**DSA LAB REPORT**

**NAME: Nikhil Badyal**

**ROLL NO.: 001810501069**

**ASSIGNMENT1**

1. **PROBLEM STATEMENT:**

Write a program to compute the factorial of an integer value both recursively and iteratively. Check for overflow in the output and change data types to accommodate higher values.

**APPROACH:**

For iterative case run a loop from 1 to n and multiply. For recursive one calls the function with base case as to return 1 upon n being 1 or 0.

**CODE:**

#include <stdio.h>

unsigned long long int factByRecursion(int );

unsigned long long int factByLoop(int );

int main()

{

int n;

printf("Enter a positive integer: ");

scanf("%d", &n);

printf("Factorial of %d by recursion = %llu\n", n, factByRecursion(n));

printf("Factorial of %d by loop = %llu\n", n, factByLoop(n));

return 0;

}

unsigned long long int factByRecursion(int n)

{

if (n >= 1)

return n\*factByRecursion(n-1);

else

return 1;

}

unsigned long long int factByLoop(int n)

{

if(n == 0 || n == 1) return 1;

int i;

unsigned long long int factorial =1 ;

for(i=1; i<=n; ++i)

{

factorial \*= i;

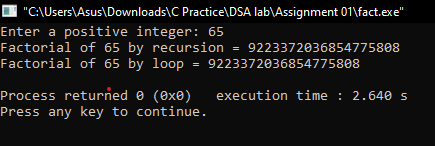
}

return factorial;

}

**LIMITATIONS**: Our code fails to provide factorials after n = 65.

**OUTPUT**:



2)

**PROBLEM STATEMENT:**

Write a program to generate nth Fibonacci number both iteratively and recursively. Check for overflow in the output and change data types to accommodate higher values. Plot the Fibonacci number vs n plot.

**APPROACH:** Fibonacci numbers follow a recurrence relation of f(n) = f(n-1) + f(n-2), n>2, f (0) = 0,f(1) =1

Using this relation nth fibonacci number can be generated easily.

**CODE :**

#include<stdio.h>

unsigned long long int fibLoop(int );

unsigned long long int fibRec(int );

unsigned long long int fibLoop(int n)

{

int i;

unsigned long long int arr[100];

arr[0]=0,arr[1]=1;

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

{

arr[i+2]=arr[i]+arr[i+1];

}

return arr[n-1];

}

unsigned long long int fibRec(int n)

{

if (n <= 1)

return n;

return fibRec(n-1) + fibRec(n-2);

}

int main ()

{

int n;

printf("Enter the term you want : \n");

scanf("%d",&n);

printf("Terms calculated using recursion %llu\n", fibRec(n-1));

printf("Terms calculated using Loop %llu\n", fibLoop(n));

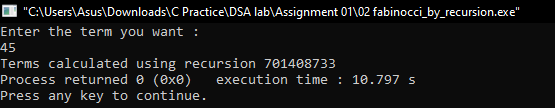
return 0;

}

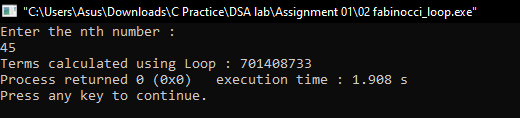
**LIMITATIONS :** As we can see recursion result are very slower when compared to loop variant due to expontial growth during recursion.

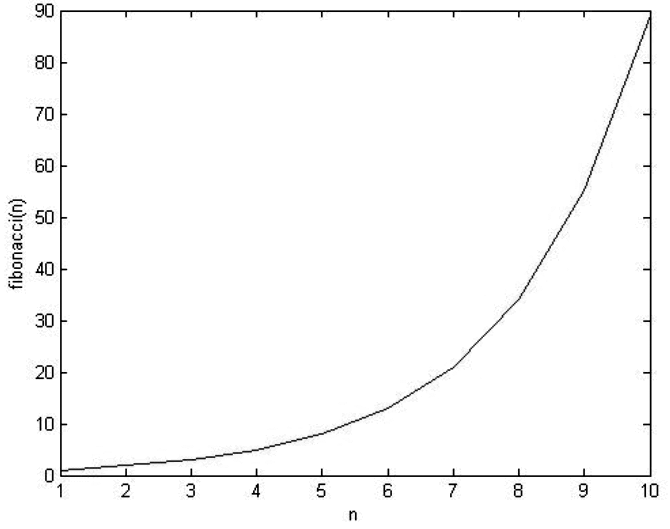
**OUTPUT:**

**Using recursion-**

****

Using for-loop-





3)

**PROBLEM STATEMENT** :

Write a program for linear and binary search.

**APPROACH :**

In linear search we check the array elements sequentially to find our required one. Worst case o(n). Binary search is only possible if the array is sorted. Here in every iteration we divide the array in two equal parts and search for our required element in the required half.

**CODE :**

#include<stdio.h>

#include<stdlib.h>

void swap(int \* , int \*);

int part(int \*,int ,int );

void quick\_sort(int \*,int ,int );

void binarySearch(int \*, int , int );

void linearSearch(int \*, int , int );

void swap(int\* a,int \*b)

{

int t;

t = \*b;

\*b = \*a;

\*a = t;

}

int part(int \*A,int start,int end)

{

int i;

int pivot = A[end]; //7

int pindex = start; //0

for(i=start;i<end;i++)

{

if(A[i]<=pivot)

{

swap(&A[i],&A[pindex]);

pindex++;

}

}

swap(&A[pindex],&A[end]);

return pindex;

}

void quick\_sort(int \*A,int start,int end)

{

if(start<=end)

{

int pindex = part(A,start,end);

quick\_sort(A,start,pindex-1);

quick\_sort(A,pindex+1,end);

}

}

void binarySearch(int \*arr, int n, int s)

{

int lower\_bound,higher\_bound,mid;

lower\_bound=0;

higher\_bound=n-1;

while(lower\_bound<=higher\_bound)

{

mid = (lower\_bound+higher\_bound)/2;

if(s<arr[mid])

higher\_bound=mid-1;

else if(arr[mid]<s)

{

lower\_bound=mid+1;

}

else if(lower\_bound==higher\_bound)

{

printf("Search successful.In sorted array element found at %d position.\n",(mid+1));

break;

}

}

if(lower\_bound>higher\_bound)

{

printf("Element %d is not present in array.\n",s);

}

return ;

}

void linearSearch(int \* arr ,int n, int x )

{

int i;

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

{

if(arr[i]==x)

{

printf("Search successful.In sorted array element found at %d position.\n",(i+1));

break;

}

}

if(i==n)

{

printf("%d not present in array.\n",x);

}

}

int main()

{

int i,n,arr[10],s;

printf("Enter the number of elements you want in array : \n");

scanf("%d",&n);

printf("Enter the elements of array : \n");

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

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

quick\_sort(arr,0,n-1);

printf("Enter the digit you want to search : \n");

scanf("%d",&s);

printf("\nResult by Binary search.\n");

binarySearch(arr,10,s);

printf("\nResult by Linear search.\n");

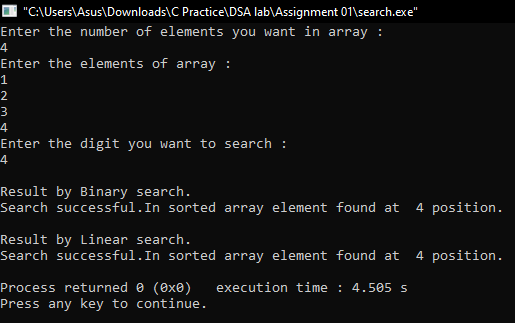
linearSearch(arr,10,s);

}

**LIMITATIONS :**

IN binary search if somenone gives an unsorted array as input the result will not be accurate.

