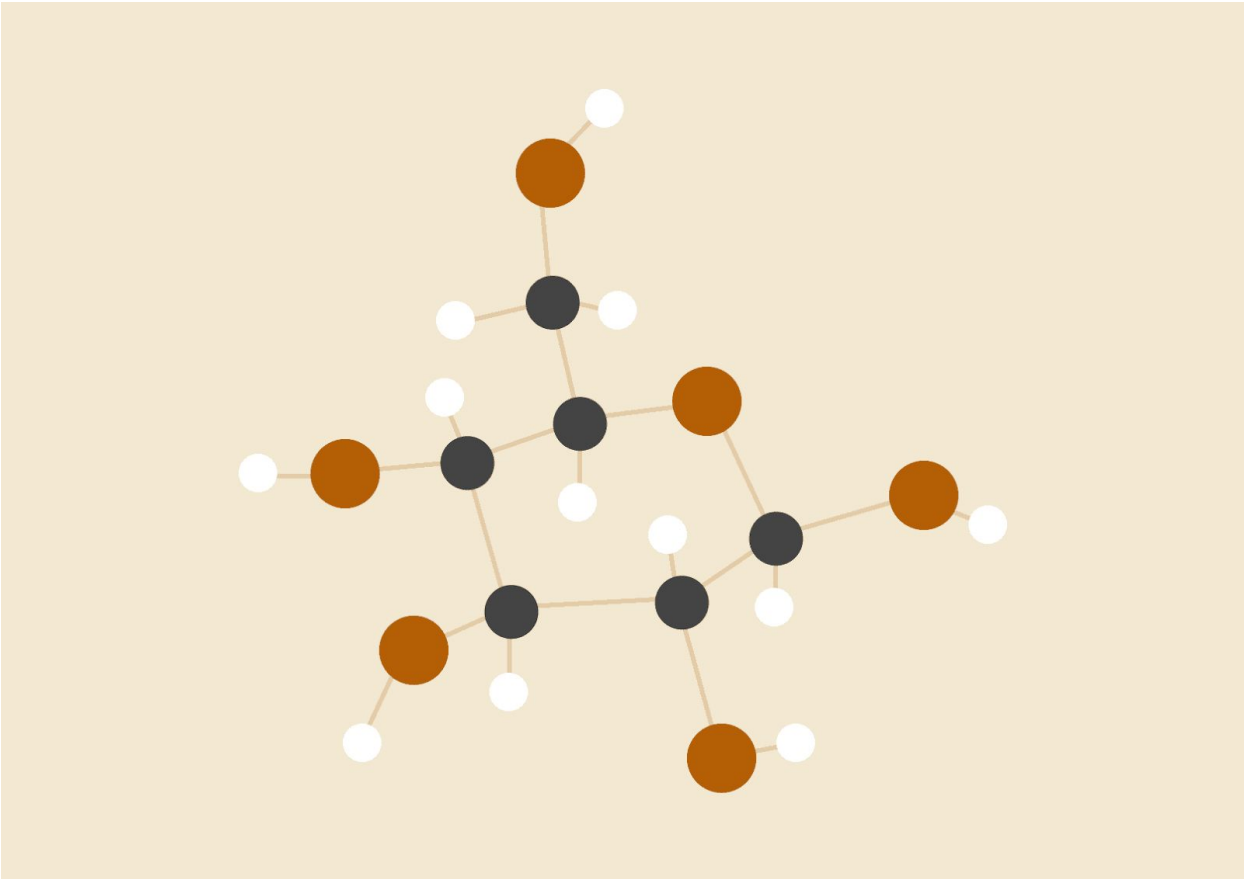
DSA LAB REPORT

CLASS *BCSE II* ***SEM*** *FIRST* ***SESSION*** *2020-21*



NAME Neeladri Pal

ROLL 001910501015

ASSIGNMENT SET - 1

PROBLEM NUMBER - 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

For recursive solution, use recurrence relation **factorial (n) = n * factorial (n-1)** with base condition **factorial (0) = 1**.

For iterative solution, factorial of n > 0 can be computed by multiplying consecutive integers from 1 to n in a loop. For n = 0, factorial (n) = 1.

