Dayalbagh Educational Institute (Deemed University)
Agra – 282005

Faculty of Engineering

Academic Semester: July – December 2020

Virtual Laboratory Record

Course No. and Title: EEM 304 – C Lab
Course Teachers: Prof. C. Patvardhan & Dr. V. Prem Prakash

Submitted by:

Name	: Nij Mehar Grover
Roll No.	: 2002147
Branch	: Electrical

Contents

Date: 14 Aug 2021	Assignment No.: 01
Assignment Title: Assignment 01a	

1. Problem Statement

Write a function: int primetest(int NUM) that takes an integer NUM as argument and returns 0 if NUM is composite or 1 if NUM is PRIME. Inside main(), read a single integer value as input, test whether the number is prime or composite by calling the function primetest(NUM) above, and print PRIME or COMPOSITE accordingly.
Input: 1 int (NUM)
Output: “PRIME” (or) “COMPOSITE”

2. Solution Strategy

Run a for loop to check if the given number is divided by more than 2 numbers or not.

If it is then it’s a composite number and if it’s not divisible by more than 2 then it’s a prime number.

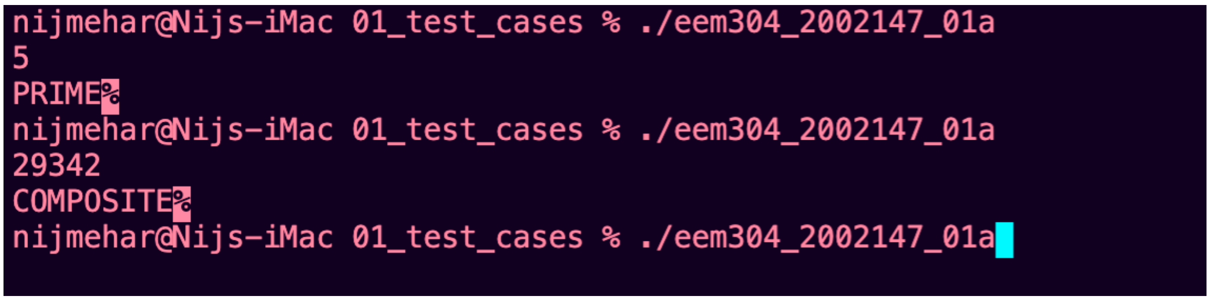
3. Algorithm/ Pseudo-code

```
Step 1: Start
Step 2: Read number n
Step 3: Set f=0
Step 4: For i=2 to n-1
Step 5: If n mod i=0 then
Step 6: Set f=1 and break
Step 7: Loop
Step 8: If f=0 then
          print 'The given number is prime'
        else
          print 'The given number is not prime'
Step 9: Stop
```

4. Source code of solution

```
#include<stdio.h>
int primetest(int NUM){
   int i,count=0;
   if(NUM==1 || NUM==0){
      return 2;
   }
   for(i=1;i<=NUM;i++){
      if(NUM%i==0)
      count++;
   }
   if(count==2)
   return 1;
   else
   return 0;
}
int main(){
   int num,opt;
   scanf("%d",&num);
   opt = primetest(num);
   if(opt==1)
   printf("PRIME");
   else if(opt==0)
   printf("COMPOSITE");
   return 0;
}
```

5. Screen-shots of program runs on test cases provided



```
nijmehar@Nijs-iMac 01_test_cases % ./eem304_2002147_01a
5
PRIME%
nijmehar@Nijs-iMac 01_test_cases % ./eem304_2002147_01a
29342
COMPOSITE%
nijmehar@Nijs-iMac 01_test_cases % ./eem304_2002147_01a
```

Date: 14 Aug 2021	Assignment No.: 01
Assignment Title: Assignment 01b	

1. Problem Statement

Write a function: `void sieve(int N)` that takes the value of N as argument and prints all prime numbers <= N (using the Sieve of Eratosthenes). Inside main(), read the value of N (an integer) and call sieve(N).

Input: 1 int (N)
Output: All prime numbers less than or equal to N

2. Solution Strategy

At the beginning we write down all numbers between 2 and n. We mark all proper multiples of 2 (since 2 is the smallest prime number) as composite. A proper multiple of a number, is a number greater than x and divisible by x. Then we find the next number that hasn't been marked as composite, in this case it is 3. Which means 3 is prime, and we mark all proper multiples of 3 as composite. The next unmarked number is 5, which is the next prime number, and we mark all proper multiples of it. And we continue this procedure until we processed all numbers in the row.

3. Algorithm/ Pseudo-code

Step 1: Start

Step 2: Read number *n*

Step 3: For i=0 to n-2

Step 4: Set prime[n-1] = 1

Step 5: Loop

Step 6: For i=0 to n-2

Step 7: if prime[I-2] = 1 then

```
    For j = I*I to n
    prime[j-2] = 0
```

```
        Loop
```

Step 8: Loop

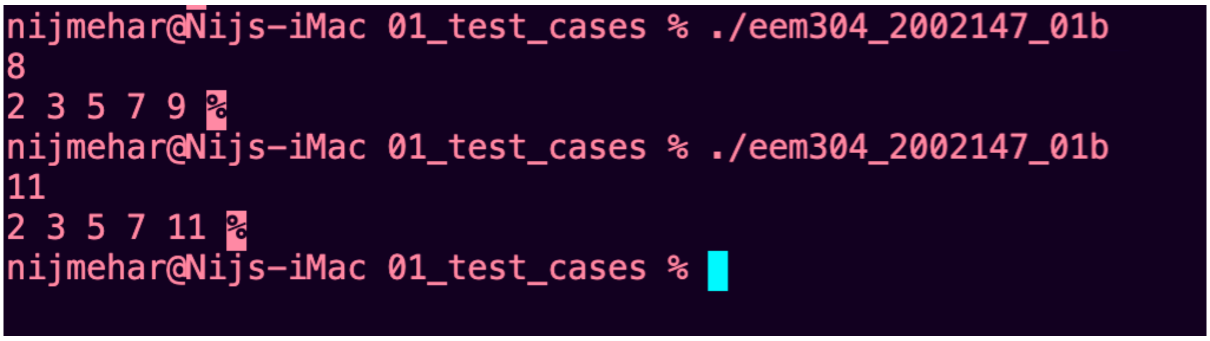
Step 9: Display

Step 10: Stop

4. Source code of solution

```c
#include<stdio.h>
void sieve(int N){
  int prime[N-1];
  int i,j;
  for(i=0;i<N-1;i++){
    prime[i]=1;
  }
  for(i=2;i<=N;i++){
    if(prime[i-2]==1){
      for(j=i*i;j<=N;j+=i){
        prime[j-2]=0;
      }
    }
  }
  for(i=0;i<N;i++){
    if(prime[i]==1)
    printf("%d ",i+2);
  }
}
int main(){
  int N;

  scanf("%d",&N);

  sieve(N);

  return 0;

}
```

5. Screen-shots of program runs on test cases provided



```
nijmehar@Nijs-iMac 01_test_cases % ./eem304_2002147_01b
8
2 3 5 7 9 %
nijmehar@Nijs-iMac 01_test_cases % ./eem304_2002147_01b
11
2 3 5 7 11 %
nijmehar@Nijs-iMac 01_test_cases %
```

Date: 14 Aug 2021	Assignment No.: 01
Assignment Title: Assignment 01c	

1. Problem Statement

Write a recursive function int factrec(int N) that takes an integer N as argument and recursively computes

and returns the value of N! Input: 1 int (N)
Output: 1 int (N!)