**OUTPUT :**

****

4)

**PROBLEM STATEMENT :**

Write a program to generate 100000 random integers between 1 and 100000 witthout any repetition and store them in a file for further use.

**APPROACH :**

We will maintain a boolean array for 100000 with value initialized as zero.Now we will run a loop from 1 to 100000 and in each iteration we will generate a random number and check for it’s uniqueness in the bool array. If condition is satisfied we will mark that number as true in the bool array else we will make the loop run again for that iteration.

**CODE :**

#include<stdio.h>

#include<stdbool.h>

#include<time.h>

int main(void)

{

int i;

bool arr[100000]={0};

time\_t t;

srand((unsigned)time(&t));

FILE \*fr;

fr = fopen("numbers.txt","w");

for(i=0;i<100000;i++)

{

//printf("%d\n",i);

int r = rand()%100000;

if(!arr[r])

{ fprintf(fr,"%d%c",r,'\n');

arr[r]=1;

}

else

i--;

}

fclose(fr);

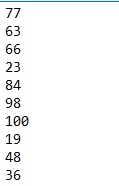
return 0;

}

**LIMITATIONS :**

To check for uniqueness the loop may have to undergo many iterations.

**OUTPUT :**



5)

**PROBLEM STATEMENT** :

Write a program to generate unique 100000 random strings of capital letters and of length 10. Store them in a file with 1 string per line.

**APPROACH :**

We are storing the random strings in a 2d character array by taking the data from alphanum string. For every generated string scan the str 2d array till we reach null if we get a match reiterate else continue.

**CODE :**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main()

{

char str[100000][10],alphanum[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

bool ar[10]={0};

int i,j,k=0;

FILE \*fp;

time\_t t;

srand((unsigned) time(&t));

fp=fopen("FILE2","w");

for(i=0;i<100000;i++)

{

for(j=0;j<10;j++)

{

str[i][j]=alphanum[rand() % (sizeof(alphanum) - 1)];

}

j=0;k=0;

while(str[j][] != ‘\0’)

{

if(!strcmp(str[j][],str[i][])

{ k=1;break;}

j++;

}

if(k==1)

i--;

}

for(i=0;i<100000;i++)

{

for(j=0;j<10;j++)

{

fprintf(fp,"%c",str[i][j]);

}

fprintf(fp,"\n");

}

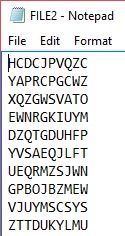
fclose(fp);

}

**LIMITATIONS :**

Sometimes the code have to reiterate for a particular value of I for uniqueness which for large input is very time consuming

**OUTPUT :**



6)

**PROBLEM STATEMENT** :

Store the names of your classmates according to roll number one text per line.Write a program to find the student with smallest and largest name. Sort the names in file according to ascending order.

**APPROACH:**

**CODE:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

int main()

{

int x,i,j,a[100],loc=1,loc1=1,max,min;

char name[100][100],temp[100];

FILE \*fp,\*fp1;

printf("Enter the number of classmates : \n");

scanf("%d",&x);

fp = fopen("random\_numbers.txt","w");

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

{

printf("Enter name : \t");

scanf("%s",&name[i]);

fprintf(fp,"%s\n",name[i]);

a[i] = strlen(name[i]);

printf("%d\n",a[i]);

}

fclose(fp);

max = a[0];

for(i=1;i<x;i++)

{

if (a[i] > max)

{

max = a[i];

loc = i+1;

}

}

printf("Maximum number of characters are present in %s located at %d and no. of characters are : %d.\n",name[loc-1],loc,max);

min = a[0];

for(i=1;i<x;i++)

{

if (a[i] < min)

{

min = a[i];

loc1 = i+1;

}

}

printf("Minimum number of characters are present in %s located at %d and no. of characters are : %d.\n",name[loc1-1],loc1,min);

for(i=0;i<x-1;i++)

{

for(j=i+1;j<x;j++)

{

if(strcmp(name[i],name[j])>0)

{

strcpy(temp,name[i]);

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

strcpy(name[j],temp);

}

}

}

fp1 = fopen("sorted\_names.txt","w");

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

{

fprintf(fp,"%s\n",name[i]);

}

}

7)

**PROBLEM STATEMENT :**

Take a 4 digit prime number p. Generate a series of large integers L and for every element L[i] divide it by p and take note of the remainder R[i]. Do the following for 7 seven other prime numbers keeping L fixed.

**APPROACH :**  The idea is simple. Input a four digit prime number by checking it’s validity. Then generate a random list of large integers. Divide the list elements by P.

**CODE :**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int i,j,upper=1000000000,lower=10000000,n,rem[40],arr[10];

int p[]= {1031,1013,1223,1249};

FILE \*fp;

fp = fopen("prime\_numbers.txt","w");

for(i=0;i<4;i++)

{

fprintf(fp,"%d\n",p[i]);

}

for(i=0;i<10;i++)

{

n=(rand()%(upper-lower+1)+lower);

arr[i]=n;

}

printf("Random number are :\n");

for(i=0;i<10;i++)

{

printf("%d - %d \n",i+1,arr[i]);

}

for(j=0;j<4;j++)

{

printf("\n\nprime number =%d \n",p[j]);

printf(" L(i)'s R(i)'s \n");

for(i=0;i<10;i++)

{

rem[i]=arr[i]%p[j];

printf(" %d = %d \n",arr[i],rem[i]);

}

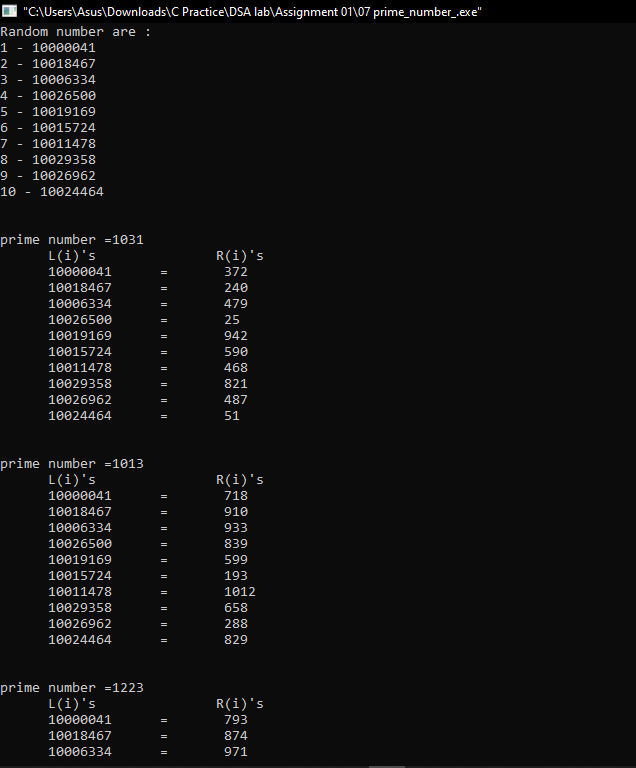
}

}

**LIMITATIONS:**

The code takes large time for considerably large list with large values.