STRUCTURED PSEUDOCODE

```
function factRecursive (n)
        if n < 2
                return 1
        else
                return n * factRecursive (n-1)
        endif

function factIterative (n)
        init f = 1
        while n > 0
                f = f * n
                n = n - 1
        end while
        return f
```

RESULTS

For data type: int,

```
Assignment 1$ gcc -o a1 sol1.c
Assignment 1$ ./a1
Enter number: 12
By Recursive method, 12! = 479001600
By Iterative method, 12! = 479001600
Assignment 1$ ./a1
Enter number: 13
By Recursive method, 13! = 1932053504
By Iterative method, 13! = 1932053504
```

For data type: long long int,

```
Assignment 1$ gcc -o a1 sol1.c
Assignment 1$ ./a1
Enter number: 20
By Recursive method, 20! = 2432902008176640000
By Iterative method, 20! = 2432902008176640000
Assignment 1$ ./a1
Enter number: 21
By Recursive method, 21! = -4249290049419214848
By Iterative method, 21! = -4249290049419214848
```

DISCUSSION

DATA TYPE	NUMBER	EXPECTED VALUE	OBTAINED VALUE
int	12	479001600	479001600
	13	6227020800	1932053504
long long int	20	2432902008176640000	2432902008176640000
	21	51090942171709440000	-4249290049419214848

For int data type, overflow occurs at n = 13.
For long long data type, overflow occurs at n = 21.

SOURCE CODE

sol1.c

PROBLEM NUMBER - 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

For recursive solution, use recurrence relation **fib (n)** = **fib (n-1) * factorial (n-2)** with base condition **fib (1)** = **0** and **fib (2)** = **1**.

For iterative solution, 1st and 2nd terms are returned directly. For n > 3, the last two terms are added iteratively to the current variable, till we get the nth term.

STRUCTURED PSEUDOCODE

```
function fibRecursive (n)
        if n == 1 or n == 2
                return n - 1
        else
                return fibRecursive (n-1) + fibRecursive (n-2)
        endif

function factIterative (n)
        if n == 1 or n == 2
                return n -1
        endif
        init previous = 0, current = 1
        for i = 2 to n - 1 in steps of 1
                init tmp_previous = precious
                previous = current
                current = tmp_previous + current
        end for
```

```
    return current
```

RESULTS

For int data type,

```
Assignment 1$ gcc -o a2 sol2.c
Assignment 1$ ./a2
Enter number (> 0): 46
By iterative method, 46th Fibonacci Number = 1134903170
By recursive method, 46th Fibonacci Number = 1134903170
Assignment 1$ ./a2
Enter number (> 0): 47
By iterative method, 47th Fibonacci Number = 1836311903
By recursive method, 47th Fibonacci Number = 1836311903
Assignment 1$ ./a2
Enter number (> 0): 48
By iterative method, 48th Fibonacci Number = -1323752223
By recursive method, 48th Fibonacci Number = -1323752223
```

For long long int data type,

```
Assignment 1$ gcc -o a2 sol2.c
Assignment 1$ ./a2
Enter number (> 0): 92
By iterative method, 92th Fibonacci Number = 4660046610375530309
By recursive method, 92th Fibonacci Number = 4660046610375530309
Assignment 1$ ./a2
Enter number (> 0): 93
By iterative method, 93th Fibonacci Number = 7540113804746346429
By recursive method, 93th Fibonacci Number = 7540113804746346429
Assignment 1$ ./a2
Enter number (> 0): 94
By iterative method, 94th Fibonacci Number = -6246583658587674878
By recursive method, 94th Fibonacci Number = -6246583658587674878
```

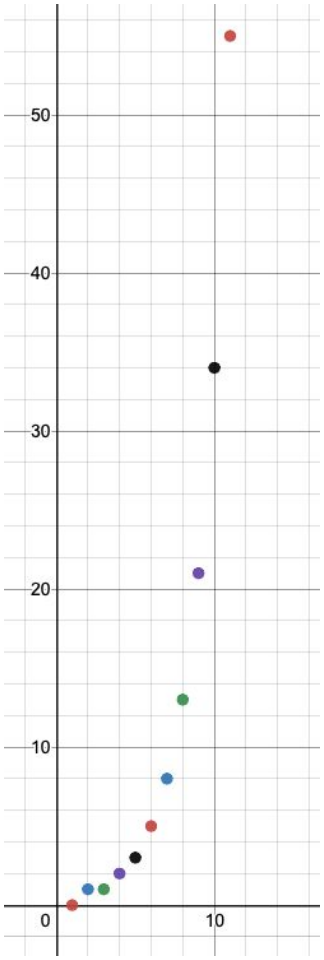
DISCUSSION

DATA TYPE	NUMBER	EXPECTED VALUE	OBTAINED VALUE
int	47	1836311903	1836311903
	48	2971215073	-1323752223

long long int	93	7540113804746346429	7540113804746346429
	94	12200160415121876738	-6246583658587674878

For int data type, overflow occurs at n = 48.
For long long data type, overflow occurs at n = 94.
Iterative solution takes fraction of seconds to execute for both the data types. But the recursive solution for large values of n takes much longer, the one for n = 94 takes around 30 min.