2. Solution Strategy

Use recursive functions to call the function factrec by passing a lesser value of N and multiplying the return with the recursive call.

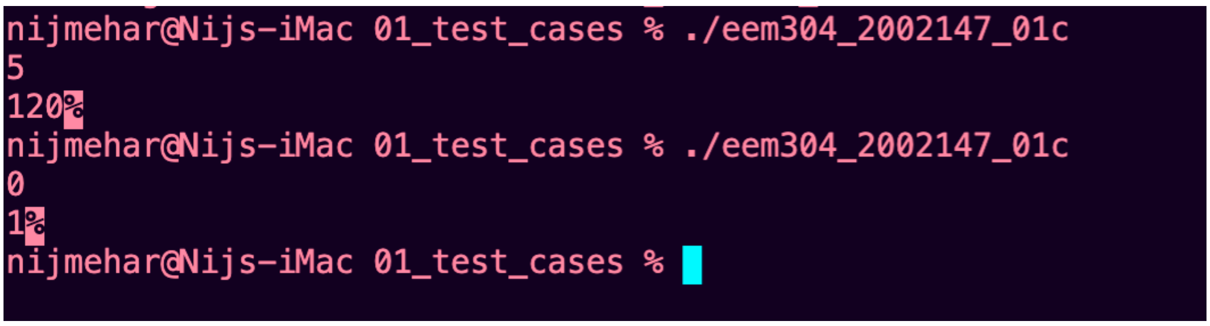
3. Algorithm/ Pseudo-code

```
Step 1: Start
Step 2: Read number N
Step 3: factrec(N)
Step 4: if N!=0
        return N * factrec(N-1)
        else
        return 1
Step 5: Stop
```

4. Source code of solution

```
#include<stdio.h>
int factrec(int N){
   if(N!=0)
   return N*factrec(N-1);
   else
   return 1;
}
int main(){
   int N;
   scanf("%d",&N);
   printf("%d",factrec(N));
}
```

5. Screen-shots of program runs on test cases provided



```
nijmehar@Nijs-iMac 01_test_cases % ./eem304_2002147_01c
5
120%
nijmehar@Nijs-iMac 01_test_cases % ./eem304_2002147_01c
0
1%
nijmehar@Nijs-iMac 01_test_cases %
```

Date: 22 Aug 2021	Assignment No.: 02
Assignment Title: Assignment 02a	

1. Problem Statement

Write a function: void insertionsort(int a[], int N, int choice)that reads N ints and sorts them using insertion sort in ascending or descending order, depending on what the user specifies (an additional integer variable that is 0 for ascending and 1 for descending order). The function should print the sorted list, with numbers separated by a single space.

Your program should read N first, then choice, then N integer elements (in that order) and then call the insertionsort() function above.

Input: int N (number of elements), int choice, N integer elements

Output: sorted list, separated by single spaces.

Example:

Input: 5 1 2 8 -3 0 1

Output:8 2 1 0 -3

2. Solution Strategy

Begin at the left-most element of the array and invoke Insert to insert each element encountered into its correct position. The ordered sequence into which the element is inserted is stored at the beginning of the array in the set of indices already examined.

3. Algorithm/ Pseudo-code

Step 1 - If the element is the first element, assume that it is already sorted. Return 1.

Step2 - Pick the next element, and store it separately in a **key.**

Step3 - Now, compare the **key** with all elements in the sorted array.

Step 4 - If the element in the sorted array is smaller/larger than the current element, then move to the next element. Else, shift greater/smaller elements in the array towards the right.

Step 5 - Insert the value.

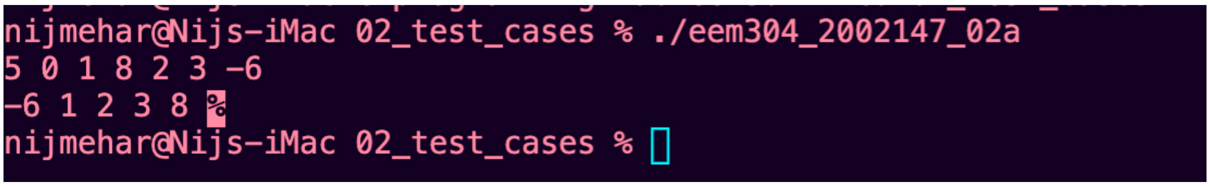
Step 6 - Repeat until the array is sorted.

4. Source code of solution

```
#include<stdio.h>
void insertionsort(int a[],int N,int choice){
   int i,key,j;
   for(i=0;i<N;i++){
      key = a[i];
      j=i-1;
      if(choice == 0){
      while(j>=0 && a[j]>key){
         a[j+1]=a[j];
         j-=1;
      }}
      else if(choice == 1){
         while(j>=0 && a[j]<key){
         a[j+1]=a[j];
         j-=1;
      }
      }
      a[j+1]=key;
   }
   for(i=0;i<N;i++){
      printf("%d ",a[i]);
   }
}
int main(){
   int size,opt;
   scanf("%d",&size);
```

```
    scanf("%d",&opt);

    int i,arr[size];

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

        scanf("%d",&arr[i]);

    }

    insertionsort(arr,size,opt);

    return 0;

}
```

5. Screen-shots of program runs on test cases provided



```
nijmehar@Nijs-iMac 02_test_cases % ./eem304_2002147_02a
5 0 1 8 2 3 -6
-6 1 2 3 8 %
nijmehar@Nijs-iMac 02_test_cases % 
```

Date: 22 Aug 2021	Assignment No.: 02
Assignment Title: Assignment 02b	

1. Problem Statement

Similarly write two more functions, one each for selectionsort and bubblesort, in another source file (...02b.c):

void selectionsort(int a[], int N, int choice) void bubblesort(int a[], int N, int choice)

Add the insertionsort() function from 02a to this file, and run all three functions on different input sizes. Plot the array size (x axis) versus execution time (y axis) for all three algorithms, increasing the array size by multiples of 10, starting from 100 elements up to the largest possible array size supported by your programming environment (100, 1000, 10000, 100000, and so on). You may use MS Excel or any other software for generating the graph. If you don't have access to any such software, draw the graph to scale on an A4 sheet. Submit the graph as a JPG or PNG image. Ensure that image size is well within the permitted submission limit (500 KB).

