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:

For recursive calculation of factorial of a number, we use the recurrance fact(n) = n \* fact(n-1), wherre fact(0) = 1.

For iterative approach we can us the same principle, but instead of finding all the factorials, we initialize a variable with 1 (fact=1), and multiply every integer less than equal to the given number, using a loop. The point is we store the factorial of 1 and use it to calculate the desired factorial.

For findng the overflow point, we can just run a loop calculating fact(i), and break the loop when factorial becomes negative, showing that ovverflow has occured.

For a better maximum limit, we can use some higher range data types like long long int, whose overflow point can be calculated using the same method.

Structured Pseudocode:

FUNCTION FACT\_RECUR ( INT N ):

IF N == 1:

RETURN 1

ELSE

RETURN N \* FACT\_RECUR( N-1 )

FUNCTION FACT\_ITER ( INT N ):

F = 1

FOR i=2 to n DO

F = F \* i

RETURN F

FUNCTION FIND\_MAX\_INT ( ):

RES=1

F=1 (int)

WHILE true DO:

IF R < 0:

BREAK

R = R + 1

F = F \* R

RETURN R

FUNCTION FIND\_MAX\_LONG ( ):

RES=1

F=1 (long long int)

WHILE true DO:

IF R < 0:

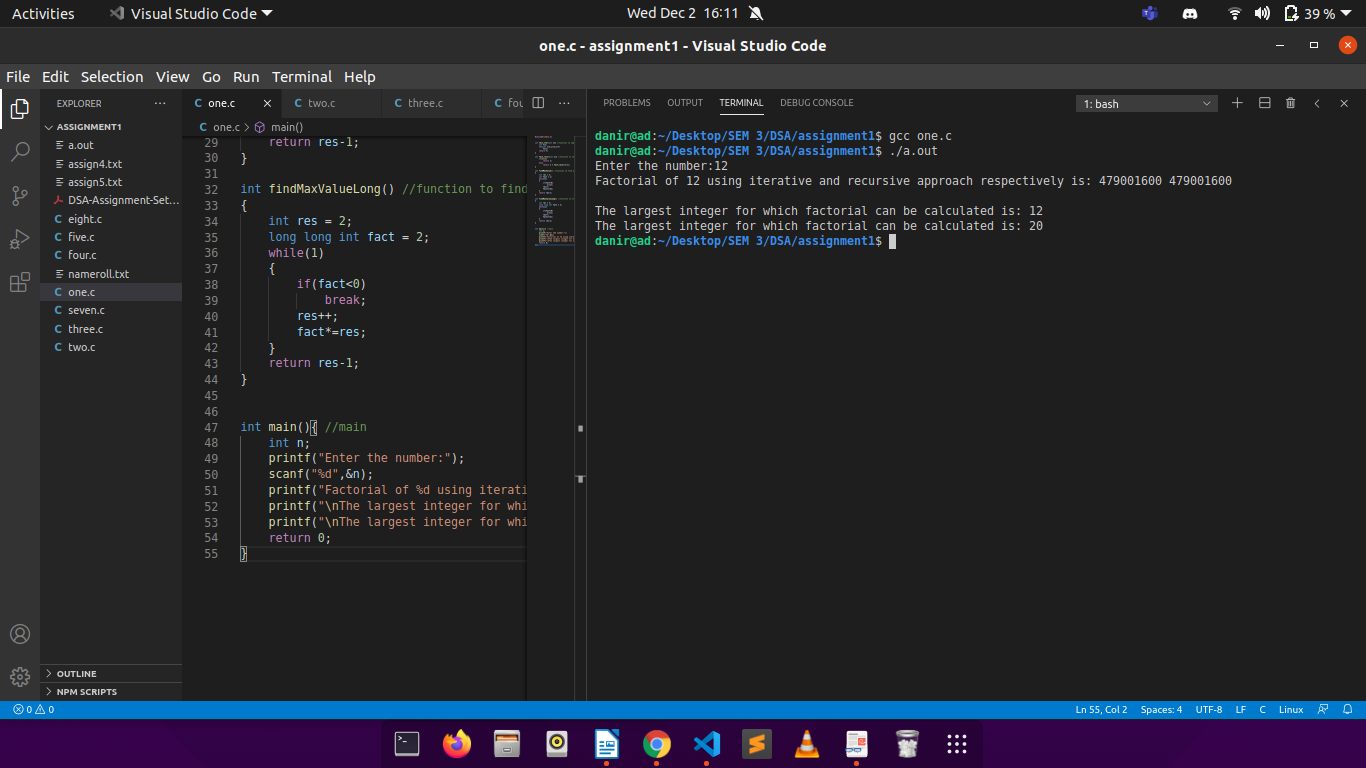
BREAK

R = R + 1

F = F \* R

RETURN R

Results:



There is a overflow of value of factorial at the integer number 13 (in case of int data type). The same overflow occurs at the integer number 21 (in case of long long data type).

Discussions:

The Time Complexiy in both recursive and iterative approach is O(n). The space complexity, however, is O(1) in case of iterative and O(n) in case of recursive.

From the results above, we can conclude that the largest int and long long number whose factorial can be calculated are 12 and 20 respectively.

Source Code:

FILE NAME: “one.c”

(can be found in the following link: <https://drive.google.com/drive/folders/1-nNb6aRleNLE1mcE58i85096fDmDUCvd?usp=sharing>)

**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:

For recursive approach to get fibonacci, we can use the recurrence fib(n)= fib(n-1)+fib(n-2), where fib(1) = 0 and fib(2) = 1. We can use these as the breaking conditions for the recursion.

For iterative calculation, using the same recurence, we can see that if we store the last 2 fibonacci values, we can get the desired fibonacci value. This can be done using a single for loop trom 3 to n, updating prev1 and prev2(storing the last 2 fibos) in each iteration.

For findng the overflow point of int, we can just run a loop calculating fib(i), fib(i-1) and fib(i-2) and check if the condition fib(i-1) + fib(i-2) == fib(i) is satisfied, if the don’t, the value of fib(i) will get overflowed. The overflow point can be increased by taking data types such as long long int.

Structured Pseudocode:

FUNCTION FIBO\_RECUR ( INT N ):

IF N <= 1:

RETURN N

ELSE

RETURN FIBO\_RECUR( N-1 ) + FIBO\_RECUR( N-2 )

FUNCTION FIBO\_ITER ( INT N ):

IF N<=1:

RETURN N

DECLARE ARRAY[N+1]

ARRAY[0]=0 ARRAY[1]=1

FOR i=2 TO N+1 DO:

ARRAY[i]=ARRAY[i-1]+ARRAY[i-2]

RETURN ARRAY[N]

FUNCTION FIND\_MAX\_INT ( ):

TEMP=0

RES=1

FIB=1 (int)

WHILE FIB+LAST>FIB DO:

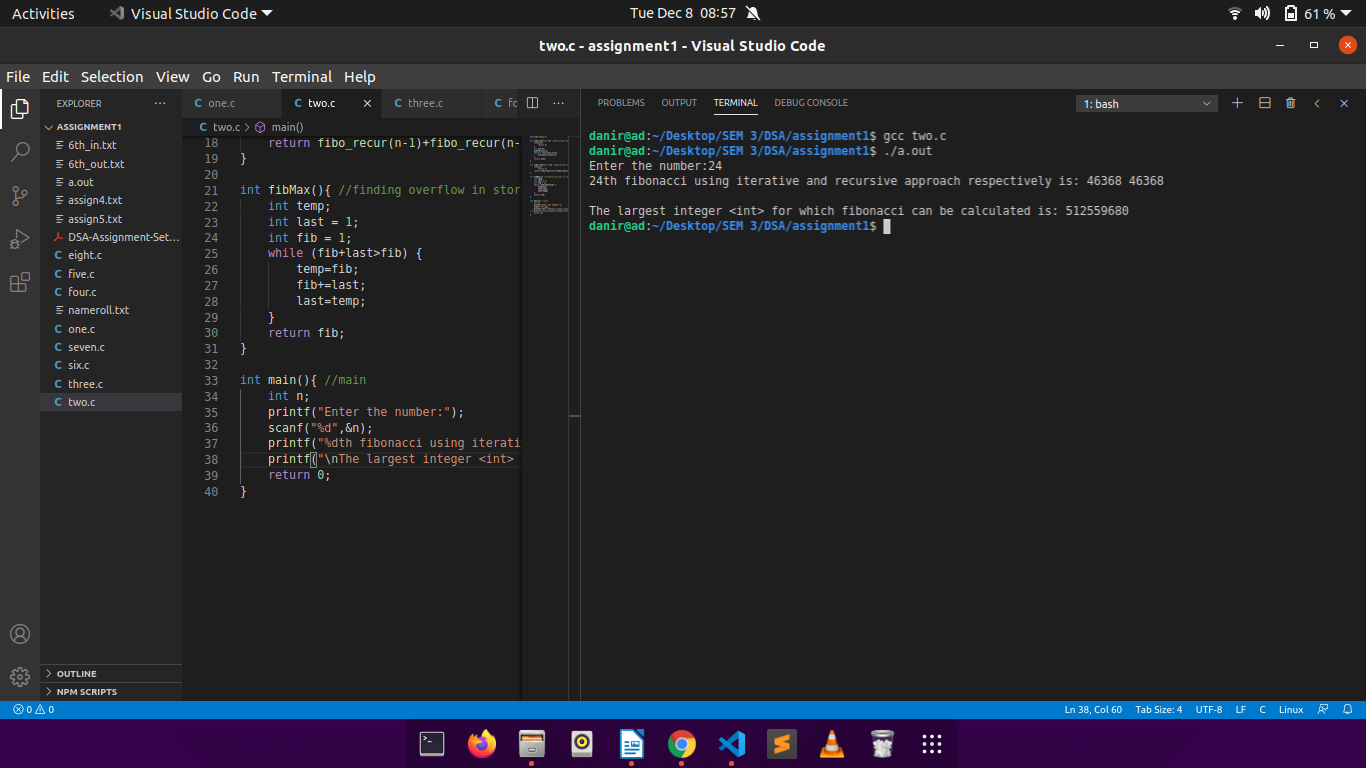
TEMP=FIB

FIB = FIB + LAST

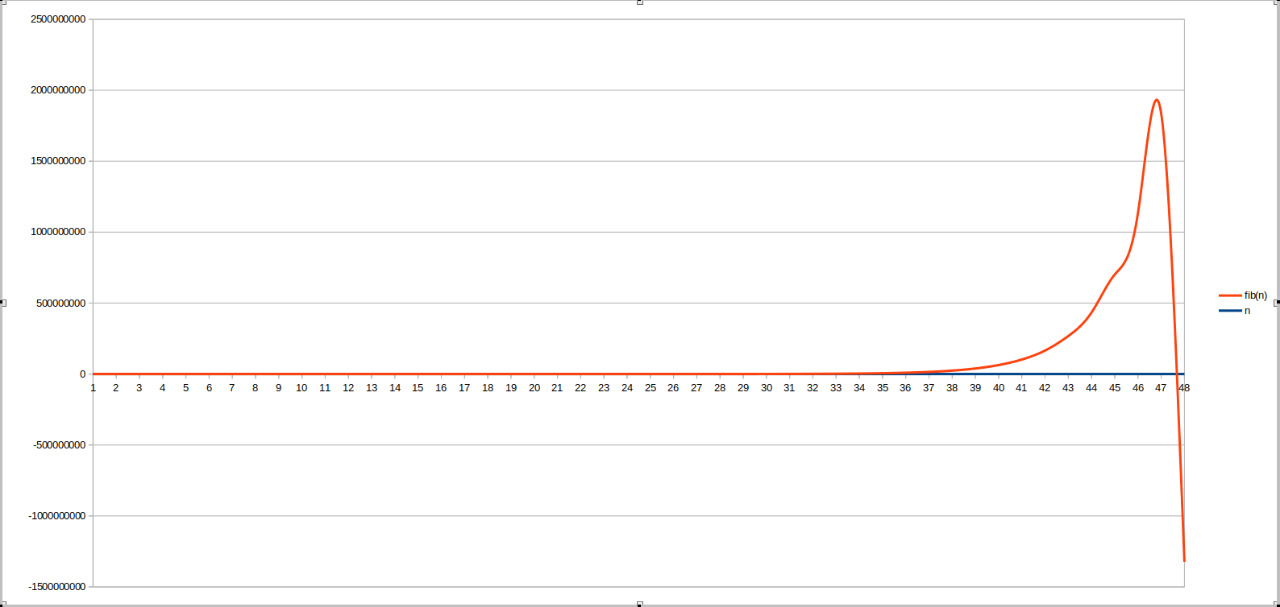
LAST = TEMP

RETURN FIB

Results:

There is a overflow in value of int for the fibbonacci number of 512559681 (in case of int data type).

Discussions:



The above graph shows fibonacci(red) vs integer number(blue). The fall at the end of the graph depicts the overflow in the value of fibonacci for int data type.

The Time Complexiy in both recursive and iterative approach is O(n). The space complexity, however, is O(1) in case of iterative and O(n) in case of recursive.

From the results we can conclude the largest fibonacci of type int that can be calculated is 512559680.

Source Code:

FILE NAME: “two.c”

(can be found in the following link: <https://drive.google.com/drive/folders/1-nNb6aRleNLE1mcE58i85096fDmDUCvd?usp=sharing>)

**Problem 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:

For the array of integers we run a binary search algorithm where we compare x(element to be seached) with the middle element of the array.

Three cases are handled here:

1. If x matches with middle element, we return the mid index.
2. If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we run recuursion for right half.
3. Else (x is smaller) recur for the left half.

For the array of floats and words a basic linear search aalgorithm is used. Here we traverse the whole array and make an equality check in every step. If the desired element is found 1(true) is returned, 0(false) otherwise.