Graph of fib (n) vs n :-



SOURCE CODE

sol2.c

PROBLEM NUMBER - 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

Linear search : Traverse the array and stop when the desired element is found.

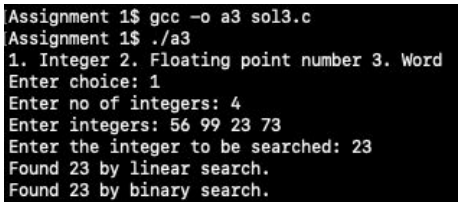
Binary search : The array needs to be sorted first. Then, a sliding window is maintained with the help of two indices left and right which are initialised to 0 and (size of array) - 1. A middle index is computed as (left + right) / 2. If the element to be found happens to be at the middle index, we have achieved the goal and hence we return, else, the middle index is assigned the value of left or right depending on the magnitude of the element to be found with respect to the one at the middle index. This loop continues till the left index becomes equal to the right index.

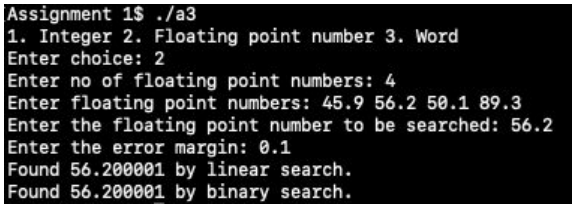
STRUCTURED PSEUDOCODE

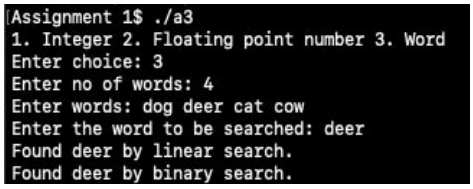
```
function binary_search (array a, n, x)
        sort array a
        init left = 0, right = n
        while left <= right
                init mid = left + (right - left) / 2
                if a[mid] is equal to x
                        print (“Element found”)
                        return
                endif
                if a[mid] > x
                        right = mid - 1
                else
                        left = mid + 1
                endif
```

```
        end while
        print ("Not found.")

function linear_search (array a, n, x)
        for i = 0 to n - 1 in steps of 1
                if a[i] is equal to x
                        print ("Found by linear search.")
                        return
                endif
        end for
        print ("Not found.")
```

RESULTS







DISCUSSION

For linear search, the worst time complexity is O(n).

For binary search, the worst time complexity is O(nlogn) since sorting is performed before proceeding to search operation.

Space complexity is O(1) in both cases.

SOURCE CODE

sol3.c

PROBLEM NUMBER - 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

An array arr of size n is allocated dynamically and values are assigned as **value at index i = i + 1**. Then, a loop runs from i = 0 to n - 1, and for each value of i, value at index i is swapped with value at a random index r between 0 to n-1 generated each time. Finally, this array of n random integers is returned.