2. Solution Strategy

Implement insertion sort, selection sort, bubble sort.

3. Algorithm/ Pseudo-code

Insertion sort -

Step 1 - Assume that first element in the list is in sorted portion and all the remaining elements are in unsorted portion.

Step 2: Take first element from the unsorted portion and insert that element into the sorted portion in the order specified.

Step 3: Repeat the above process until all the elements from the unsorted portion are moved into the sorted portion

Selection sort -

Step 1 - Select the first element of the list (i.e., Element at first position in the list).

Step 2: Compare the selected element with all the other elements in the list.

Step 3: In every comparision, if any element is found smaller than the selected element (for Ascending order), then both are swapped.

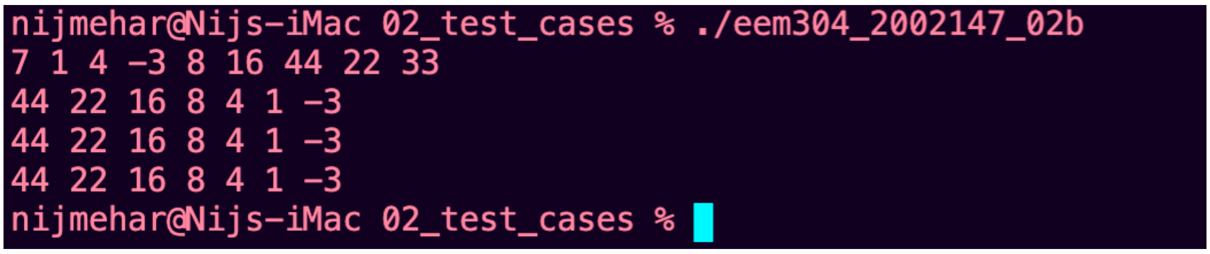
Step 4: Repeat the same procedure with element in the next position in the list till the entire list is sorted.

4. Source code of solution

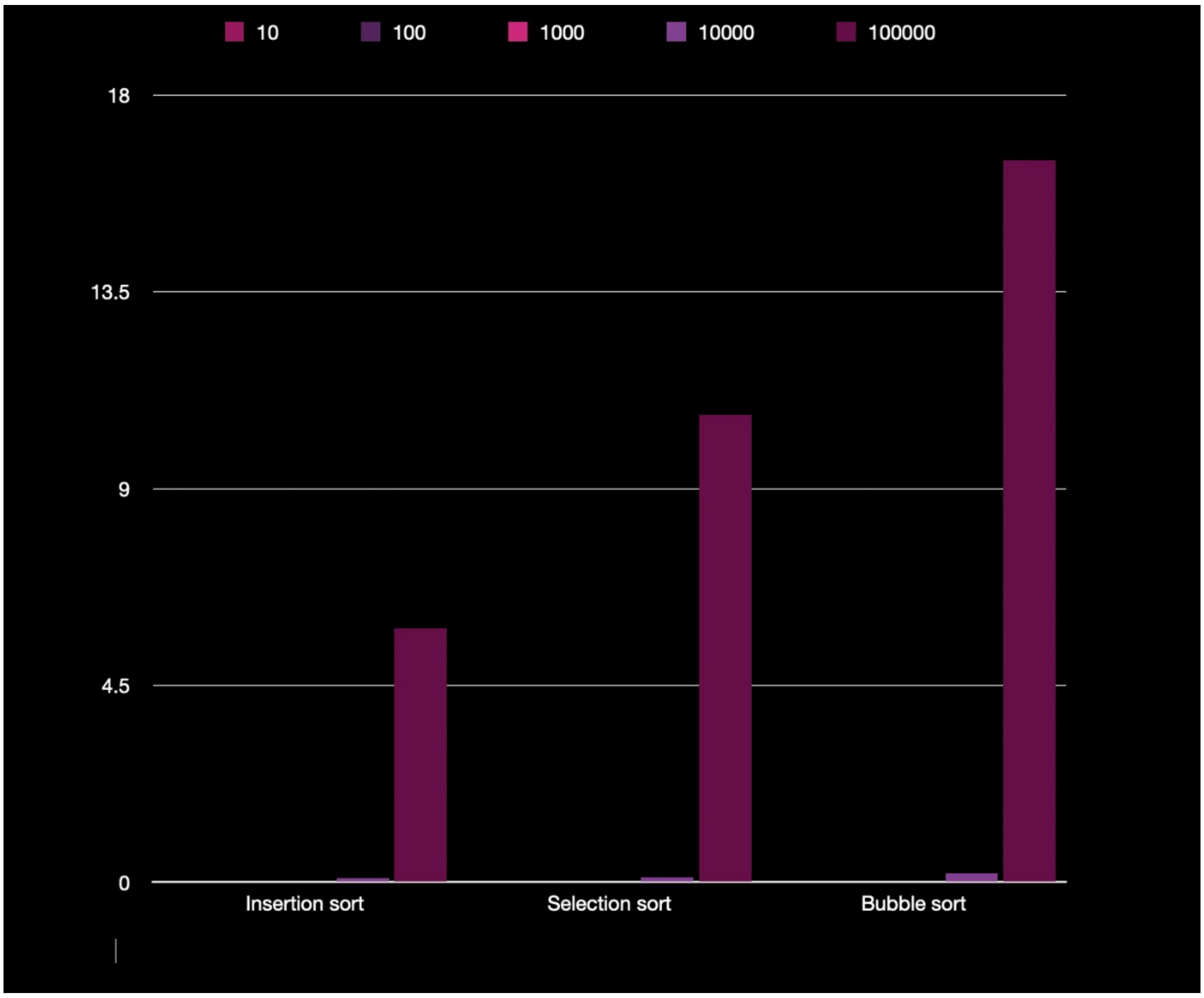
```
#include<stdio.h>
#include <stdlib.h>
void insertionsort(int a[],int N){
    int i,key,j;
   for(i=0;i<N;i++){
      key = a[i];
      j=i-1;

      while(j>=0 && a[j]>key){
         a[j+1]=a[j];
         j-=1;
      }

      a[j+1]=key;
   }
   for(i=N-1;i>=0;i--){
      printf("%d ",a[i]);
   }
   printf("\n");
}
void selectionsort(int a[],int N){
   int def;
   int i,j,k;
   for(j=0;j<N;j++){
      def=a[j];
      k=j;
      for(i=j+1;i<N;i++){
       if(a[i]<def){
          def=a[i];
          k=i;
       }
      }
      a[k] = a[j];
      a[j] = def;
   }
   for(i=N-1;i>=0;i--){
      printf("%d ",a[i]);
   }
   printf("\n");

}
void bubblesort(int a[],int N){
   int i,def,x=1;
   while(x!=0)
   {
      x=0;
      for(i=0;i<N-1;i++){
      if(a[i]>a[i+1]){
         def = a[i+1];
         a[i+1] = a[i];
         a[i] = def;
         x++;
       }
      }
```

```
    }
     for(i=N-1;i>=0;i--){
        printf("%d ",a[i]);
    }
    printf("\n");
}
int main(){
    int size,i,x;
    scanf("%d",&size);
    int arr[size];
    for(i=0;i<size;i++){
        scanf("%d",&x);
        arr[i] = x;
    }
    insertionsort(arr,size);
    selectionsort(arr,size);
    bubblesort(arr,size);
    return 0;
}
```