**OUTPUT:**



8)

**PROBLEM STATEMENT :**

Convert your name and surname into large integers by juxtaposing integer ascii code for alphabets. Print the corresponding converted integer. Cut the large integer into two halves and add the two halves. Compute the remainder by dividing it by the prime numbers in problem 5.

APPROACH :

CODE:

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

void ascii(char[]);

int main()

{

int a;

char ASCII[15];

printf("Enter the character : \n");

gets(ASCII);

printf("\nThe ASCII value of entered name(%s): \n",ASCII);

ascii(ASCII);

return 0;

}

void ascii(char ASCII[])

{

int i,atoi1,prim1,a,x=0,mid,length,k,str11,str21,str\_sum,p[4],arr[100],n,div[10],lastans[100];

char str[100],str1[50],str2[50],prim[10],\*buf[100],prime[100][100];

FILE \*fp,\*fp1,\*fp2;

fp = fopen("ascii.txt","wt");

for(i=0;ASCII[i]!='\0';i++)

{

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

fprintf(fp,"%d",ASCII[i]);

}

fclose(fp);

printf("\n\n");

fp1 = fopen("ascii.txt","r");

if(fp==NULL)

{

printf("NOt found");

}

fgets(str,100,fp);

length = strlen(str);

mid=length/2;

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

str1[i]= str[i];

}

for(i = mid, k = 0; i <= length; i++, k++)

{

str2[k]= str[i];

}

str11 = atoi(str1);

str21 = atoi(str2);

printf("\nDividing the number in two half: \n");

printf("First half : %d\n",str11);

printf("Second half : %d\n",str21);

printf("\nSum after adding both the halves is : \n");

str\_sum = str11+str21;

printf("%d\n",str\_sum);

fp2 = fopen("prime\_numbers.txt","r");

i=0;

while(!feof(fp2))

{

fscanf(fp2,"%s",prime[i]);

i++;

}

for(i=0;i<4;i++)

{

div[i]=atoi(prime[i]);

}

printf("\n");

for(i=0;i<4;i++)

{

lastans[i]=str\_sum%div[i];

}

for(i=0;i<4;i++)

{

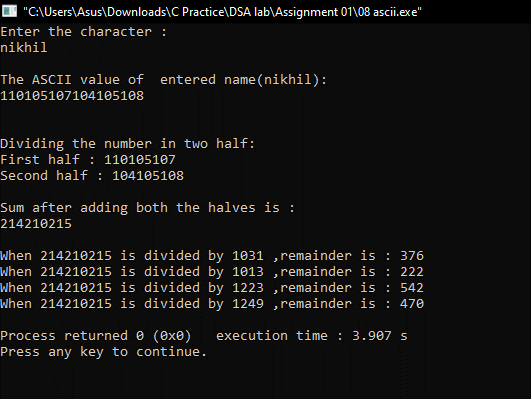
printf("When %d is divided by %d ,remainder is : ",str\_sum,div[i]);

printf("%d\n",lastans[i]);

}

}

OUTPUT:



**ASSIGNMENT 2**

1)

**PROBLEM STATEMENT :**

Define an ADT for polynomials. Write C data representations and functions in a header file and implement.

**APPROACH :**

First we declare a stucture for a single term, whose attributes are coefficients and exponent. Then to define a polynomial, we declare another structure named 'poly' whose attributes are number of terms and the terms. Then we initialise the coefficients and the exponents as 0, as there are no terms initially. To append new term to the polynomial we store the values of coefficient and exponent as given by users ans increment the number of terms by 1. To display the terms we loop through every elements of struct poly. For addition of two polynomials we first check the exponent of both polynomials, if the match then e add the coefficients and obtain the answer. To multiply two polynomials, for each term of first polynomial we add the exponents of the variable and multiply the coefficient.

**CODE:**

**header.h**

struct poly

{

float coeff;

int exp;

};

struct poly a[50],b[50],c[50],d[50];

int deg1,deg2;

void subtract();

void multiply();

void create\_poly();

void add\_poly();

void subtract()

{

int i,m=0;

if(deg1>deg2)

{

for(i=0;i<=deg2;i++)

{

c[m].coeff = a[i].coeff - b[i].coeff;

c[m].exp = a[i].exp;

m++;

}

for(i=deg2+1;i<=deg1;i++)

{

c[m].coeff = a[i].coeff;

c[m].exp = a[i].exp;

m++;

}

}

else

{

for(i=0;i<=deg1;i++)

{

c[m].coeff = a[i].coeff - b[i].coeff;

c[m].exp = a[i].exp;

m++;

}

for(i=deg1+1;i<=deg2;i++)

{

c[m].coeff = b[i].coeff;

c[m].exp = b[i].exp;

m++;

}

}

//printing the sum of the two polynomials

printf("\nExpression after absolute subtraction = %.1f",c[0].coeff);

for(i=1;i<m;i++)

{

printf("+ %.1fx^%d",c[i].coeff,c[i].exp);

}

}

void multiply()

{

int i,m=0;

if(deg1>deg2)

{

for(i=0;i<=deg2;i++)

{

c[m].coeff = a[i].coeff \* b[i].coeff;

c[m].exp = a[i].exp;

m++;

}

for(i=deg2+1;i<=deg1;i++)

{

c[m].coeff = a[i].coeff;

c[m].exp = a[i].exp;

m++;

}

}

else

{

for(i=0;i<=deg1;i++)

{

c[m].coeff = a[i].coeff \* b[i].coeff;

c[m].exp = a[i].exp;

m++;

}

for(i=deg1+1;i<=deg2;i++)

{

c[m].coeff = b[i].coeff;

c[m].exp = b[i].exp;

m++;

}

}

//printing the product of the two polynomials

printf("\nExpression after multiplication = %.1f",c[0].coeff);

for(i=1;i<m;i++)

{

printf("+ %.1fx^%d",c[i].coeff,c[i].exp);

}

}

void create\_poly()

{

int i;

int k=0,l=0;

printf("Enter the highest degree of polynomial1: ");

scanf("%d",&deg1); //taking polynomial terms from the user

for(i=0;i<=deg1;i++)

{

//entering values in coefficient of the polynomial terms

printf("Enter the coeff of x^%d : ",i);

scanf("%f",&a[i].coeff);

//entering values in exponent of the polynomial terms

a[k++].exp = i;

}

//taking second polynomial from the user

printf("Enter the highest degree of polynomial2: ");

scanf("%d",&deg2);

for(i=0;i<=deg2;i++)

{

printf("Enter the coeff of x^%d : ",i);

scanf("%f",&b[i].coeff);

b[l++].exp = i;

}

//printing first polynomial

printf("Expression 1 = %.1f ",a[0].coeff);

for(i=1;i<=deg1;i++)

{

printf("+ %.1fx^%d",a[i].coeff,a[i].exp);

}

//printing second polynomial

printf("\nExpression 2 = %.1f",b[0].coeff);

for(i=1;i<=deg2;i++)

{

printf("+ %.1fx^%d",b[i].coeff,b[i].exp);

