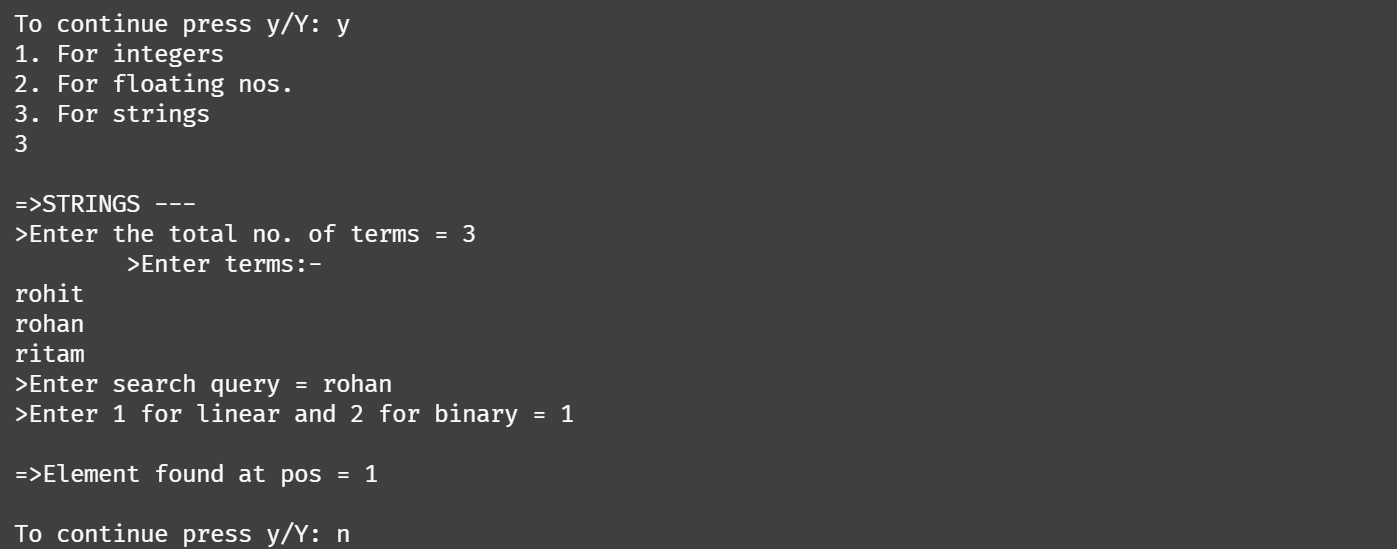
then, end = mid – 1 and mid = (start + end) / 2

else

then, start = mid + 1 and mid = (start + end) / 2

We can implement this for different data types like int, float and char.

**Results:**



**Discussion:**

Searching algorithms help us to find our desired result in an array. The popular searching algorithms include Linear Search and Binary Search algorithms. The linear search algorithm time complexity is of O(N) whereas the Binary Search algorithm time complexity is O(log(N)). We can clearly see that Binary Search is way more optimized than Linear Search and hence it is more preferred.

**Problem 4.**

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:**

To solve this problem we can use the rand() function in C present in stdlib.h header file. Also to find unique random nos. we may use a array to store a flag value which will indicate whether the random no. generated is already present in array or not.

**Structured Pseudocode:**

array[1e5 + 1]

r = rand() % (1e5 + 1)

for(loop from 1 to 100000)

If (r is present in array)

Then, continue

Else,

Flag = 1 (this specifies that element is now generated, to avoid repeatition)

**Results:**

The above results are stored in the mentioned file as output.

**Discussion:**

To show that the file containing the random nos. are not repeated we can sort the file using linux commands and check for the repeatitions. The rand() in C generates random values in the integer range and also we can use srand() to generate different random values each time we run the program.

**Problem 5.**

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:**

To solve this problem we can use the rand() function in C present in stdlib.h header file. Also to find unique random nos. we may use a array to store a flag value which will indicate whether the random no. generated is already present in array or not.

**Structured Pseudocode:**

array[1e5 + 1]

for(loop from 1 to 100000)

array[i] = rand() % 65 + 26

Now we print the values by using file handling.

**Results:**

The above results are stored in the mentioned file as output.

**Discussion:**

To show that the file containing the random nos. are not repeated we can sort the file using linux commands and check for the repeatitions. The rand() in C generates random values in the integer range and also we can use srand() to generate different random values each time we run the program.