5. Screen-shots of program runs on test cases provided



```
nijmehar@Nijs-iMac 02_test_cases % ./eem304_2002147_02b
7 1 4 -3 8 16 44 22 33
44 22 16 8 4 1 -3
44 22 16 8 4 1 -3
44 22 16 8 4 1 -3
nijmehar@Nijs-iMac 02_test_cases % 
```



10
100
1000
10000
100000
18
13.5
9
4.5
0
Insertion sort
Selection sort
Bubble sort

Date: 5 Sept 2021	Assignment No.: 03
Assignment Title: Assignment 03a	

1. Problem Statement

Matrix Multiplication. Write the following two functions:

void matmult(int A[][MAX], int B[][MAX], int C[][MAX], int P, int Q, int R)

(A function that multiplies two matrices, APxQ and BQxR and stores the result in matrix CPxR)

void display(int A[][MAX], int rows, int cols)

(A function that prints the contents of matrix Arows x cols row by row, with a single space between elements within a row)

Your program should be able to support up to 500x500 (MAX=500) element arrays (Assume this to be the maximum size of array that we will be testing the program on). The program will read the values of P, Q and R, followed by PQ + QR elements from the keyboard. All these would be stored in the input files provided. These elements would be read into arrays A and B respectively. The program should check if the dimensions of the two matrices are compatible for the operation. Appropriate error message should be printed in case either of these conditions fails. The program should call the **`matmult()`** function to perform the multiplication operation C=AxB, and then display the matrix C using the **`display()`** function.

2. Solution Strategy

we keep on iterating the loop and perform the matrix multiplication like we do in mathematics.

C[i][j] += A[i][k]*B[k][j];

3. Algorithm/ Pseudo-code

Step 1: Start

Step 2: Read number *P, R, Q*

Step 3: Read arrays A, B, C

Step 4: for I = 0 to P-1

Step 5: for j = 0 to R-1

Step 6: for k = 0 to Q-1

Step 7: C[i][j] += A[I][k]*B[k][j]

Step 8: Loop

Step 9: Loop

Step 10: Loop

Step 11: Stop

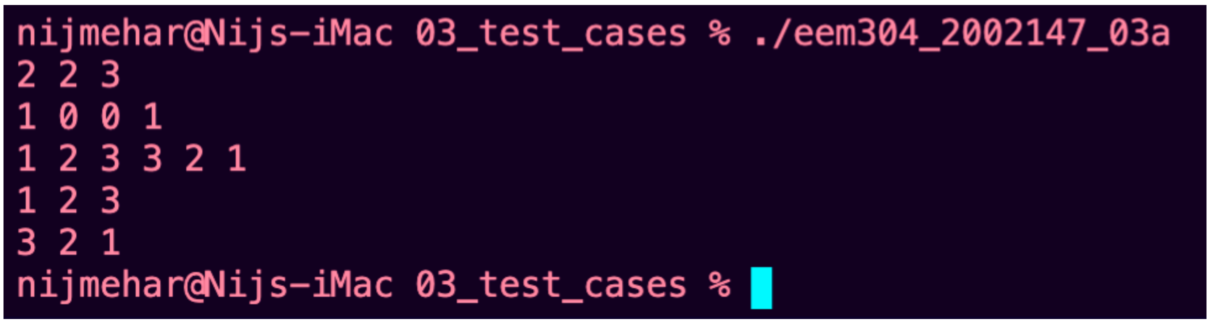
4. Source code of solution

```
#include<stdio.h>
#define MAX 500

void matmult(int A[][MAX],int B[][MAX],int P,int Q,int R);
void display(int A[][MAX], int rows, int cols);
int main(){
  int P,Q,R,i,j;
  scanf("%d %d %d",&P,&Q,&R);
  int A[P][MAX],B[Q][MAX];
  for(i=0;i<P;i++){
    for(j=0;j<Q;j++){
      scanf("%d",&A[i][j]);
    }
  }
  for(i=0;i<Q;i++){
  for(j=0;j<R;j++){
      scanf("%d",&B[i][j]);
    }
  }
  if(P>500 || Q>500 || R>500)
    {printf("enter array size less than 500");
```

```
        return 0;
     }
     matmult(A,B,P,Q,R);
     return 0;
}

void matmult(int A[][MAX],int B[][MAX],int P,int Q,int R){
     int i,j,k;
     int C[P][MAX];
     for(i=0;i<P;i++){
        for(j=0;j<R;j++){
              C[I][j] = 0;
           for(k=0;k<Q;k++){
               C[i][j] += A[i][k]*B[k][j];
           }
        }
     }
     display(C,P,R);
}

void display(int C[][MAX], int rows, int cols){
     int i=0,j=0;
     for(i=0;i<rows;i++)
     {
     for(j=0;j<cols;j++){
           printf("%d ",C[i][j]);
        }
        printf("\n");
```

```
   }
}
```

5. Screen-shots of program runs on test cases provided



```
nijmehar@Nijs-iMac 03_test_cases % ./eem304_2002147_03a
2 2 3
1 0 0 1
1 2 3 3 2 1
1 2 3
3 2 1
nijmehar@Nijs-iMac 03_test_cases %
```

Date: 14 Aug 2021	Assignment No.: 03
Assignment Title: Assignment 03b	

1. Problem Statement

Sparse Matrix Addition. Write a program that reads two sparse matrices, adds them and display the resulting sum matrix. Input will be provided in the test files in the following order:

nrowsA ncolsA nelemsA nrowsB ncolsB nelemsB row1A col1A val1A row2A col2A … row1B col1B val1B row2B col2B … (single space separated). Assume that no source matrix (A or B) may contain more than 100 entries.

2. Solution Strategy

implement sparse matrix focusing only on the given elements and dropping out the 0s.