}

}

void add\_poly()

{

int i,m=0;

if(deg1>deg2)

{

for(i=0;i<=deg2;i++)

{

c[m].coeff = a[i].coeff + b[i].coeff;

c[m].exp = a[i].exp;

m++;

}

for(i=deg2+1;i<=deg1;i++)

{

c[m].coeff = a[i].coeff;

c[m].exp = a[i].exp;

m++;

}

}

else

{

for(i=0;i<=deg1;i++)

{

c[m].coeff = a[i].coeff + b[i].coeff;

c[m].exp = a[i].exp;

m++;

}

for(i=deg1+1;i<=deg2;i++)

{

c[m].coeff = b[i].coeff;

c[m].exp = b[i].exp;

m++;

}

}

//printing the sum of the two polynomials

printf("\nExpression after addition = %.1f",c[0].coeff);

for(i=1;i<m;i++)

{

printf("+ %.1fx^%d",c[i].coeff,c[i].exp);

}

}

**Evaluate.h**

float poly(float \*, int, float);

void evaluate();

void evaluate()

{

float x, a[10], y1;

int deg, i;

printf("Enter the degree of polynomial equation: ");

scanf("%d", &deg);

printf("Enter the value of x for which the equation is to be evaluated: ");

scanf("%f", &x);

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

{

printf("Enter the coefficient of x to the power %d: ",i);

scanf("%f",&a[i]);

}

y1 = poly(a, deg, x);

printf("The value of polynomial equation for the value of x = %.2f is: %.2f",x,y1);

}

float poly(float z[], int deg, float x)

{

float p;

int i;

p = z[deg];

for(i=deg;i>=1;i--)

{

p = (z[i-1] + x\*p);

}

return p;

}

**Derivate.h**

float deriv(float \*, int, float);

void derivate();

void derivate()

{

float x, a[10], dy1;

int deg, i;

printf("Enter the degree of polynomial equation: ");

scanf("%d", &deg);

printf("Enter the value of x for which the equation is to be evaluated: ");

scanf("%f", &x);

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

{

printf("Enter the coefficient of x to the power %d: ",i);

scanf("%f",&a[i]);

}

dy1 = deriv(a, deg, x);

printf("\nThe value of the derivative of the polynomial equation at x = %.2f is: %.2f",x,dy1);

}

/\* function for finding the derivative at some value of x \*/

float deriv(float z[], int deg, float x)

{

float o[10], pd = 0, ps;

int i;

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

{

ps = pow(x, deg-(i+1));

o[i] = (deg-i)\*z[deg-i]\*ps;

pd = pd + o[i];

}

return pd;

}

**main.c**

// Polynomial adt to create ,add , subtract , multiply , evaluate , derivate polynomial.

#include<stdio.h>

#include<math.h>

#include<stdlib.h>

#include<stdbool.h>

#include"header.h"

#include"evaluate.h"

#include"derivate.h"

int main()

{

int task,x=1;

while(x==1)

{

printf("\nEnter 1 to create polynomial.\nEnter 2 to add polynomial.\nEnter 3 to subtract polynomial.\nEnter 4 to multiply polynomial.\nEnter 5 to evaluate."

"\nEnter 6 to derivate polynomial \nEnter 7 to exit \n");

while(true)

{

scanf("%d",&task);

switch(task)

{

case 1: create\_poly();

printf("\n");

break;

case 2: add\_poly();

printf("\n");

break;

case 3: subtract();

printf("\n");

break;

case 4: multiply();

printf("\n");

break;

case 5: evaluate();

printf("\n");

break;

case 6: derivate();

printf("\n");

break;

case 7 : exit(0);

default : printf("Enter choose valid number.\n");

}

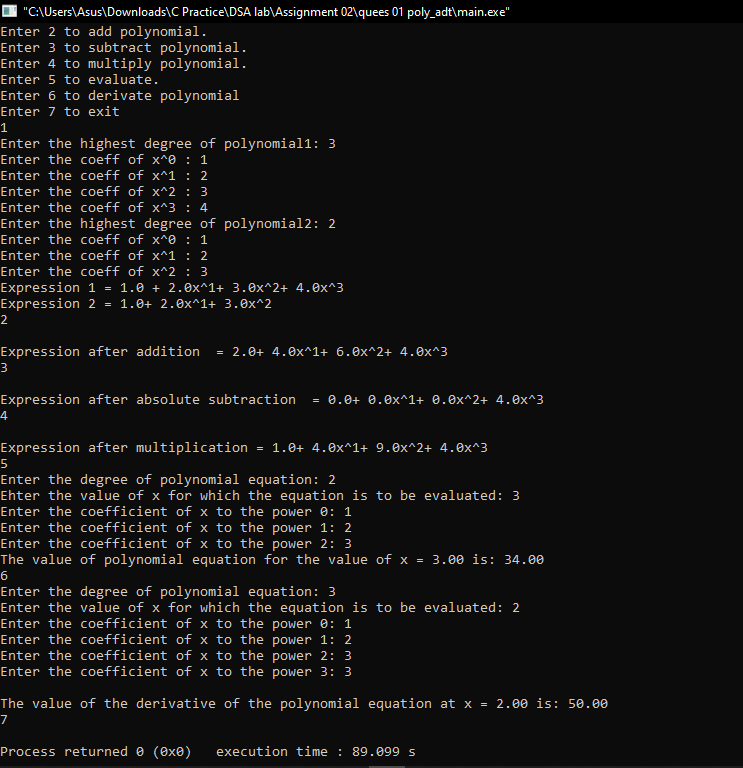
}

}

return 0;

}

**OUTPUT:**

****

**2)**

**PROBLEM STATEMENT :**

. Define ADT for sparse matrix. Write C data representation and functions for the operations on sparse matrix in a header file and implement them.

**APPROACH :**

Number of non-zero elements, transpose of sparse matrix, creating tuple,displaying tuple.

First we create a struct name sparse whose attributes are an array 'sp' and the number of rows of the newly formed sparse matrix which is equal to number of non-zero elements + 1 for keeping track of number of rows and columns of actual matrix. Then , we counted the number of non-zero elements and keep the count in a variable. After that, we created tuplefor non-zero elements, where the first and second element of each row denotes the row and column number of the elements in the actual matrix and in the last entry of a row we stored the element. We have a display function to show the actual matrix and the sparse matrix. Then we created a function to transpose this sparse matrix.First, we allocated memory required to store the elements in the target 3-tuple.Then we store the number of rows and columns.The transpose operation is carried out through a pair of for loops.the oure for loop runs till the non-zero elements of col number of columns are not scanned. In the inner for loop first we have obtained the position at which the column number of a non-zero element is stored . Then we have checked whether the column number of a non-zero element matches with the column number currently being considered.Another variable is used for the tuple, to store the position at which data from souce tuple, should get copied. Similarly another variable is used for the tuple, to extract data from it. Then we copied the column position of a non-zero element from source tuple.This column number gets stored at the row position in target tuple. On similar lines the row position of a non-zero element of source tuple is copied at the column position of the target tuple.