Structured Pseudocode:

FUNCTION BSEARCH( INT ARRAY, INT LOWER, INT UPPER, INT X):

IF (UPPER>=LOWER):

MID=LOWER+(UPPER-1)/2

IF (ARRAY[MID]==X)

RETURN MID

IF (ARRAY[MID]>X)

RETURN BSEARCH(ARRAY, LOWER, MID-1, X)

IF (ARRAY[MID]<X)

RETURN BSEARCH(ARRAY, MID+1, UPPER, X)

RETURN -1

FUNCTION LSEARCH1 (INT ARRAY, INT N, INT X):

FOR i=0 TO N-1 DO:

IF (ARRAY[i]==X)

RETURN 1

RETURN -1

FUNCTON LSEARCH2 (CHAR ARRAY[], INT N, CHAR X):

FOR i=0 TO N-1 DO:

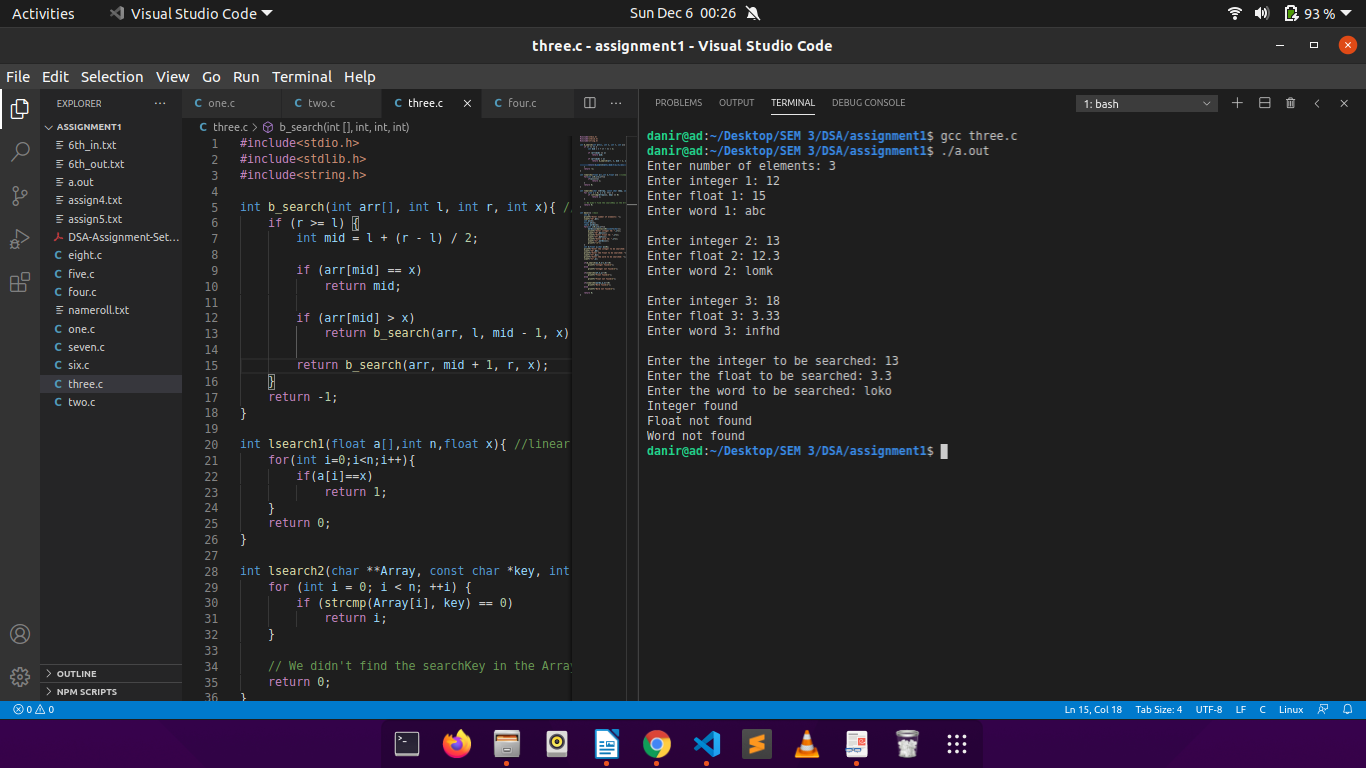
COMPARE ARRAY[i] AND X:

IF MATCHES

RETURN 1

RETURN 0

Results:



Discussions:

The binary search function takes O(log n) time complexity, whereas the linear search takes a classic O(n) as traversal of whole array takes place in the worst case.

Source Code:

FILE NAME: “three.c”

(can be found in the following link: <https://drive.google.com/drive/folders/1-nNb6aRleNLE1mcE58i85096fDmDUCvd?usp=sharing>)

**Problem 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:

The system time changes every second, and this is the property that is used in the following solution where srand() and rand() together with a time\_t variable is called. Inside a loop the rand()%100000 generates a random number less than or equal to 100000. A binary array is made which stores 1 if that index is already present in the file, 0 otherwise. A check is made if the number is repeated in which case, the loop variable is repeated, written to the file otherwise.

A binary file (also attached below) is used to store the random numbers.

Structured Pseudocode:

FILE \*PTR = FOPEN(FILE\_NAME, MODE)

TIME\_T VAR

ARRAY[10000] = {0}

SRAND (VAR)

FOR i=1 TO 100000 DO:

INT K=RAND()%100000

IF (K NOT IN FILE)

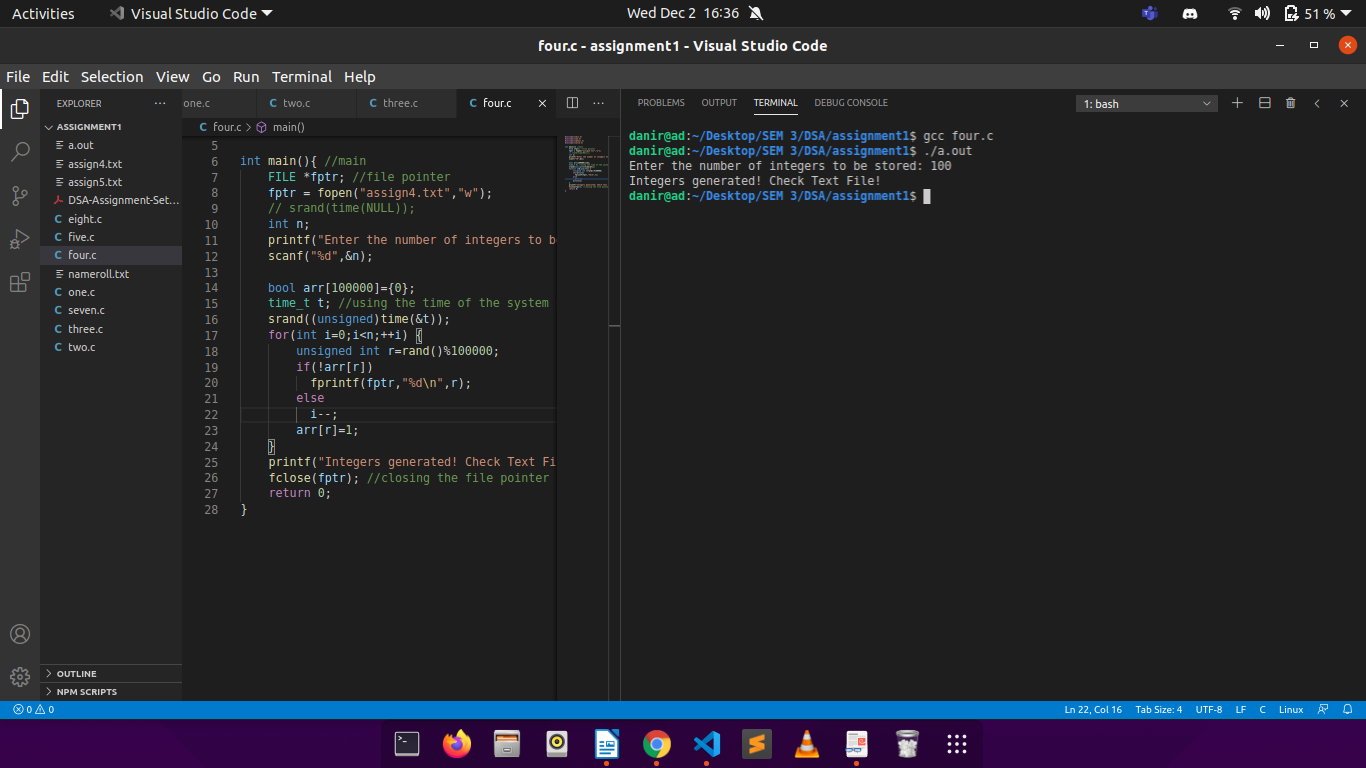
FPRINTF(PTR,K);

ELSE

i = i-1

ARRAY[i]=1

Results:



Discussions:

The time complexity of this solution is strictly O(n).