**Problem 6.**

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:**

We first store the names in a text file as “names.txt” (one name per line) and then sort the names by using any sorting algorithm which I have used bubble sorting herein.

**Structured Pseudocode:**

Pseudocode to sort the names is as follows:

We take a 2-D character array and store the names first, then

For( loop i from1 to N)

For(loop j from 1 to N-1)

If(name j > name j+1)

Swap the names

Pseudocode to store the names in file:

File \*f = fopen(newfile name, file opening mode)

For(loop from 1 to N)

Using fputs/fwrite function we write the names in the file.

**Results:**

The output of the sorted names can be viewed from the newfile where they are stored and also the smallest and largest names are also displayed on the output screen once we run the program.

**Discussion:**

Here I demonstrated the method where I dynamically create arrays and store the result there temporarily, then sort them and again transfer the records to new file. This process is a bit time consuming but does the job quite well.

**Problem 7.**

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 others four-digit prime numbers keeping Li fixed.

**Solution Approach:**

We first generate a series of random nos. using inbuilt rand() function in C and then take user input of some prime nos. and display the modulus of the series of nos. by the user input values.

**Structured Pseudocode:**

Pseudocode to sort the names is as follows:

We take a 1-D integer array to store the random series of nos.

Array[N]

For( loop i from1 to N)

Array[i] = rand()

Then we take user input of some prime nos.

Input primes[M] (where primes array stores all the prime nos. input by user)

Then we do the modulus of the series of nos. present in array[N] by primes[M]

For(loop i from 1 to M)

For(loop j from 1 to N)

Print array[j] % primes[i]

**Results:**

Enter your prime no. = 1087

1. 654 | 945 | 592 | 311 | 538 | 43 | 766 | 810 | 887 | 1001 |

Enter your prime no. = 1993

2. 472 | 945 | 591 | 1578 | 1624 | 223 | 220 | 1172 | 1247 | 1544 |

Enter your prime no. = 2293

3. 1265 | 945 | 1084 | 1771 | 2117 | 416 | 306 | 572 | 1633 | 644 |

Enter your prime no. = 2909

4. 1710 | 945 | 913 | 984 | 1946 | 2538 | 1025 | 2249 | 59 | 1705 |

Enter your prime no. = 3082

5. 1191 | 945 | 221 | 119 | 1254 | 1846 | 2377 | 2076 | 1411 | 1359 |

Enter your prime no. = 3911

6. 2615 | 945 | 816 | 3796 | 1849 | 2441 | 2738 | 1247 | 1772 | 3612 |

Enter your prime no. = 4241

7. 1955 | 945 | 4067 | 2806 | 859 | 1451 | 428 | 917 | 3703 | 3282 |

Enter your prime no. = 4943

8. 551 | 945 | 2663 | 700 | 3696 | 4288 | 457 | 215 | 4434 | 2580 |

These are modulus outputs.

**Discussion:**

Here I demonstrated the method where I created two arrays and store the result there temporarily, and display them on the screen.

**Problem 8.**

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 first convert the name entered as input by juxtaposing it and then take user input of some prime nos. and display the modulus of the series of nos. by the user input values.

**Structured Pseudocode:**

Pseudocode to sort the names is as follows:

We take a 1-D integer array to store the random series of nos.

Input name

Then we take an array and convert the name to numericals by juxtaposing it.

Numerals[] = stores juxtaposed values of name

Then we take user input of some prime nos.

Input primes[M] (where primes array stores all the prime nos. input by user)

Then we do the modulus of the series of nos. present in array[N] by primes[M]

For(loop i from 1 to M)

Print numerals % primes[i]

**Results:**

Enter name: rohit sadhu

=> Name: rohit sadhu

SUM = 23110061133

Enter your prime no. = 1009

mod val = 808

Enter your prime no. = 1993

mod val = 438

Enter your prime no. = 2293

mod val = 1015

Enter your prime no. = 2909

mod val = 2254

Enter your prime no. = 3082

mod val = 1579

Enter your prime no. = 3911

mod val = 1243

Enter your prime no. = 4241

mod val = 3933

Enter your prime no. = 4943

mod val = 3803

**Discussion:**

Here I demonstrated the method where I converted the name by juxtaposing it then stored the result there temporarily, computed modulus of it by some prime nos. and displayed them on the screen.