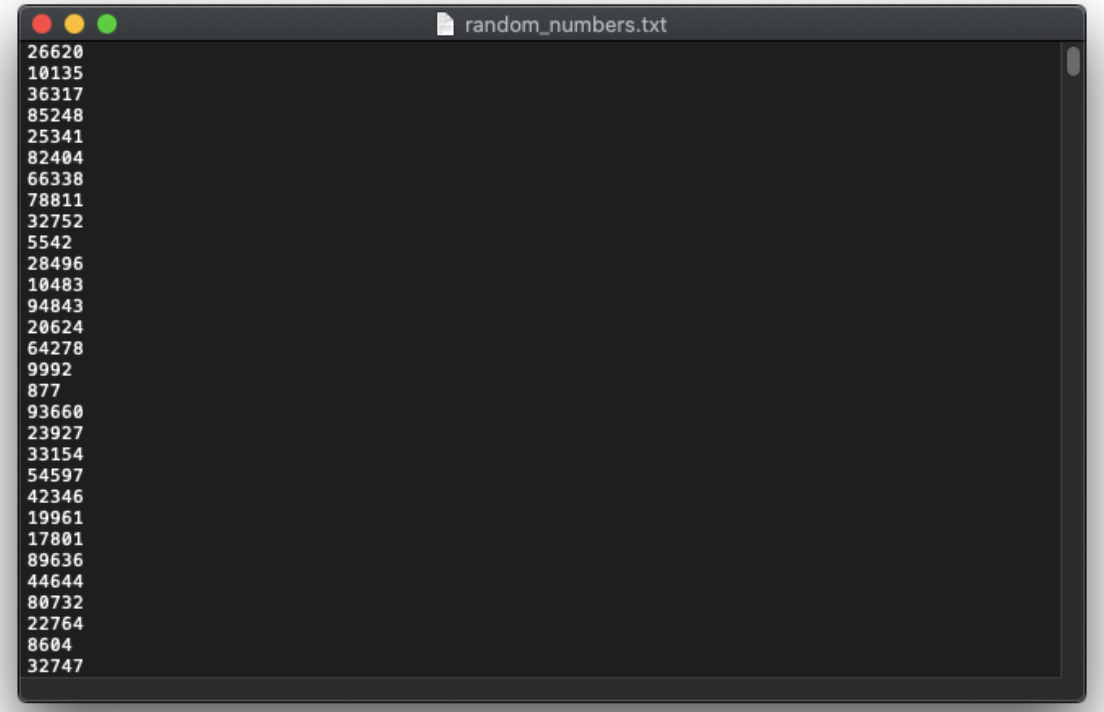
In main() function, a file is opened in write mode and each integer from the array of 100000 random integers is stored digit-wise as character, followed by a newline character at the end of each integer.

STRUCTURED PSEUDOCODE

```
function generate_random (n)
        init p = pointer to dynamically allocated array of size n
        for i = 0 to n - 1 in steps of 1
                p[i] = i + 1
        end for
        srand (time (0))
        for i = 0 to n - 1 in steps of 1
                init r = rand() % n
                swap values of p[i] and p[r]
        end for
        return p

function main()
        init n = 100000
```

```
init fp = pointer to file opened in write mode
if fp is NULL
        print (“File could not be created”)
        exit
end if
init random = call generate_random (n)
for i = 0 to n-1 in steps of 1
        init t = random[i], x = - 1
        while t > 0
                x = x + 1
                t = t / 10
        end while
        init divisor = 10^x
        while divisor > 0
                init c = (cast to character data type) ((random[i] / divisor) % 10)
                write c in fp
                divisor = divisor / 10
        end while
        write ‘\n’ in fp
end for
close file pointed to by fp
free space pointed to by random
```

RESULTS



DISCUSSION

The following code uses Fisher-Yates shuffle algorithm to generate a random permutation of integers from 1 to 100000. The algorithm works in O(n) time complexity and O(1) space complexity.