**CODE :**

**create.h**

#include<stdio.h>

#include<stdlib.h>

#define MAX 20

void readsparse(int b[MAX][3]);

void printsparse(int b[MAX][3]);

void create\_sparse()

{

int b1[MAX][3],b2[MAX][3];

readsparse(b1);

printsparse(b1);

}

void readsparse(int b[MAX][3])

{

int i,t,m,n;

printf("Enter no. of rows and columns:\n");

scanf("%d%d",&m,&n);

printf("No. of non-zero triples:");

scanf("%d",&t);

b[0][0]=m;

b[0][1]=n;

b[0][2]=t;

for(i=1;i<=t;i++)

{

printf("\nEnter the triples(row,column,value):\n");

scanf("%d%d%d",&b[i][0],&b[i][1],&b[i][2]);

}

}

void printsparse(int b[MAX][3])

{

int i,t;

t=b[0][2];

printf("\nrow\tcolumn\tvalue");

for(i=1;i<=t;i++)

{

printf("\n%d\t%d\t%d",b[i][0],b[i][1],b[i][2]);

}

}

**add.h**

#include<stdio.h>

#include<stdlib.h>

#define MAX 20

void printsparse1(int b[MAX][3]);

void readsparse1(int b[MAX][3]);

void addsparse1(int b1[MAX][3],int

b2[MAX][3],int b3[MAX][3]);

void add\_sparse()

{

int b1[MAX][3],b2[MAX][3],b3[MAX][3];

readsparse1(b1);

readsparse1(b2);

addsparse1(b1,b2,b3);

printsparse1(b3);

}

void readsparse1(int b[MAX][3])

{

int i,t,m,n;

printf("nEnter no. of rows and columns:");

scanf("%d%d",&m,&n);

printf("No. of non-zero triples:");

scanf("%d",&t);

b[0][0]=m;

b[0][1]=n;

b[0][2]=t;

for(i=1;i<=t;i++)

{

printf("Enter the triples(row,column,value):");

scanf("%d%d%d",&b[i][0],&b[i][1],&b[i][2]);

}

}

void addsparse1(int b1[MAX][3],int

b2[MAX][3],int b3[MAX][3])

{

int t1,t2,i,j,k;

if(b1[0][0]!=b2[0][0]||b1[0][1]!=b2[0][1])

{

printf("\nYou have entered invalid matrix!!Size must be equal");

exit(0);

}

t1=b1[0][2];

t2=b2[0][2];

printf("hello");

printf("%d%d",t1,t2);

i=j=k=0;

b3[0][0]=b1[0][0];

b3[0][1]=b1[0][1];

while(i<=t1&&j<=t2)

{

if(b1[i][0]<b2[j][0])

//row numbers are not equal

{

b3[k][0]=b1[i][0];

b3[k][1]=b1[i][1];

b3[k][2]=b1[i][2];

k++;

i++;

}

else if(b2[j][0]<b1[i][0])

//row numbers are not equal

{

b3[k][0]=b2[j][0];

b3[k][1]=b2[j][1];

b3[k][2]=b2[j][2];

k++;

j++;

}

else if(b1[i][1]<b2[j][1])

//row numbers are equal, compare column

{

b3[k][0]=b1[i][0];

b3[k][1]=b1[i][1];

b3[k][2]=b1[i][2];

k++;

i++;

}

else if(b2[j][1]<b1[i][1])

//row numbers are equal, compare column

{

b3[k][0]=b2[j][0];

b3[k][1]=b2[j][1];

b3[k][2]=b2[j][2];

k++;

j++;

}

else

{

b3[k][0]=b1[i][0]; //row and column numbers are equal

b3[k][1]=b1[i][1];

b3[k][2]=b1[i][2]+b2[j][2];

k++;

i++;

j++;

}

}

while(i<=t1) //copy remaining terms from b1

{

b3[k][0]=b1[i][0];

b3[k][1]=b1[i][1];

b3[k][2]=b1[i][2];

i++;

k++;

}

while(j<=t2) //copy remaining terms from b2

{

b3[k][0]=b2[j][0];

b3[k][1]=b1[j][1];

b3[k][2]=b1[j][2];

j++;

k++;

}

b3[0][2]=k-1; //set number of terms in b3

}

void printsparse1(int b[MAX][3])

{

int i,t;

t=b[0][2];

printf("\nrow\tcolumn\tvalue");

for(i=1;i<=t;i++)

{

printf("\n%d\t%d\t%d",b[i][0],b[i][1],b[i][2]);

}

}

**subtract.h**

#include<stdio.h>

#include<stdlib.h>

#define MAX 20

void printsparse4(int b[MAX][3]);

void readsparse4(int b[MAX][3]);

void subtractsparse4(int b1[MAX][3],int

b2[MAX][3],int b3[MAX][3]);

void subtract\_sparse()

{

int b1[MAX][3],b2[MAX][3],b3[MAX][3];

readsparse4(b1);

readsparse4(b2);

subtractsparse4(b1,b2,b3);

printsparse4(b3);

}

void readsparse4(int b[MAX][3])

{

int i,t,m,n;

printf("nEnter no. of rows and columns:");

scanf("%d%d",&m,&n);

printf("No. of non-zero triples:");

scanf("%d",&t);

b[0][0]=m;

b[0][1]=n;

b[0][2]=t;

for(i=1;i<=t;i++)

{

printf("Enter the triples(row,column,value):");

scanf("%d%d%d",&b[i][0],&b[i][1],&b[i][2]);

}

}

void subtractsparse4(int b1[MAX][3],int

b2[MAX][3],int b3[MAX][3])

{

int t1,t2,i,j,k;

if(b1[0][0]!=b2[0][0]||b1[0][1]!=b2[0][1])

{

printf("\nYou have entered invalid matrix!!Size must be equal");

exit(0);

}

t1=b1[0][2];

t2=b2[0][2];

i=j=k=0;

b3[0][0]=b1[0][0];

b3[0][1]=b1[0][1];

while(i<=t1&&j<=t2)

{

if(b1[i][0]<b2[j][0])

//row numbers are not equal

{

b3[k][0]=b1[i][0];

b3[k][1]=b1[i][1];

b3[k][2]=b1[i][2];

k++;

i++;

}

else if(b2[j][0]<b1[i][0])

//row numbers are not equal

{

b3[k][0]=b2[j][0];

b3[k][1]=b2[j][1];

b3[k][2]=b2[j][2];

k++;

j++;

}

else if(b1[i][1]<b2[j][1])

//row numbers are equal, compare column

{

b3[k][0]=b1[i][0];

b3[k][1]=b1[i][1];

b3[k][2]=b1[i][2];

k++;

i++;

}

else if(b2[j][1]<b1[i][1])

//row numbers are equal, compare column

{

b3[k][0]=b2[j][0];

b3[k][1]=b2[j][1];

b3[k][2]=b2[j][2];

k++;

j++;

}

else

{

b3[k][0]=b1[i][0]; //row and column numbers are equal

b3[k][1]=b1[i][1];

b3[k][2]=b1[i][2]-b2[j][2];

k++;

i++;

j++;

}

}

while(i<=t1) //copy remaining terms from b1

{

b3[k][0]=b1[i][0];

b3[k][1]=b1[i][1];

b3[k][2]=b1[i][2];

i++;

k++;

}

while(j<=t2) //copy remaining terms from b2

{

b3[k][0]=b2[j][0];

b3[k][1]=b1[j][1];

b3[k][2]=b1[j][2];

j++;

k++;