Source Code:

FILE NAME:

Code – “four.c”

Binary File – “assign4.txt”

(can be found in the following link: <https://drive.google.com/drive/folders/1-nNb6aRleNLE1mcE58i85096fDmDUCvd?usp=sharing>)

**Problem 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:

The system time changes every second, and this is the property that is used in the following solution where srand() and rand() together with a time\_t variable is called.

A character array containing the 26 alphabets is used to find different permutations and hence generate non-duplicate words.

Inside a loop the rand()%100000 generates a random number less than or equal to 100000. For eeach iteration of the outer loop an inner loop is ran in which a letter is chosen at random from the character array ad appended at the end of the string i.e to be written in the file.

A binary array is made which stores 1 if that index is already present in the file, 0 otherwise. A check is made if the number is repeated in which case, the loop variable is repeated, written to the file otherwise.

A binary file (also attached below) is used to store the random numbers.

Structured Pseudocode:

FILE \*PTR = FOPEN(FILE\_NAME, MODE)

TIME\_T VAR

CHAR ARRAY[26]={‘A’, ‘B’, ‘C’,............, ‘Y’, ‘Z’}

ARR[10000] = {0}

SRAND (VAR)

FOR i=1 TO 100000 DO:

INT K=RAND()%100000

FOR j=0 TO 9 DO:

RESULT[j]=ARRAY[RAND()%26]

IF (K NOT IN FILE)

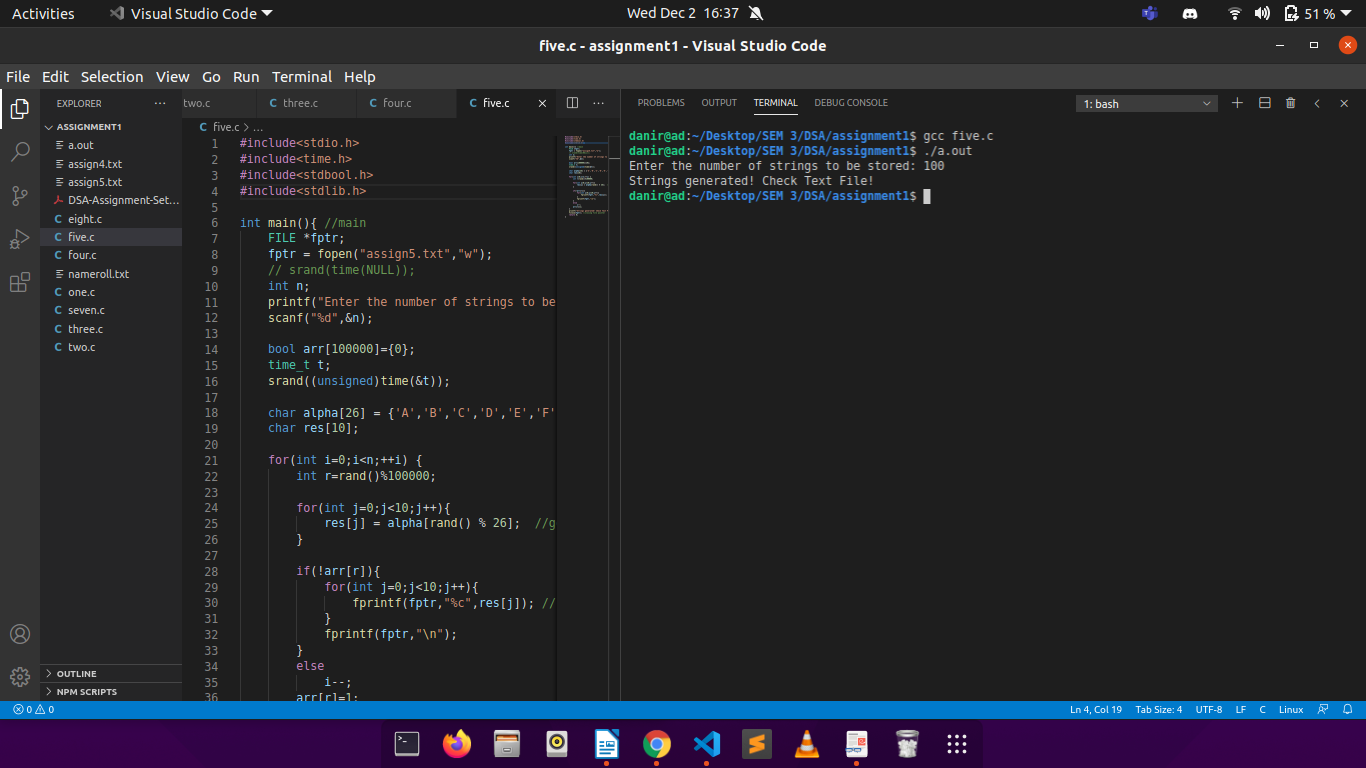
FPRINTF(PTR,K);

ELSE

i = i-1

ARR[i]=1

Results:



Discussions:

The solution approach uses O(n) time complexity.

Source Code:

FILE NAME:

Code – “five.c”

Binary File – “assign4.txt”

(can be found in the following link: <https://drive.google.com/drive/folders/1-nNb6aRleNLE1mcE58i85096fDmDUCvd?usp=sharing>)

**Problem 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:

Firstly input of students is taken in a text file and stored. Now this file is opened in read mode (“r”), and length of each student name is compared with pre-initialized variavles “max” and “min”, and the same are updated as required. A counter variable is incremented by 1 in each iteration to count the number of students. This process is ended when we reach EOF.

Next a 2-D array of strings is made and the correspondiong names from the file are stored. This array is modified by sorting the names in alphabetcal order using Bubble Sort algorithm.

Finally, a new file is created and the array of strings is written into it, followed by the Student names with maximum and minimum lengths respectively.