SOURCE CODE

sol4.c

PROBLEM NUMBER - 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

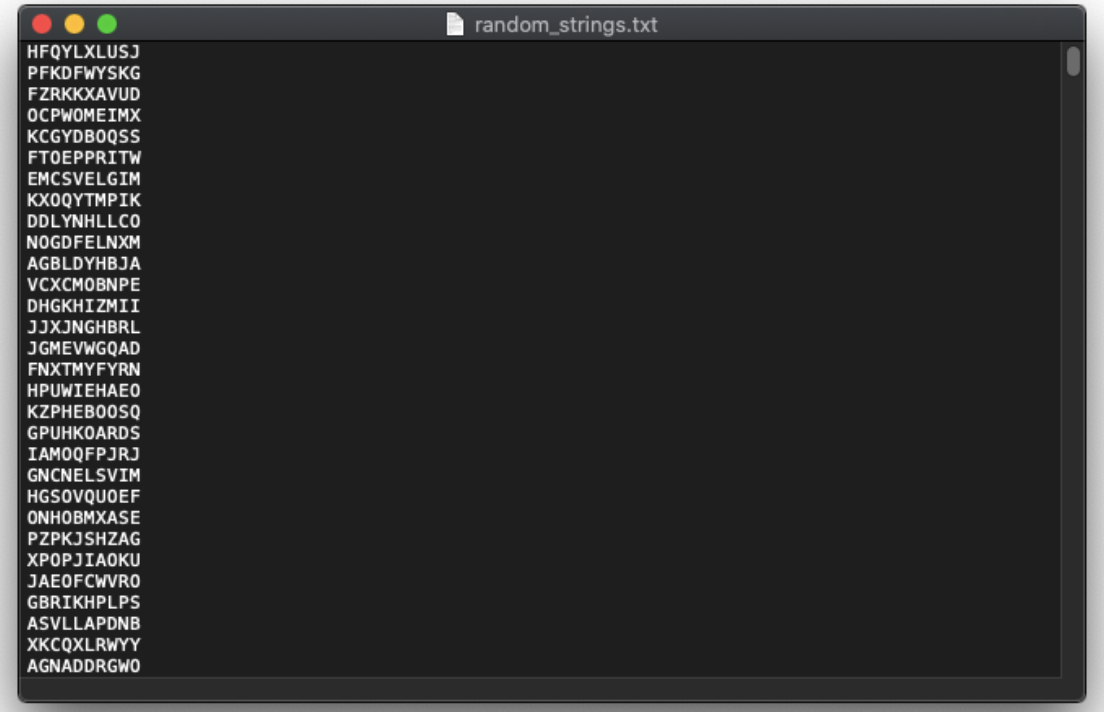
A two-dimensional character array **a** of dimensions (100000, 11) is declared. A loop runs from i = 0 to 100000. For each value of i, a random string **t** is generated by declaring a character array of size 11 and assigning **value at index i** = **random uppercase alphabet between A - Z** and t[10] = '\0'. This random string is compared with all the strings from a[0] to a[i]. If it turns out to be unique, this random string is copied into a[i], else the loops goes for another extra iteration. Thus, array **a** is generated which contains 10,000 random strings of size 10 without any repetitions. These strings are then stored in a file with one string on each line.

STRUCTURED PSEUDOCODE

```
function get_random_string (t[])
        for i = 0 to 9 in steps of 1
                t[i] = random uppercase alphabet between A - Z
        end for
        t[10] = '\0'

function main ()
        declare 2d character array a[100000][11]
        for i = 0 to 99999 in steps of 1
                declare 1d character array t[11]
                get_random_string (t);
                init flag = 1;
                for j = 0 to i - 1 in steps of 1
                        if t and a[i] are same
```

```
                    flag = 0
                endif
            if (flag)
                copy t in a[i]
            else
                i = i - 1
            endif
    end for
    init *fp = pointer to file opened in write mode
    if fp is NULL
            print ("File could not be opened.")
            exit
    endif
    for i = 0 to 99999 in steps of 1
            write a[i] in fp
    end for
    close file pointed to by fp
```

RESULTS



DISCUSSION

The following code generates a random string in O(m) time complexity, m = length of each word, and uses a brute-force algorithm to check duplicate strings which works in $O(n^2)$ time complexity, n = number of words.
The overall time-complexity is O(n*(n+m)). Space complexity is O(n*(n+m))

SOURCE CODE

sol5.c

PROBLEM NUMBER - 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

A two-dimensional character array **names** of dimensions (100, 31) is taken and names from the file are stored in it. This array is passed to the sort() function. A loop runs from i = 0 to n - 2. Inside this, a nested loop runs from j = 0 to n - i - 2. Whenever names[j] > names[j + 1], the values of the two arrays are swapped. Thus, in each iteration of the outer loop, the lexicographically largest array between names[0] to names[n - i - 1] is stored in names[n - i - 1]. At the last iteration, we get the sorted array.