}

b3[0][2]=k-1; //set number of terms in b3

}

void printsparse4(int b[MAX][3])

{

int i,t;

t=b[0][2];

printf("\nrow\tcolumn\tvalue");

for(i=1;i<=t;i++)

{

printf("\n%d\t%d\t%d",b[i][0],b[i][1],b[i][2]);

}

}

**transpose.h**

#include<stdio.h>

#define MAX 20

void printsparse2(int[][3]);

void readsparse2(int[][3]);

void transpose2(int[][3],int[][3]);

void transpose()

{

int m,n;

printf("Enter the size of matrix (rows,columns):");

scanf("%d%d",&m,&n);

int b1[MAX][3],b2[MAX][3];

b1[0][0]=m;

b1[0][1]=n;

readsparse2(b1);

transpose2(b1,b2);

printsparse2(b2);

}

void readsparse2(int b[MAX][3])

{

int i,t;

printf("\nEnter no. of non-zero elements:");

scanf("%d",&t);

b[0][2]=t;

for(i=1;i<=t;i++)

{

printf("\nEnter the next triple(row,column,value):");

scanf("%d%d%d",&b[i][0],&b[i][1],&b[i][2]);

}

}

void printsparse2(int b[MAX][3])

{

int i,n;

n=b[0][2]; //no of 3-triples

printf("\nAfter Transpose:\n");

printf("\nrow\t\tcolumn\t\tvalue\n");

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

printf("%d\t\t%d\t\t%d\n",b[i][0],b[i][1],b[i][2]);

}

void transpose2(int b1[][3],int b2[][3])

{

int i,j,k,n;

b2[0][0]=b1[0][1];

b2[0][1]=b1[0][0];

b2[0][2]=b1[0][2];

k=1;

n=b1[0][2];

for(i=0;i<b1[0][1];i++)

// k=1;

for(j=1;j<=n;j++)

//if a column number of current triple==i then insert the current triple in b2

if(b1[j][1]==i)z

{

b2[k][0]=i;

b2[k][1]=b1[j][0];

b2[k][2]=b1[j][2];

k++;

}

}

**main.c**

// Polynomial adt to create,display ,add , subtract , transpose sparse matrix

#include<stdio.h>

#include"pre\_check.h"

#include"create.h"

#include"add.h"

#include"subtract.h"

#include"transpose.h"

int main()

{

int task,x=1;

while(x==1)

{

printf("Enter 1 to create sparse.\nEnter 2 to add sparse.\nEnter 3 to subtract.\nEnter 4 to transpose.\n ");

task = pre\_check();

switch(task)

{

case 1: create\_sparse();

break;

case 2: add\_sparse();

break;

case 3: subtract\_sparse();

break;

case 4: transpose();

break;

}

printf("\nEnter 1 to continue.\n");

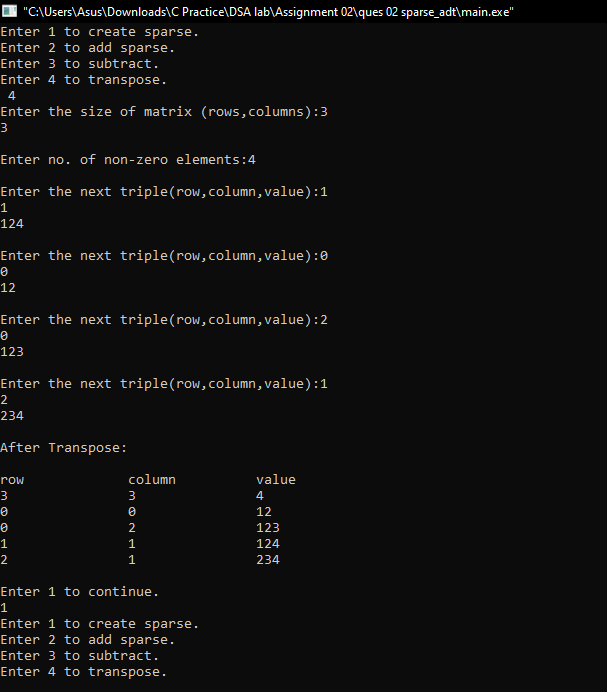
scanf("%d",&x);

}

return 0;

}

**OUTPUT:**

****

3)

**PROBLEM STATEMENT :**

Define an ADT for list. Write C data representation and functions for the operations on the header file and implement it.

**APPROACH :**

Add a node at the front:

The new node is always added before the head of the given Linked List. And newly added node becomes the new head of the Linked List. For example if the given Linked List is 10->15->20->25 and we add an item 5 at the front, then the Linked List becomes 5->10->15->20->25. Let us call the function that adds at the front of the list is push(). The push() must receive a pointer to the head pointer, because push must change the head pointer to point to the new node

Add a node after a given node:

We are given pointer to a node, and the new node is inserted after the given node.We traverse through the list and find the node after which we have to insert the new node, and as per requirement we set the link .

Add a node at last:

The new node is always added after the last node of the given Linked List. For example if the given Linked List is 5->10->15->20->25 and we add an item 30 at the end, then the Linked List becomes 5->10->15->20->25->30.

Since a Linked List is typically represented by the head of it, we have to traverse the list till end and then change the next of last node to new node.

Delete a node:

We loop through each element and find the node which we have to delete, then we copy the link part of it and paste it into it's previous node's link part.

**CODE :**

**LIST.h**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define MAX 4 // Macro defines maximum no. of elements in the list. It is a user defined data type

struct

{

int list[MAX];

int element; //new element to be inserted

int pos; //position of the element to be inserted or deleted

int length; //total no of elements

}l;

enum boolean

{

true, false

};

typedef enum boolean boolean; //function prototypes

int menu(void); //This function displays the list of operations

void create(void); //This function creates initial set of elements

void insert(int, int); //This function inserts the element at specified position

void delet(int); //This function deletes the element at given position

void find(int); //This function finds the position of the given element, if exists

void display(void); //This function displays the elements in the list

boolean islistfull(void); //This function checks whether the list is full or not boolean

boolean islistempty(void); //This function checks whether the list is empty or not

void main()

{

int ch;

int element;

int pos;

l.length = 0;

while(1)

{

ch = menu();

switch (ch)

{

case 1: l.length = 0;

create();

break;

case 2:

if (islistfull() != true)

{

printf("Enter New element: ");

scanf("%d", &element);

printf("Enter the Position : ");

scanf("%d", &pos);

insert(element, pos);

}

else

{

printf("List is Full. Cannot insert the element");

printf("\n Press any key to continue...");

getch();

}

break;

case 3:

if (islistempty() != true)

{

printf("Enter the position of element to be deleted : ");

scanf("%d", &pos);

delet(pos);

}

else

{

printf("List is Empty.");

printf("\n Press any key to continue...");

getch();

}

break;

case 4:

printf("No of elements in the list is %d", l.length);

printf("\n Press any key to continue...");

getch();

break;

case 5:

printf("Enter the element to be searched : ");

scanf("%d", &element);

find(element);

break;

case 6:

display();

break;

case 7:

printf("Exit");

exit(0);

break;

default: printf("Invalid Choice");

printf("\n Press any key to continue...");

getch();

}

}

} //function to display the list of elements

int menu()

{

int ch;