3. Algorithm/ Pseudo-code

```
Step 1 - Begin.
Step 2 - n = 1.
Step 3 - ra = rb = rc = ca = cb = cc = 1.
Step 4 - Repeat while (ra <= m)
Step 5 - if(a[ra] == 0 AND b[rb] ==0)
Step 6 - c[rc++] = 0.
Step 7 - end if.
Step 8 - else if(a[ra] == 0)
```

4. Source code of solution

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>
```

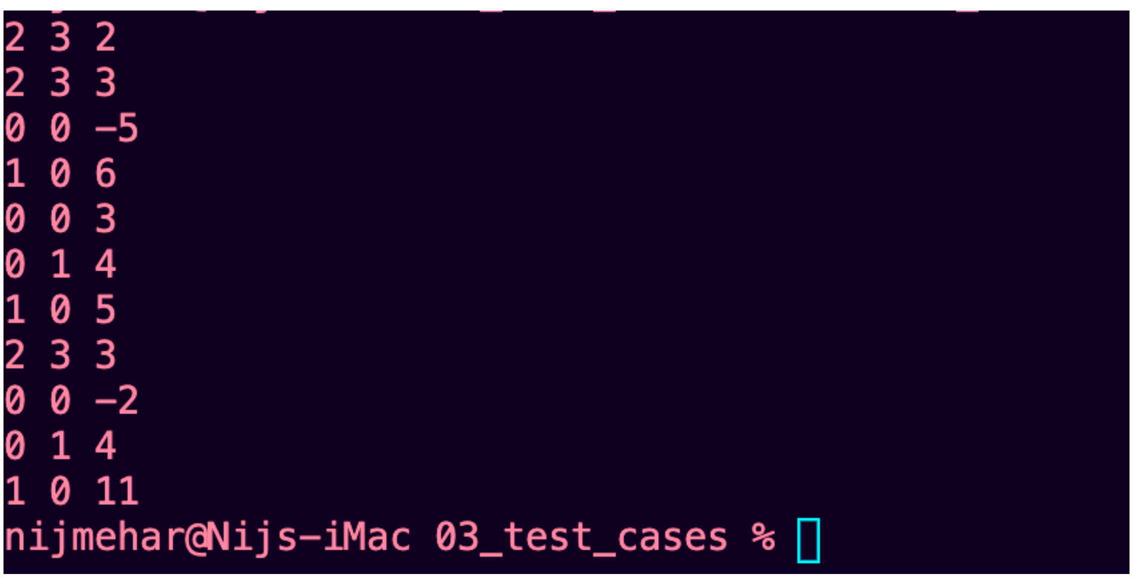
```
struct sparceMatrix
{
   int row;
   int col;
   int el;
}
a[100],b[100],c[100];
int main()
{
  int rowA=0,colA=0,la=0,rowB=0,colB=0,lb=0,i,pos1=0,pos2=0,pos3=0;
  scanf("%d",&rowA);
  scanf("%d",&colA);
  scanf("%d",&la);
  scanf("%d",&rowB);
  scanf("%d",&colB);
  scanf("%d",&lb);
  for(i=0;i<la;i++)
  {
     scanf("%d",&a[i].row);
     scanf("%d",&a[i].col);
     scanf("%d",&a[i].el);
  }
  for(i=0;i<lb;i++)
  {
     scanf("%d",&b[i].row);
     scanf("%d",&b[i].col);
     scanf("%d",&b[i].el);
  }
```

```
if(rowA==rowB && colA==colB)
{
 while(la>pos1 && lb>pos2)
 {
    if(a[pos1].row<b[pos2].row || (a[pos1].row==b[pos2].row && a[pos1].col<b[pos2].col) )
    {
        c[pos3].row=a[pos1].row;
        c[pos3].col=a[pos1].col;
        c[pos3].el=a[pos1].el;
        pos3++;
        pos1++;
    }
    else if(a[pos1].row>b[pos2].row || (a[pos1].row==b[pos2].row && a[pos1].col>b[pos2].col) )
    {
        c[pos3].row=b[pos2].row;
        c[pos3].col=b[pos2].col;
        c[pos3].el=b[pos2].el;
        pos3++;
        pos2++;
    }
    else
    {
        c[pos3].row=b[pos2].row;
        c[pos3].col=b[pos2].col;
        c[pos3].el=a[pos1].el+b[pos2].el;
        pos3++;
        pos2++;
        pos1++;
```

```
   }
}
while(la>pos1)
{
   c[pos3].row=a[pos1].row;
   c[pos3].col=a[pos1].col;
   c[pos3].el=a[pos1].el;
   pos3++;
   pos1++;
}
while(lb>pos2)
{
   c[pos3].row=b[pos2].row;
   c[pos3].col=b[pos2].col;
   c[pos3].el=b[pos2].el;
   pos3++;
   pos2++;
}

printf("%d ",rowA);
printf("%d ",colA);
printf("%d ",pos3);
printf("\n");
for(i=0;i<pos3;i++)
{
   printf("%d ",c[i].row);
   printf("%d ",c[i].col);
   printf("%d ",c[i].el);
```

```
        printf("\n");
      }
    }
    else
    return 0;
}
```

5. Screen-shots of program runs on test cases provided



```
2 3 2
2 3 3
0 0 -5
1 0 6
0 0 3
0 1 4
1 0 5
2 3 3
0 0 -2
0 1 4
1 0 11
nijmehar@Nijs-iMac 03_test_cases %
```

Date: 14 Aug 2021	Assignment No.: 04
Assignment Title: Assignment 04a	

1. Problem Statement

Towers of Hanoi using recursion. Given three poles A, B and C, and a stack on n disks of increasing diameters, numbered 1, 2, 3 … respectively mounted on pole A, move this stack to pole C, obeying the following rules:

1) Only one disk is to be moved at a time. More than one disk cannot be picked up simultaneously.

2) A move consists of taking the uppermost disk from one of the stacks and placing it on top of another stack. So a disk can only be moved if it is the uppermost disk on a stack.

3) No disk may be placed on top of a smaller disk.

You will write a recursive function for this purpose, to be called from main(), which will read the value of n redirected from input test file (plus additional arguments for the source/dest/aux poles) and print a string of moves representing the correct sequence of <disk#,sourcepole,destpole> moves that represent the solution.

Example 1:

input: 1

output: 1AC

Example 2:

input: 2

output: 1AB2AC1BC

Example 3:

input: 4

output: 1AB2AC1BC3AB1CA2CB1AB4AC1BC2BA1CA3BC1AB2AC1BC

2. Solution Strategy

use recursion to implement tower of hanoi

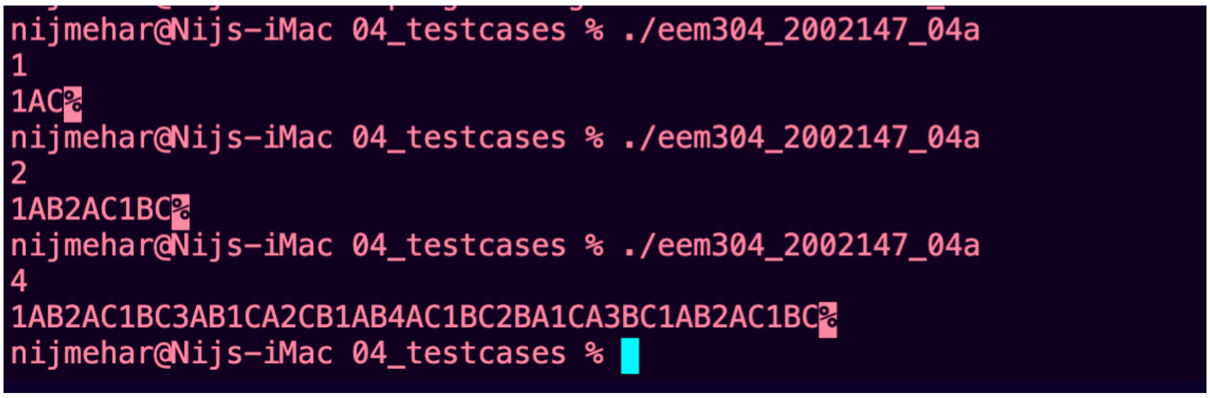
3. Algorithm/ Pseudo-code