STRUCTURED PSEUDOCODE

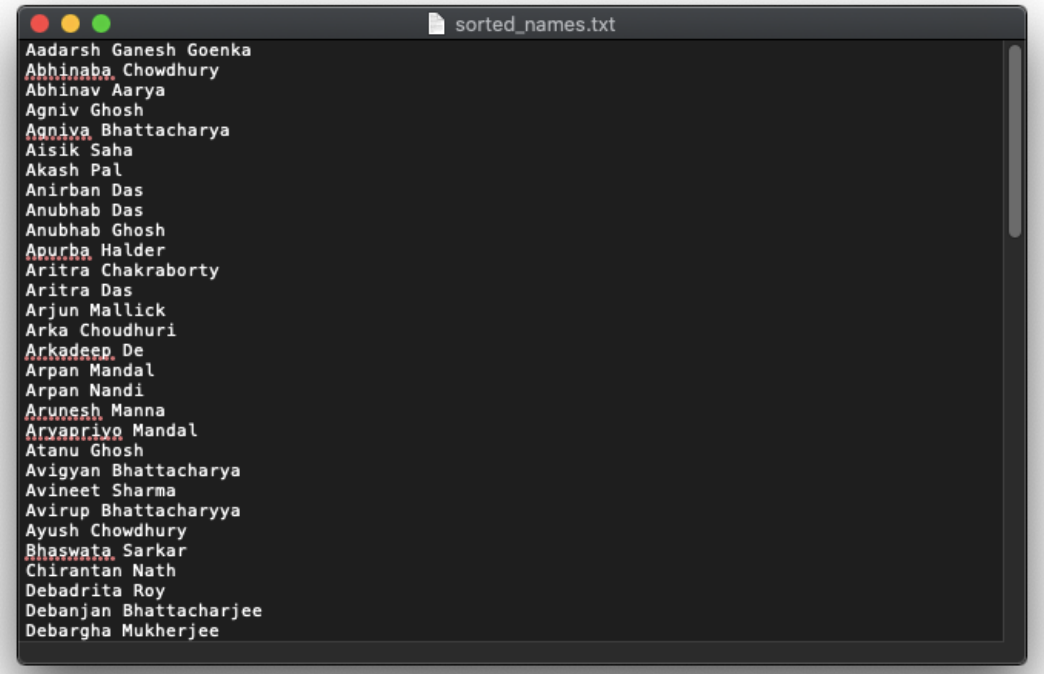
```
function sort (arr[][31], n)
        declare 1d character array temp[31]
        for i = 0 to n - 2 in steps of 1
                for j = 0 to n - i - 2 in steps of 1
                        if arr[j] > arr[j+1]
                                copy arr[j] in temp
                                copy arr[j+1] in arr[j]
                                copy temp in arr[j+1]
                        endif
                end for
        end for


function main ()
        init fp = pointer to file opened in read mode containing unsorted names
```

```
if fp is NULL
        print ("File could not be opened")
        exit
endif

declare 2d character array names[100][31];
init n = 0
while end of file pointed by fp is not reached
        read one line of fp into names [n]
        n = n + 1
end while
close file pointed to by fp
init M = 0, m = a large integer value, index_M = -1, index_m = -1;
for i = 0 to n - 1 in steps of 1
        init slen = length of name[i]
        if slen > M
                M = slen
                index_M = i
        endif
        if slen < m
                m = slen
                index_m = i
        endif
end for

print names[index_m], m
print names[index_M], M
call sort (names, n)

init fp = pointer to file opened in read mode containing unsorted names
if fp is NULL
        print ("Second file could not be opened.")
        exit
endif

for i = 0 to n - 1 in steps of 1
        write names[i] in fp
```

end for
close file pointed to by fp

RESULTS

```
Assignment 1$ gcc -o a6 sol6.c
Assignment 1$ ./a6
Smallest name (length):- Saif Raj (7)
Largest name (length):- Debanjan Bhattacharjee (21)
```



DISCUSSION

The following code uses bubble sort to sort the strings which works in $O(n^2)$ time complexity and O(1) space complexity.

SOURCE CODE

sol6.c

PROBLEM NUMBER - 7

PROBLEM STATEMENT

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

SOLUTION APPROACH

8 four-digit prime numbers are taken as input in an one dimensional integer array of size 8 and a sequence of 10 large random integers of length 9 are generated by taking a random value for each digit of a certain number. The remainders on dividing the large integers by the prime numbers are then printed in tabular format.

STRUCTURED PSEUDOCODE

```
function main ()
        declare 1d integer array prime [8]
        for i = 0 to 7 in steps of 1
                input prime[i]
        end for

        declare 1d integer array L [10]
        for i = 0 to 9 in steps of 1
                L[i] = random value between 1 to 9
                for j = 1 to 8 in steps of 1
                        L[i] = L[i] * 10 + random value between 0 to 9
                end for
        end for

        print ("Prime Numbers ->")
        for i = 0 to 7 in steps of 1
                print prime[i]
        end for
```

```
print '\n'

for i = 0 to 100 in steps of 1
        print '-'
end for
print '\n'

for i = 0 to 9 in steps of 1
        print L[i]
        for j = 0 to 7 in steps of 1
                print L[i] % prime[j]
        end for
        print '\n'
end for
```