//clrscr();

printf("1. Create\n2. Insert\n3. Delete\n4. Count\n5. Find\n6. Display\n7.Exit\n\n Enter your choice : ");

scanf("%d", &ch);

printf("\n\n");

return ch;

}

void create(void)

{

int element;

int flag=1;

while(flag==1)

{

printf("Enter element : ");

scanf("%d", &element);

l.list[l.length] = element;

l.length++;

printf("To insert another element press '1' : ");

scanf("%d", &flag);

}

}

void display(void)

{

int i;

for (i=0; i<l.length; i++)

printf("Element %d : %d \n", i+1, l.list[i]);

printf("Press any key to continue...");

getch();

}

void insert(int element, int pos)

{

int i;

if (pos == 0)

{

printf("\nCannot insert an element at 0th position");

getch();

return;

}

if (pos-1 > l.length)

{

printf("\nOnly %d elements exit. Cannot insert at %d position", l.length, pos);

printf("\n Press any key to continue...");

getch();

}

else

{

for (i=l.length; i>=pos-1; i--)

{

l.list[i+1] = l.list[i];

}

l.list[pos-1] = element;

l.length++;

}

}

void delet(int pos)

{

int i;

if(pos == 0)

{

printf("\nCannot delete at an element 0th position");

getch();

return;

}

if (pos > l.length)

{

printf("\n\n Only %d elements exit. Cannot delete", l.length, pos);

printf("\n Press any key to continue...");

getch();

return;

}

for (i=pos-1; i<l.length; i++)

{

l.list[i] = l.list[i+1];

}

l.length--;

}

void find(int element)

{

int i;

int flag = 1;

for (i=0; i<l.length; i++)

{

if(l.list[i] == element)

{

printf ("%d exists at %d position",element, i+1);

flag = 0;

printf("\n Press any key to continue...");

getch();

break;

}

}

}

boolean islistfull(void)

{

if (l.length == MAX)

return true;

else

return false;

}

boolean islistempty(void)

{

if (l.length == 0)

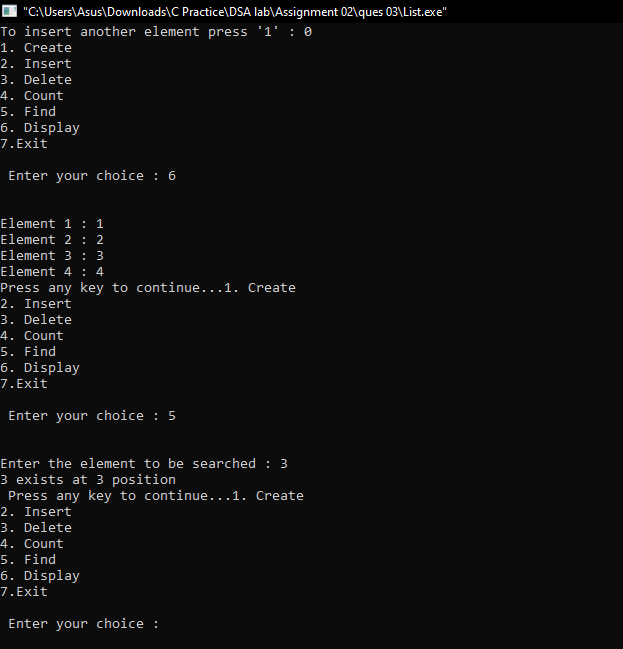
return true;

else

return false;

}

**OUTPUT :**

****

4)

**PROBLEM STATEMENT :**

Define ADT for set. Write C data representation for the opeartions on set in a header file and implement it.

**APPROACH :**

First we sorted the elements to store in a set. Then we checked fro duplicates and if found we get rid of it, and we did it by running a loop and comparing current and next element. Then we wrote a function to find the number of elements in the set. There is a function for searching a particular key. We performed linear search here, although we could have done it by binary search as we have sorted order of elements. There is a function fpr deleting element. For this we have to copy all the next elements and at last we will have the required set with total number of elements decreased by 1.

**CODE :**

**SET.h**

#include <stdio.h>

void sort(int \*a,int n){

int i,j,temp;

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

for(j=i+1; j<n; j++){

if(a[i]>a[j]){

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

}

}

void final(int\* a,int n){

int i,j=0;

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

if(a[i]!=a[i+1]){

a[j] = a[i];

j++;

}

}

if(a[n-2]!=a[n-1]) return;

a[j] = a[n-1];

}

int lenght(int\* a,int n){

int i,j=0;

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

if(a[i] != a[i+1]) j++;

}

return j;

}

void display(int \*a,int len){

int i;

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

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

}

printf("\n");

}

int find(int \*a,int n,int len){

int i;

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

if(a[i] == n) return i+1;

}

return 0;

}

void erase(int \*a,int pos,int \*len){

int i;

if(pos>\*len) return ;

for(i=pos; i<\*len-1; i++){

a[i] = a[i+1];

}

\*len--;

}

**MAIN.C**

#include <stdio.h>

#include <stdlib.h>

#include "set.h"

int main(){

int \*a,n,i,size,m,key,d;

printf("Enter the number of elements you want to enter\n");

scanf("%d",&n);

a = (int\*)malloc(sizeof(int)\*n);

printf("Enter the elements.\n");

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

scanf("%d",a+i);

}

sort(a,n);

final(a,n);

size = lenght(a,n);

printf("After converting the inputs into set, we get:\n");

display(a,size);

printf("Number of elements of the set = %d\n",size);

printf("Enter the value you want to search for:\n");

scanf("%d",&m);

key = find(a,m,size);

if(key) printf("Found!It is present at position %d\n",key);

else printf("No element found\n");

printf("Enter the position whose element you want to delete.\n");

scanf("%d",&d);

printf("After deleting ,the set is:\n");

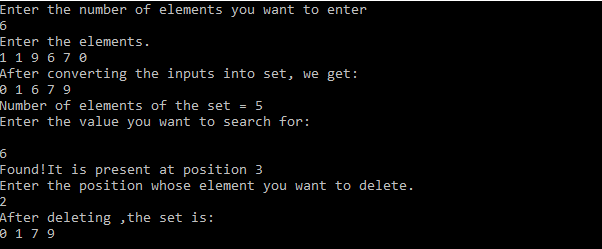
erase(a,d,&size);

display(a,size-1);

return 0;

}

**OUTPUT :**

****

5)

**PROBLEM STATEMENT :**

Define ADT for string. Write C data representation for the opeartions on set in a header file and implement it.

**APPROACH :**

We have a function for finding the number of characters in a string. We just loop through each element and increase the counter by 1 till we enconter '\0'. Then we defined another function to find out the number of occurences a particular character made, we achieved this by looping through each element till the last one and compared the elements with the given character, if they are equal we inrease the counter by 1. There is also another function to find out the first occurrence of a character. Again we loop through the elements and check whether the element matches with the given character or not and for its first appearence we return the position. To copy one string into another, we loop through the elements and copy the parent string character by character. We compared two given strings character by character, if the strings are same we return 0 and if they are not then we return the ASCII value difference between the first unmatched character. To concatenate strings we declare another larger empty character array, and copy the elements of given strings into it. And at last we have a function to dislay the elements. We loop through the elements till we find '\0' and printed the characters on the screen.