```
Step 1 – Move n-1 disks from source to aux
Step 2 – Move nth disk from source to dest
Step 3 – Move n-1 disks from aux to dest
```

4. Source code of solution

```
#include<stdio.h>
void hanoi(int n,char a,char c,char b){
   if(n==0)
   return;
   hanoi(n-1,a,b,c);
   printf("%d%c%c",n,a,c);
   hanoi(n-1,b,c,a);
}
int main(){
   int n;
   scanf("%d", &n);
   hanoi(n,'A','C','B');
   return 0;
}
```

5. Screen-shots of program runs on test cases provided



```
nijmehar@Nijs-iMac 04_testcases % ./eem304_2002147_04a
1
1AC%
nijmehar@Nijs-iMac 04_testcases % ./eem304_2002147_04a
2
1AB2AC1BC%
nijmehar@Nijs-iMac 04_testcases % ./eem304_2002147_04a
4
1AB2AC1BC3AB1CA2CB1AB4AC1BC2BA1CA3BC1AB2AC1BC%
nijmehar@Nijs-iMac 04_testcases %
```

Date: 14 Aug 2021	Assignment No.: 04
Assignment Title: Assignment 04b	

1. Problem Statement

Recursive Sum-of-Squares Write a recursive function `findSum()` that takes an array A of integers and two indices (lo and hi, lo <=hi) as arguments and returns (A[lo]2+ A[lo+1]2… A[hi]2).

int findSum(int A[], int lo, int hi)

Your program will receive as input the value of N (the number of array elements) followed by N array elements separated by single spaces, followed by hi and lo index values, separated by single spaces as before. The output of your program should be a single integer that represents the sum of squares of the array elements between indices lo and hi.

Example.

Input: 5 1 2 3 4 5 2 4

Output: 50

Explanation:

5 1 2 3 4 5 **2 4**

5 is the number of elements in the array

1 2 3 4 5 are the array elements

2 is the lo index and 4 is the hi index

So the function computes 32+42+52 = 50 and returns this value.

2. Solution Strategy

Iterate through lo to hi and keep adding the sum of squares of that index in array to a counter variable.

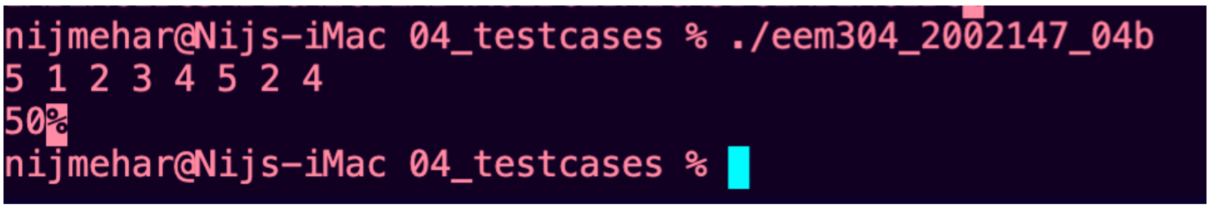
3. Algorithm/ Pseudo-code

```
Step 1: Start
Step 2: Read n and A[n]
Step 3: findsum(A[],lo,hi)
Step 4: if lo<=hi
            return (A[lo]*A[lo]) + (findSum(A,lo+1,hi))
            else
            return 0
Step 5: Stop
```

4. Source code of solution

```
#include<stdio.h>
int findSum(int A[],int lo,int hi){
   if(lo<=hi)
   return (A[lo]*A[lo]) + (findSum(A,lo+1,hi));
   else
   return 0;
}
int main(){
   int n;
   scanf("%d",&n);
   int A[n];
   int i;
   for( i=0;i<n;i++)
   {
      scanf("%d",&A[i]);
   }

   int lo,hi;

   scanf("%d %d",&lo,&hi);

    if(lo>hi)

   return 0;

   if(lo>=n || hi>=n)

   return 0;

   printf("%d",findSum(A,lo,hi));

   return 0;

}
```

5. Screen-shots of program runs on test cases provided



```
nijmehar@Nijs-iMac 04_testcases % ./eem304_2002147_04b
5 1 2 3 4 5 2 4
50%
nijmehar@Nijs-iMac 04_testcases %
```

Date: 2 Oct 2021	Assignment No.: 05
Assignment Title: Assignment 05	

1. Problem Statement

Write push(), pop(), isFull() and isEmpty() functions for an integer stack. Create a menu-driven program that repeatedly takes one of the following four options as input: 1, 2 ,3 or 4 for PUSH, POP, DISPLAY and EXIT respectively, and performs the appropriate sequence of actions. As discussed in the class, you must ensure that there are no extraneous printf statements in your code. Note that DISPLAY prints stack elements in order from A[0] up to A[TOP]. Test input will comprise of a series of PUSH/POP followed by DISPLAY and EXIT. Any attempts to PUSH onto a full stack or POP from an empty stack should be simply ignored by your program, i.e., it should not even print any error message. Assume that the maximum stack size is 10 elements.

Sample input:
1 10 1 20 1 30 2 3 4
(Meaning: Push 10, Push 20, Push 30, Pop, Display Exit) Output: 10 20

2. Solution Strategy

Use recursive functions to call the function factrec by passing a lesser value of N and multiplying the return with the recursive call.

3. Algorithm/ Pseudo-code

push(value)

Step 1 - Check whether stack is FULL. (top == SIZE-1)

Step 2 - If it is FULL, then display "Stack is FULL!!! Insertion is not possible!!!" and terminate the function.

Step 3 - If it is NOT FULL, then increment top value by one (top++) and set stack[top] to value (stack[top] = value).

pop()

Step 1 - Check whether stack is EMPTY. (top == -1)

Step 2 - If it is EMPTY, then display "Stack is EMPTY!!! Deletion is not possible!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, then delete stack[top] and decrement top value by one (top--).

display()

Step 1 - Check whether stack is EMPTY. (top == -1)

Step 2 - If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, then define a variable 'i' and initialize with top. Display stack[i] value and decrement i value by one (i--).

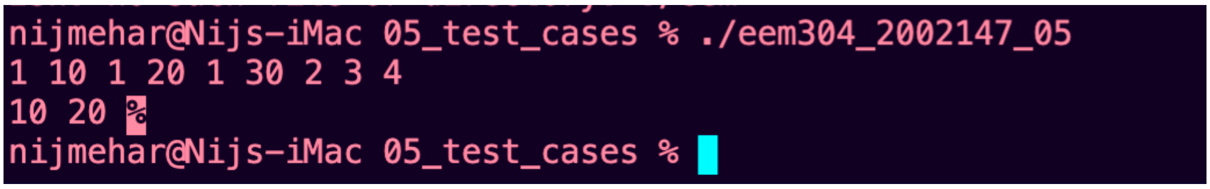
Step 4 - Repeat above step until i value becomes '0'.

4. Source code of solution