(NOTE: To find the length of th characters a function is defined which counts the length excluding the spaces)

Structured Pseudocode:

FUNCTION LENGTH(CHAR \*NAME):

C=0

L=STRLEN(NAME)

FOR i=0 TO L-1 DO:

IF(!ISSPACE(NAME[i))

C = C + 1

RETURN C

MAIN():

FILE \*OP = FOPEN(“FILE1.TXT”, “W”)

WHILE(TRUE) DO:

FGETS(STR, SIZEOF(STR), STDIN)

FPUTS(STR, OF)

FCLOSE(OP)

FILE\* READ = FOPEN(“FILE1.TXT”, “R”)

WHILE(FGETS(NAME, READ)!=NULL) DO:

IF(LENGTH(NAME)>MAX)

MAX=LENGTH(NAME)

IF(LENGTH(NAME)<MIN)

MIN=LENGTH(NAME)

COUNT = COUNT + 1

ARRAY [COUNT][50]

FOR i=0 TO COUNT-1 DO:

FSCANF(READ, ARRAY[i])

i = i+1

FOR i=0 TO COUNT-1 DO:

FOR j=i+1 TO COUNT-1 DO:

SWAP (ARRAY[j], ARRAY[j+1])

FILE\* OUT=FOPEN(“FLE2.TXT”, “W”)

FOR i= TO COUNT-1 DO:

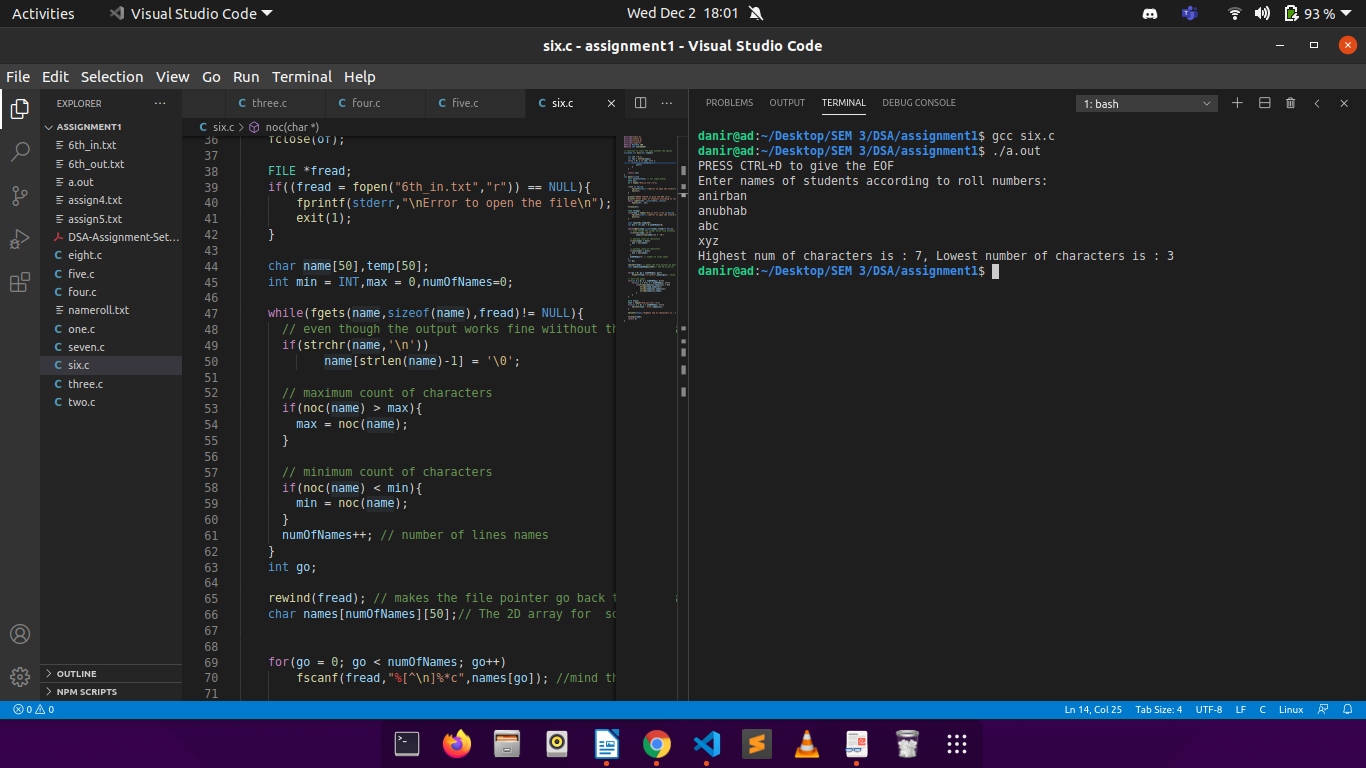
PRINT(ARRAY[i], OUT)

PRINT(MAX, MIN, OUT)

FCLOSE(OUT)

FCLOSE(READ)

Results:



Discussions:

In the worst case scenario the soting algorithm would take n\*n time to execute, hence the Time Complexity is O(n\*n).

Source Code:

FILE NAME:

Code – “six.c”

Binary Files – “6th\_in.txt” & “6th\_out.txt”

(can be found in the following link: <https://drive.google.com/drive/folders/1-nNb6aRleNLE1mcE58i85096fDmDUCvd?usp=sharing>)

**Problem 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 L(i) and R(i). Repeat for seven other four digit prime numbers keeping L(i) fixed.

Solution Approach:

Firstly a binary array is taken, which stores the prime status of a range of integers, i.e it stores 1 if the number is prime and 0 otherwise.

In the main, 20 large integers are generated using the rand() funtion on a very large integer (a.k.a 1e8), and stored in an integer array.

Using the binary array created earlier, the first eight, 4-digit prime numbers are found and stored in an array.

Finally, two nested loops are made with the outer one running 8 times and the inner 20 times. In eacch iteration of the outer loop, the 20 large integers are divided by a single prime number and the corresponding reminders are printed.

Structured Pseudocode:

FUNCTION SIEVE():

N = LARGE INTEGER (1e8)

FOR i=2 TO n DO:

IF( PRIME[i]!=0 ):

FOR j=2\*i TO n DO:

PRIME[j]=1

j = j + 1

i = i+1

MAIN():

FOR i=0 TO 20 DO:

LARGE[i] = 1E8 + RAND() % 1E8

i = i + 1

COUNT = 0

ARRAY[8] = {0}

FOR i=1001 DO:

IF (COUNT<8)

IF (PRIME[i] == 1)

ARRAY[COUNT] = i

COUNT = COUNT +1

i = i+1

FOR i=0 TO 8 DO:

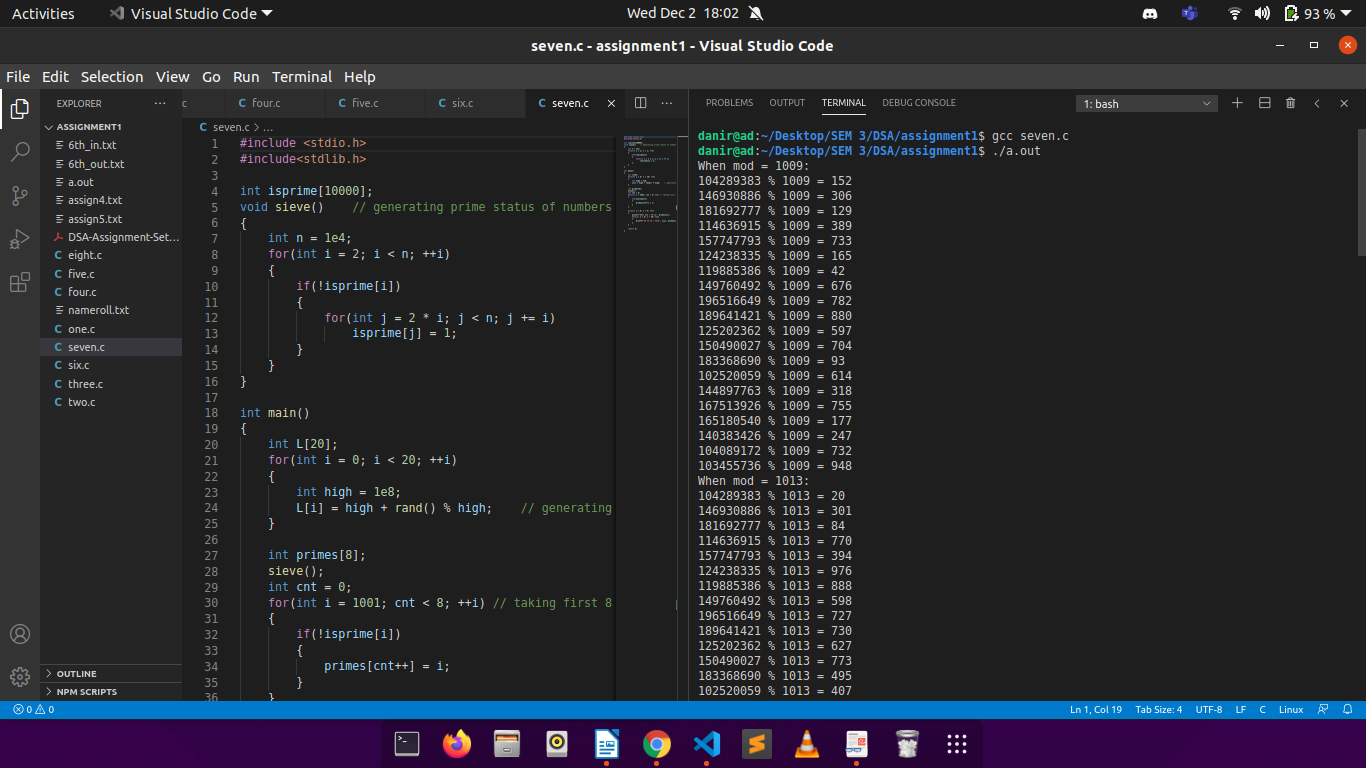
FOR j=0 TO 20 DO:

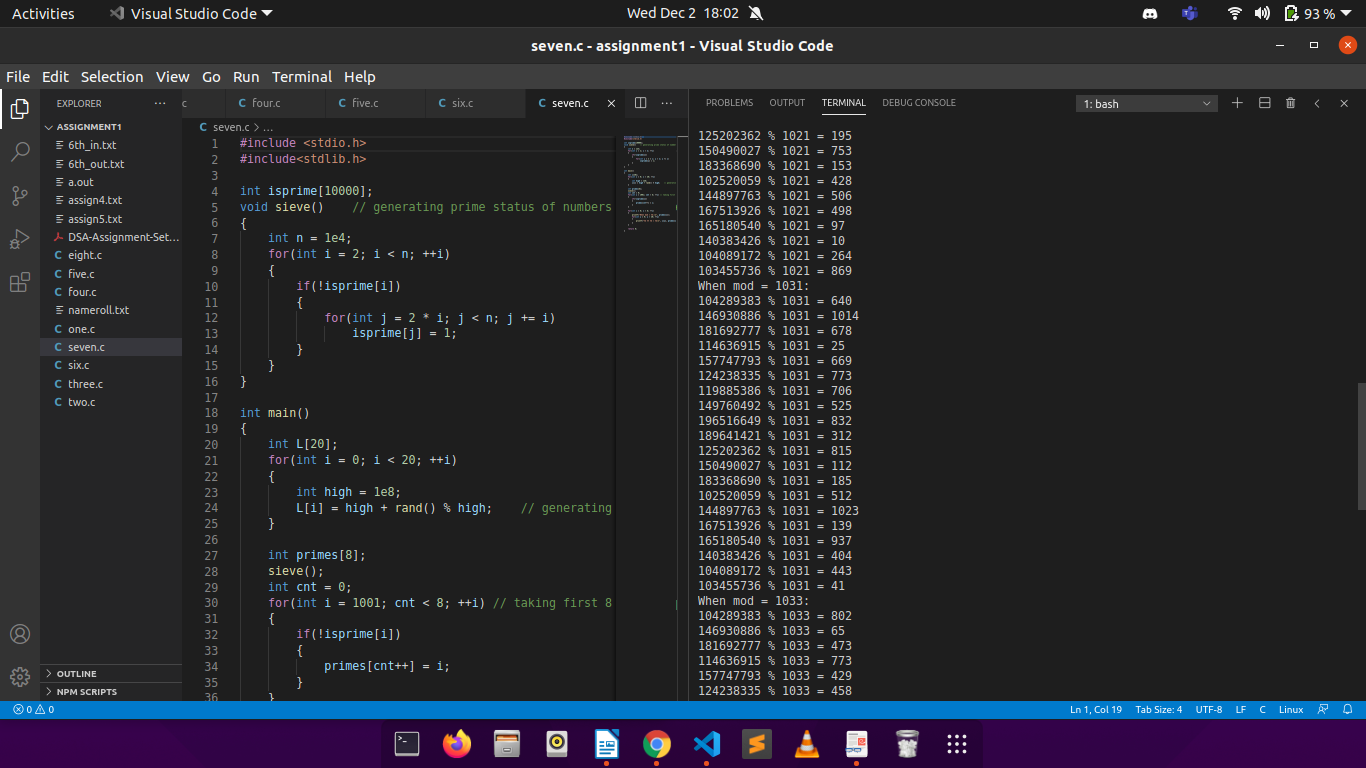
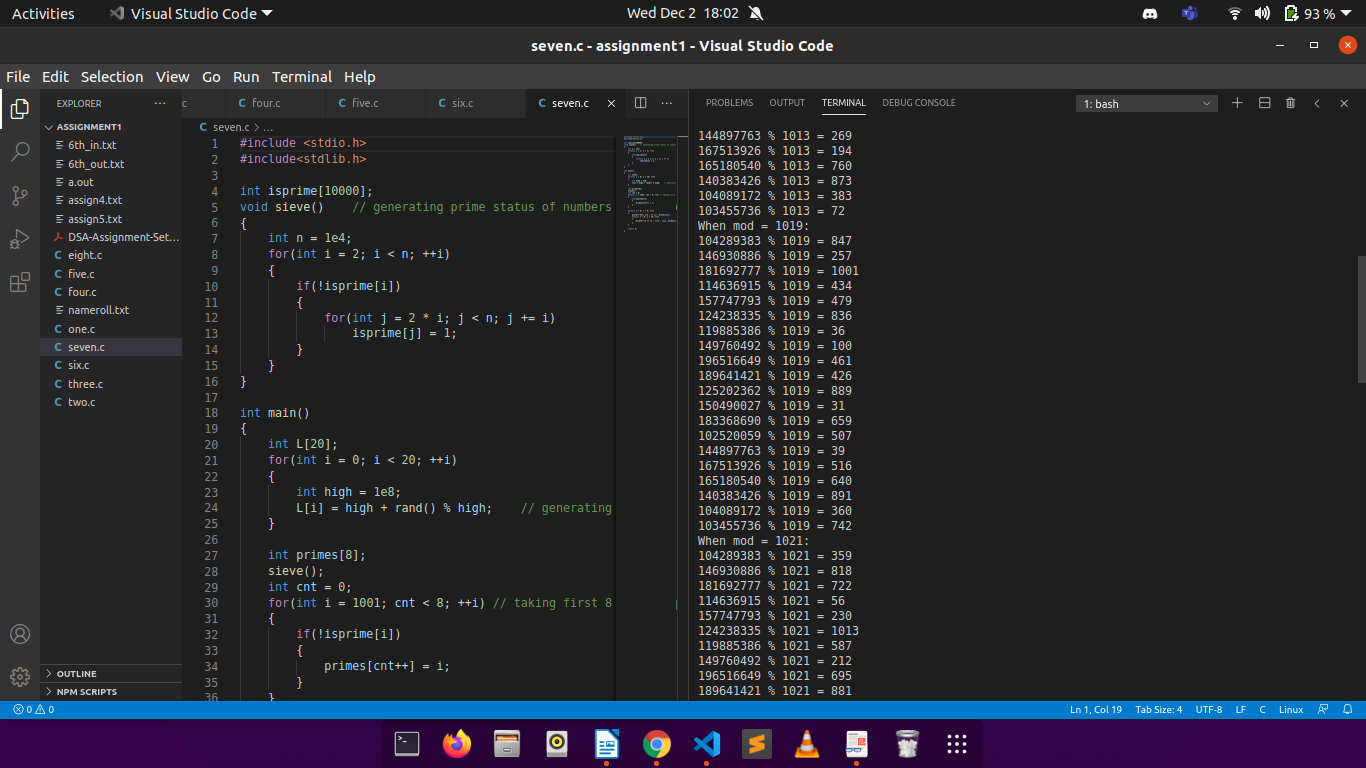
PRINT(ARRAY[i] % LARGE[j])