RESULTS

```
Assignment 1$ gcc -o a7 sol7.c
Assignment 1$ ./a7
Enter 8 four-digit prime numbers: 1009 1033 1061 2857 3593 3677 4523 4969
Prime Numbers ->      1009      1033      1061      2857      3593      3677      4523      4969
-------------------------------------------------------------------------------------------------
      344951743       877      1020       484       420      2185      1342       625      3763
      388222003       172       976      1042      1415      1946       666      3867      3971
      357448581       241       624       864       740      2569        57       414      3566
      763459917        67       607       391       949      1312       730       132      2881
      419083839       734       904       510      1937      3505      1441       751      3348
      915187272        56       922       502      1605      3463       357      3452      1821
      984839580        85       106       282       253      1873      2931      1560      3656
      171790380        58       414       687      1827      1864       940      2317      2112
      794905650       315       787       206      2540      1109       759      1969      4782
      645366511        21       794       529      1638      2630      1533      2256      2729
```

SOURCE CODE

sol7.c

PROBLEM NUMBER - 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

Name is taken as input. The corresponding ASCII values are stored in a large_int 1d character array with each digit occupying one index. Two-separate 1-d integer arrays left_half and right_half are declared and the values from large_int are copied index-wise into the two arrays depending on the value of the index, whether it falls on the left half or the right half. Then, array addition is performed to obtain the sum in the form of a 1d integer array. Finally, the four-digit prime number P is taken as input and modular arithmetic is performed to get the remainder on dividing the sum by P.

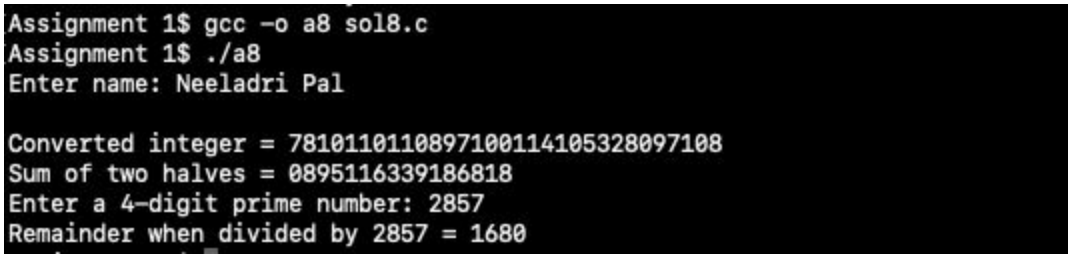
STRUCTURED PSEUDOCODE

```
function main ()
        declare 1d character array name[31]
        input name
        init 1d character array large_int[62] = ""

        for i = 0 until name[i] equals '\0' in steps of 1
                declare char temp[4]
                write ASCII value of name[i] in temp
                append temp to large_int
        end for

        print large_int
        init len = length (large_int)
```

```
declare 1d integer array left_half[31], right_half[31]
init p =0, q =0
for i = 0 to len - 1 in steps of 1
        if i < len / 2
                left_half[p] = (casted to integer data type) large_int[i]
                p = p + 1
        else
                right_half[q] = (casted to integer data type) large_int[i]
                q = q + 1
        endif
declare sum[32], reverse_sum[32]
init k =0, carry = 0;

while p > 0 AND q > 0
        int s = left_half[p] + right_half[q] + carry
        p = p - 1
        q = q - 1
        reverse_sum[k] = s % 10
        k = k + 1
        carry = s / 10
end while
while q > 0
        init s = right_half[q] + carry
        q = q - 1
        reverse_sum[k] = s % 10
        k = k + 1
        carry = s / 10
end while
if carry > 0
        reverse_sum[k] = carry
        k = k + 1
endif

for i = 0 to k - 1 in steps of 1
        print reverse_sum[k - i - 1]
end for
```

```
declare P
input P
init x = 1, rem = 0

for i = 0 to k - 1 in steps of 1
        rem += (reverse_sum[i] * x) % P
        rem %= P
        x = (x * 10) % P
print rem
```

RESULTS



DISCUSSION

Since the large integer formed by juxtaposing ASCII values of name may exceed the maximum limit of an integer data type, it is essential to use arrays in order to accommodate the actual integer.

Also, since remainder is required, modular addition and multiplication helps.
(x + y) % p = (a % p + b % p) % p
(x * y) % p = ((x % p) * (y % p)) % p

SOURCE CODE

sol8.c