```
#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

struct Stack {

   int top;

   unsigned capacity;

   int* array;

};

 struct Stack* createStack(unsigned capacity)

{

   struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));

   stack->capacity = capacity;

   stack->top = -1;

   stack->array = (int*)malloc(stack->capacity * sizeof(int));

   return stack;

}


int isFull(struct Stack* stack)

{

   return stack->top == stack->capacity - 1;

}
```

```
int isEmpty(struct Stack* stack)

{

   return stack->top == -1;

}

void push(struct Stack* stack, int item)

{

   if (!isFull(stack))

   stack->array[++stack->top] = item;


}

void pop(struct Stack* stack)

{

   if (!isEmpty(stack))

   stack->top--;

}

void display(struct Stack* stack)

{

  int i = 0;

   while(i<=stack->top){

   printf("%d ",stack->array[i++]);


   }

}

int main()

{

   struct Stack* stack = createStack(10);

   int n,t;

   while(1){
        scanf("%d",&t);

        if(t==1){

            scanf("%d",&n);

            push(stack,n);

        }

        else if(t==2){

            pop(stack);

        }

        else if(t==3)

        display(stack);

        else if(t==4)

        break;

    }

    return 0;

}
```

5. Screen-shots of program runs on test cases provided



```
nijmehar@Nijs-iMac 05_test_cases % ./eem304_2002147_05
1 10 1 20 1 30 2 3 4
10 20 %
nijmehar@Nijs-iMac 05_test_cases % 
```

Date: 23 Oct 2021	Assignment No.: 06
Assignment Title: Assignment 06	

1. Problem Statement

Create a singly linked list of integer values. Implement functions for inserting at the head of the list, inserting an element before or after another specified element in the list, deleting a given element and simple list traversal. Use the C template (`Asg06_template.c`) given to you for this purpose. DO NOT MAKE ANY CHANGES to the `main()` program and the structure definitions given in the template. Your job in this assignment is only to implement the functions declared and used in this program. The behaviour of the program is also clearly explained in the associated video. Please go through it and carefully implement your solution in the same template file, rename it properly and submit.

Note: To prevent confusion in the output portion, I've also provided an implementation of the traverse function. So you no longer have not implement the traverse function as mentioned in the video. The code will directly use the traverse () function provided in the template.

2. Solution Strategy

3. Algorithm/ Pseudo-code

Insertafter()

Step 1 - Create a newNode with given value.

Step 2 - Check whether list is Empty (head == NULL)

Step 3 - If it is Empty then, set newNode → next = NULL and head = newNode.

Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.

Step 5 - Keep moving the temp to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode).

Step 6 - Every time check whether temp is reached to last node or not. If it is reached to last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp to next node.

Step 7 - Finally, Set 'newNode → next = temp → next' and 'temp → next = newNode' deleteitem()

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.

Step 4 - Keep moving the temp1 until it reaches to the exact node to be deleted or to the last node. And every time set 'temp2 = temp1' before moving the 'temp1' to its next node.

Step 5 - If it is reached to the last node then display 'Given node not found in the list! Deletion not possible!!!'. And terminate the function.

Step 6 - If it is reached to the exact node which we want to delete, then check whether list is having only one node or not

Step 7 - If list has only one node and that is the node to be deleted, then set head = NULL and delete temp1 (free(temp1)).

Step 8 - If list contains multiple nodes, then check whether temp1 is the first node in the list (temp1 == head).

Step 9 - If temp1 is the first node then move the head to the next node (head = head → next) and delete temp1.

Step 10 - If temp1 is not first node then check whether it is last node in the list (temp1 → next == NULL).

Step 11 - If temp1 is last node then set temp2 → next = NULL and delete temp1 (free(temp1)).

Step 12 - If temp1 is not first node and not last node then set temp2 → next = temp1 → next and delete temp1 (free(temp1)).

traverse()

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!!' and terminate the function.

Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with head.

Step 4 - Keep displaying temp → data with an arrow (--->) until temp reaches to the last node

Step 5 - Finally display temp → data with arrow pointing to NULL (temp → data ---> NULL).

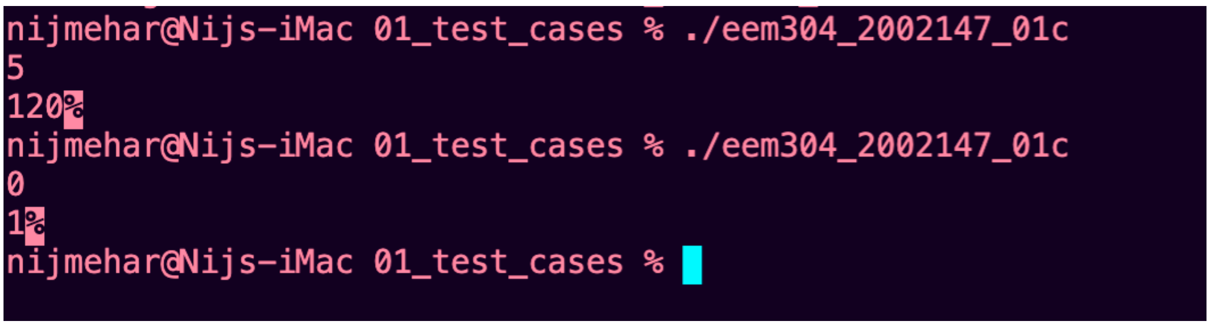
4. Source code of solution