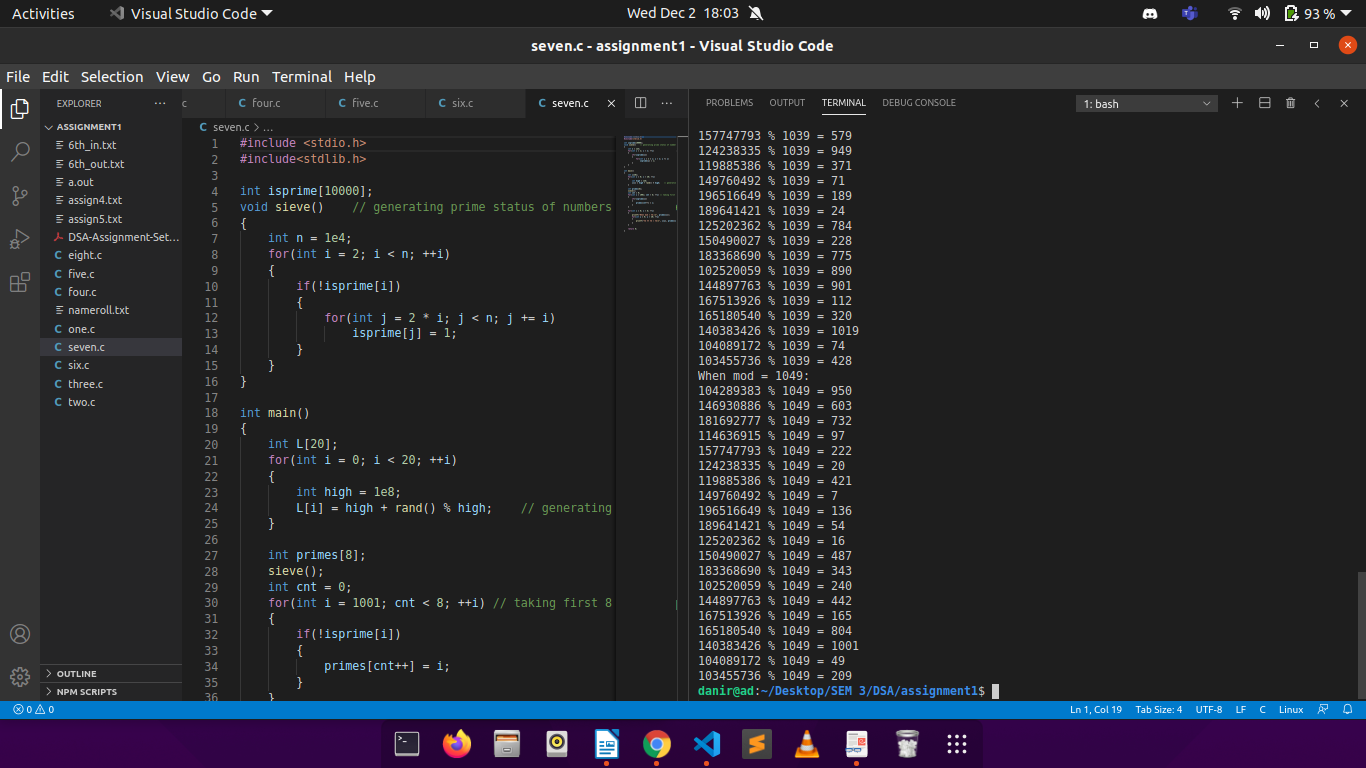
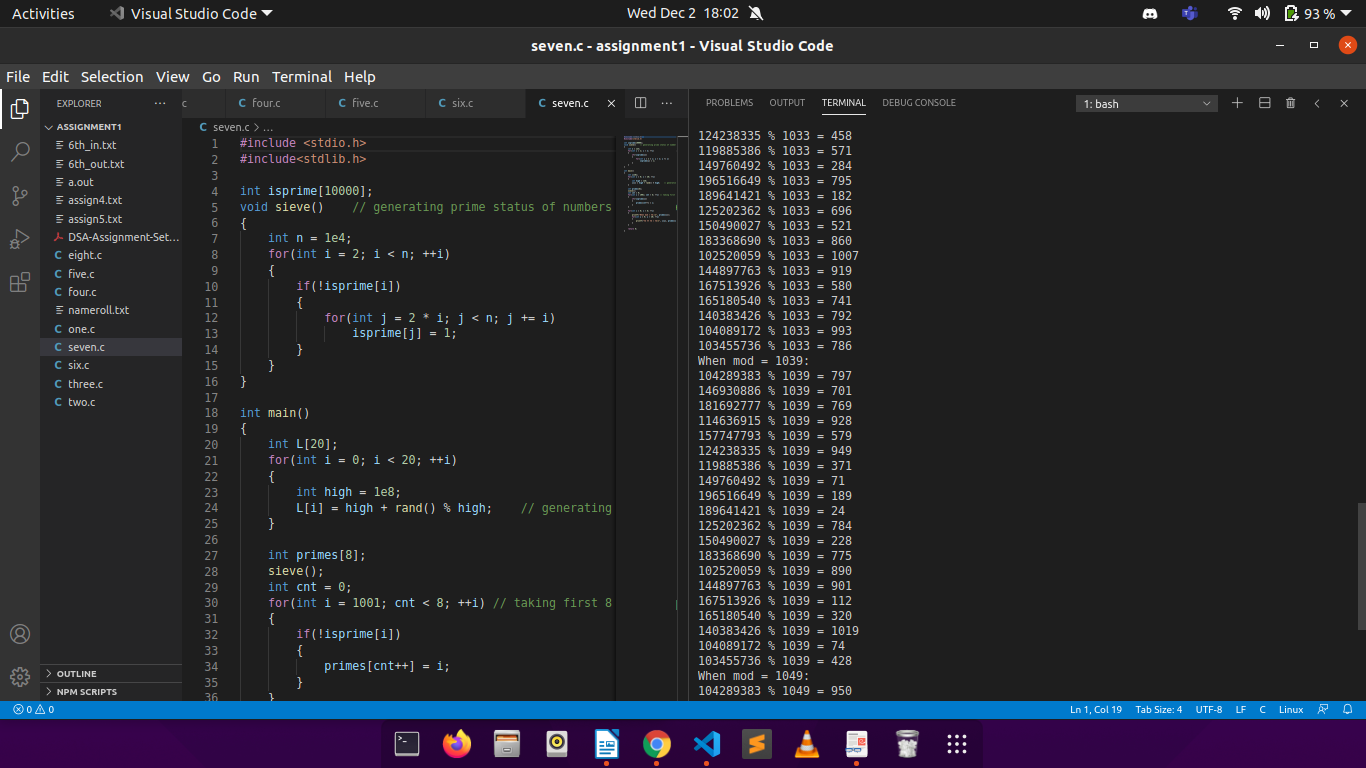
j = j+1

i = i + 1

Results:





Discussions:

The solution takes O(n) time complexity.