```
#include <stdio.h>
#include <stdlib.h>
void insertAtHead(int no);
void insertAfter(int itemafter,int after);
void insertBefore(int itembefore,int before);
void deleteItem(int itemDelete);
void traverse (void);

 typedef struct node_tag
 {
    int val;
    struct node_tag * next;
 } node;

 node *head;

 int main ()
 {
head=(node*)malloc(sizeof(node));
 int i,n, before, after, itemBefore, itemAfter, itemDelete;
 void insertAtHead(int n);
 void insertBefore (int itemBefore, int before);
 void insertAfter (int itemAfter, int after);
 void traverse(void);
 void deleteItem(int itemDelete);
 scanf("%d",&n);

 scanf("%d",&itemBefore);

 scanf("%d",&before);

 scanf("%d",&itemAfter);

 scanf("%d",&after);

 scanf("%d",&itemDelete);

 insertAtHead(0);

 for (i = 2; i<=n; i+=2)

     insertAfter(i,i-2);


insertAfter(itemAfter, after);

insertBefore(itemBefore, before);

deleteItem(itemDelete);

traverse();


 return 0;

}

void insertAtHead(int no)

{

    head->val=no;

    head->next=NULL;

}

void insertAfter(int itemafter,int after)

{   node *p;

    node *nn;

    node *q;

    nn=( node*)malloc(sizeof(node));

    p=head;
```

```
   while(p!=NULL)

   {


   if(p->val==after)

      {

         nn->val=itemafter;

         q=p->next;

         if(q!=NULL)

            nn->next=q;

         else

            nn->next=NULL;

   p->next=nn;


      }

   p=p->next;


   }

}

void insertBefore(int itembefore,int before)

{

   int i=0,n=0;

   node *p;

   node *nn;

   node *q;

   nn=( node*)malloc(sizeof(node));

   p=head;

   q=head;

   while(p!=NULL)
```

```
    {
        n++;


    if(p->val==before && before!=0)
    {
        while(i<n-2)
        {
            i++;
            q=q->next;
        }
        nn->val=itembefore;
        nn->next=p;
        q->next=nn;
    }
    else if(before==0 && n==1)
    {
        nn->val=itembefore;
        nn->next=p;
        head=nn;
    }
    p=p->next;


    }
}
void deleteItem(int itemDelete)
{
    int i=1,n=1;
    node *temp;
    node *cr;
    node *news;
    temp=head;
    cr=head;
    while(temp!=NULL)
    {
        n++;
        if(temp->val==itemDelete && itemDelete!=0)
        {
          while(i<n-2)
          {
              i++;
              cr=cr->next;
          }
          news=temp->next;
          cr->next=news;
        }
        else if(itemDelete==0 && n==2)
        {
        cr=temp->next;
        head=cr;
        }
        temp=temp->next;
    }
}
void traverse (void)
{
    node * n = head;
    while (n)

    {

        printf("->%d",n->val);

        n = n->next;

    }

}
```

5. Screen-shots of program runs on test cases provided



```
nijmehar@Nijs-iMac 01_test_cases % ./eem304_2002147_01c
5
120%
nijmehar@Nijs-iMac 01_test_cases % ./eem304_2002147_01c
0
1%
nijmehar@Nijs-iMac 01_test_cases % 
```

Date: 14 November 2021	Assignment No.: 07
Assignment Title: Assignment 07	

1. Problem Statement

Create a linked stack of city names. For this purpose, the template provides you with a structure called node, which has two elements: a char pointer and a next pointer.

Implement functions push() and pop() for pushing and popping respectively, from the list. The push() function will take two arguments, the address of the head of the list, and the city name to be pushed. The pop() function will remove an free the top element of the list, and return the number of characters of the city name stored in that element

2. Solution Strategy

use pointers to implement linked stack.

3. Algorithm/ Pseudo-code

Push()

Step 1 - Create a newNode with given value.

Step 2 - Check whether stack is Empty (top == NULL)

Step 3 - If it is Empty, then set newNode → next = NULL.

Step 4 - If it is Not Empty, then set newNode → next = top.

Step 5 - Finally, set top = newNode.

Pop()

Step 1 - Check whether stack is Empty (top == NULL).

Step 2 - If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the function

Step 3 - If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.

Step 4 - then set 'top = top → next'.

Step 5 - Finally, delete 'temp'. (free(temp)).

Display()

Step 1 - Check whether stack is Empty (top == NULL).

Step 2 - If it is Empty, then display 'Stack is Empty!!!' and terminate the function.

Step 3 - If it is Not Empty, then define a Node pointer 'temp' and initialize with top.

Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches to the first node in the stack. (temp → next != NULL).

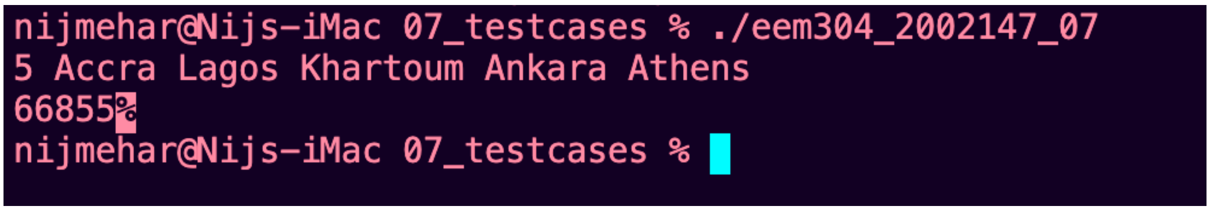
Step 5 - Finally! Display 'temp → data ---> NULL'.

4. Source code of solution

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define SIZE 20


typedef struct node_tag {

   char city[SIZE];

   struct node_tag *next;

} node;



int main () {


 node* head=NULL;

 int i,n;

 char item[SIZE];


 void push(node **, char []);
 int pop (node **);


 scanf("%d",&n);
 for (i=0; i<n; i++) {
    scanf("%s", item);
    push(&head, item);
 }

for (i=0; i<n; i++)
    printf("%d",pop(&head));

 return 0;
}

void push(node **head, char item[]) {
   node* newNode = (node*)malloc(sizeof(node));
   if(*head == NULL){
      strcpy(newNode->city, item);
      newNode->next = NULL;
      *head = newNode;
   }
   else{
      strcpy(newNode->city, item);
      newNode->next = *head;
      *head = newNode;
   }
}
```

```
int pop(node ** head) {
  if (*head == NULL) {
      return -1;
  }
  node* hd = *head;
  char *c = hd->city;      // pull out data before the node is deleted
  int x = strlen(c);
  (*head) = (*head)->next;
  hd->next = NULL; // unlink the head node for the caller
                              // Note the `*` — uses a reference-pointer
                              // just like `push()` and `deleteList()`.
  free(hd);            // free the head node
  return x;


}
```

5. Screen-shots of program runs on test cases provided



```
nijmehar@Nijs-iMac 07_testcases % ./eem304_2002147_07
5 Accra Lagos Khartoum Ankara Athens
66855%
nijmehar@Nijs-iMac 07_testcases %
```