Source Code:

FILE NAME:

Code – “seven.c”

(can be found in the following link: <https://drive.google.com/drive/folders/1-nNb6aRleNLE1mcE58i85096fDmDUCvd?usp=sharing>)

Problem 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:

The user entered name is stored in an integer array to get the ascii values of individual characters. This integer array is modified to store 2-digit ascii codes as two separate integers to avoid overflow in integer value. The ascii array is then printed. Next this ascii array is separated into two halves and stored in two integer arrays which are printed.

These two large integer halves are now added (arithmaticaly), and the result is sstored in another array. A carry variable is used to store the carry (if encountered) while adding the digits and is finally added to the last element of the addition array.

Next first eight 4-digit prime numbers are generated and stored in a prime array. The added large integer is then divided by each of the eight prime numbers and the reminder is strored and printed.

Note: Functions for checking prime and calculating reminder using modular arithmatic are defined separately.

Structured Pseudocode:

FUNCTION PRIMR(INT N):

FOR i=2 TO N-1 DO:

IF (N % i == 0)

RETURN 0

i = i\*i

RETURN 1

FUNCTION REMINDER(INT A, INT B, INT M):

A = A%M

RES = 1

WHILE B>0 DO:

IF(B IS ODD):

RES = RES\*A %M

A = A\*A %M

B = B/2

RETURN RES

MAIN():

INPUT NAME

INT ARRAY1[ ]

ARRAY[i] = NAME[i] FOR ALL CHARACTERS OF NAME

ASCII[2\*i] = ARRAY[i]/10 ASCII[2\*i +1] = ARRAY[i]%10

PRINT ASCII[j] FOR ALL i=0 TO i < LENGTH(ASCII)

STORE FIRST HALF OF ASCII IN ARRAY ‘A’

STORE SECOND HALF OF ASCII IN ARRAY ‘B’

PRINT A,B

FINAL[i] = (A[i] + B[i] + CARRY)%10 FOR ALL i=0 TO LENGTH(A)

CARRY = (A[LENGTH(A)-1] + B[LENGTH(A)-1] + CARRY)/10

PRINT FINAL

FOR i=1001 DO:

IF (COUNT<8)

IF (PRIME[i] == 1)

ARRAY[COUNT] = i

COUNT = COUNT +1

i = i+1

FOR i=0 TO 8 DO:

REM=0

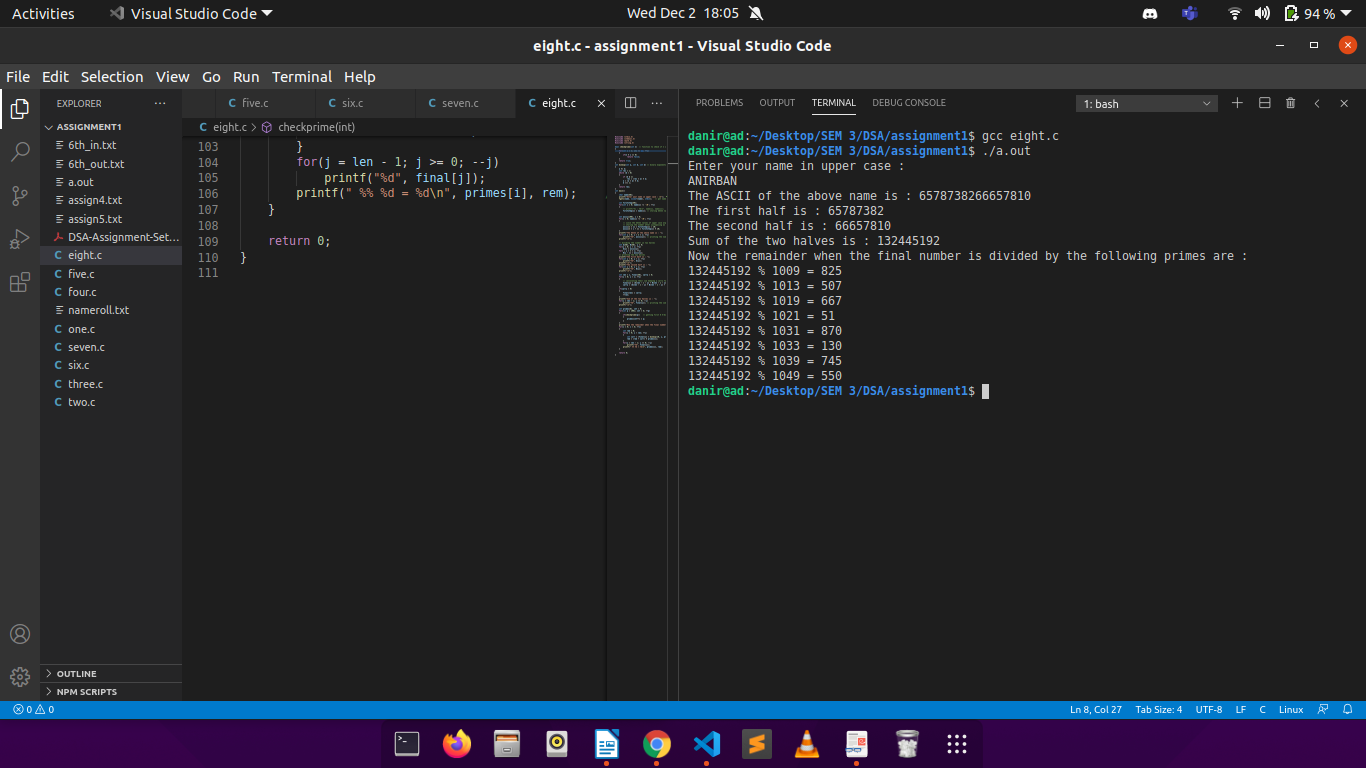
FOR j=0 TO LENGTH(FINAL) DO:

VAR = (FINAL[j] \* REMINDER(10, j, PRIMEi])) % PRIME[i]

REM = (REM+VAR) % PRIME[i]

PRINT PRIME[i], REM

Results:



Discussions:

Most of the functions use O(n) time complexity, the exceptoion being the modular exponentiation which uses O(log n). Therefore, in the worst case the complexity is O(n).

Source Code:

FILE NAME:

Code – “eight.c”

(can be found in the following link: <https://drive.google.com/drive/folders/1-nNb6aRleNLE1mcE58i85096fDmDUCvd?usp=sharing>)