**CODE :**

**STRING.h**

#include<stdlib.h>

#include <stdbool.h>

#include<stdio.h>

#define FALSE 0

#define TRUE !(FALSE)

int size(const char arr[])

{

int i=0;

while(arr[i] != '\0')

{

i++;

}

return (i+1);

}

void concatinate(char lhs[] , char rhs[] )

{

int i=0,j=0;

char res[30];

while(lhs[i] != '\0' )

{

res[i] = lhs[i];

++i;

}

while(rhs[j] != '\0')

{

res[i] = rhs[j];

++j;

++i;

}

res[i] = '\0';

printf("%s",res);

}

bool string\_equal(const char lhs[], const char rhs[])

{

int s1,s2;

s1 = size(lhs);

s2 = size(rhs);

if(s1==s2)

{

int i=0; // Not equal

while(lhs[i]==rhs[i] || i<=s1)

{

i++;

}

if(i!=s1){ return 0;}

else { return 1;}

}

else{ return 0;}

}

int to\_integer(char lhs[])

{

int num;

if(sscanf(lhs,"%d",&num)==EOF){

fprintf(stderr,"Warning ! Incorrect value for device.");

return FALSE;

}

else { return num;}

}

char append\_char(char lhs[],char s)

{

int i=0;

while(lhs[i]!='\0') {i++;}

lhs[i] = s;

lhs[i+1] = '\0';

}

**Main.h**

#include<stdlib.h>

#include<stdio.h>

#include"String.h"

char lhs[15] = "Nikhil";

char rhs[15] = "Aikhil";

int main()

{

printf("Strings are ");

if(string\_equal(lhs,rhs))

printf("Equal\n");

else{

printf("UnEqual\n");

}

printf("Strings after concatenating is ");

concatinate(lhs,rhs);

printf("\n");

printf("String after conversion ");

printf("%d",to\_integer("234"));

printf("\n");

printf("String size is ");

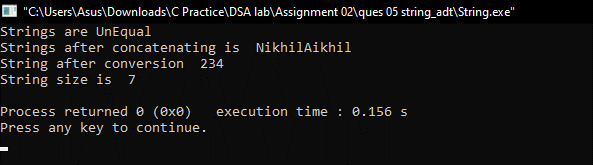
printf("%d",size(lhs));

printf("\n");

return 0;

}

**OUTPUT :**



6)

**PROBLEM STATEMENT :**

Given a large single dimension array of integers write function for sliding window filter with maximum , minimum , median and average to generate an output. The window size must be odd . Explain , how will u handle the boundary cases.

**APPROACH :**

After recieving the array and the length of sliding window from the user, we find the maximum element for the window size for the entire array. In the outer for loop we just loop through every element and in the inner-loop for we again loop throught the elements for the given window size and find the maximum.

For minimum, in the outer for loop we just loop through every element and in the inner-loop for we again loop throught the elements for the given window size and find the minimum.

For median, in outer loop we visit every element and in inner loop we iterate for the given window size and sort those many number of elements and find the median.

For minimum, in the outer for loop we just loop through every element and in the inner-loop for we again loop throught the elements for the given window size and add the elements and divide the sum by size of the given window for finding the average.

**CODE :**

**WINDOW.h**

#include<stdio.h>

#include<stdlib.h>

int arr[1000] = {0};

int op[101];

int temp[101];

void bubbleSort(int a[], int n)

{

int temp;

for(int i=0;i<n;i++)

{

for(int j=0;j<n-i-1;j++)

{

if(a[j]>a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

}

void maximum(int a[], int n, int k)

{

for(int i=0;i<n;i++)

{

int temp=a[i];

for(int j=1;j<k;j++)

{

if(a[i+j]>temp)

temp=a[i+j];

}

op[i]=temp;

}

}

void minimum(int a[], int n, int k)

{

for(int i=0;i<n;i++)

{

int temp=a[i];

for(int j=1;j<k;j++)

{

if(a[i+j]<temp)

temp=a[i+j];

}

op[i]=temp;

}

}

void median(int a[], int n, int k)

{

int c;

for(int i=0;i<n;i++)

{

c=0;

for(int j=i;j<i+k;j++)

temp[c++]=a[j];

bubbleSort(temp,c);

op[i]=temp[c/2];

}

}

void average(int a[], int n, int k)

{

int sum;

for(int i=0;i<n;i++)

{

sum=0;

for(int j=i;j<i+k;j++)

{

sum+=a[j];

}

op[i]=sum/k;

}

}

void display(int a[], int n)

{

for(int i=0;i<n;i++)

{

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

}

}

**MAIN.C**

#include"window.h"

int main()

{

int n,k,ch,c;

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_MENU\_LIST\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("Enter 1 for Maximum\nEnter 2 for Minimum\nEnter 3 for Median\nEnter 4 for Average\n\n");

printf("Enter your choice : ");

do

{

scanf("%d",&ch);

printf("Enter length of array : ");

scanf("%d",&n);

printf("Enter size of sliding window : ");

scanf("%d",&k);

printf("Enter array elements : \n");

for(int i=0;i<n;i++)

{

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

}

switch(ch)

{

case 1:

maximum(arr,n,k);

break;

case 2:

minimum(arr,n,k);

break;

case 3:

median(arr,n,k);

break;

case 4:

average(arr,n,k);

break;

default:

printf("Wrong Choice!\n");

}

display(op,(n-k+1));

printf("\n");

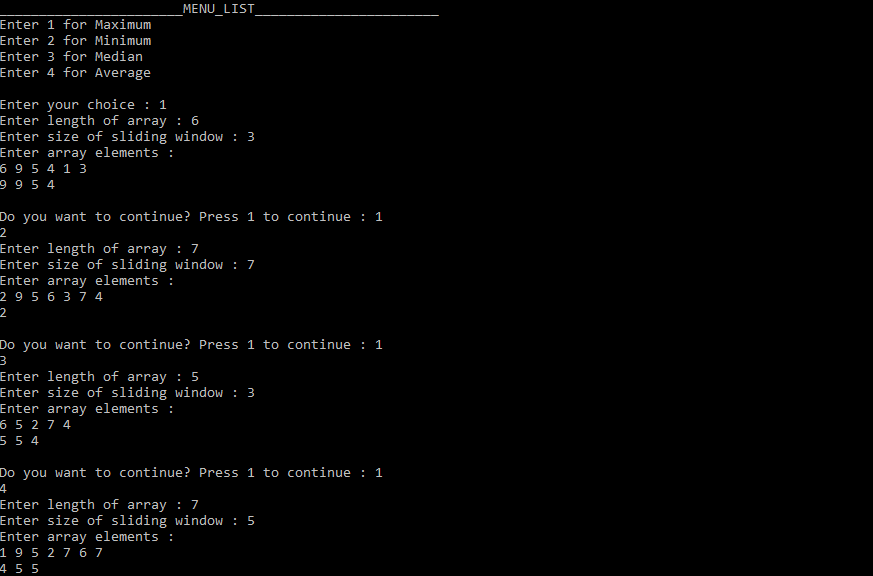
printf("\nDo you want to continue? Press 1 to continue : ");

scanf("%d",&c);

}while(c==1);

}

**OUTPUT :**

****

7)

**PROBLEM STATEMENT :**

Take an arbitrary matrix of positive integers say 128X128.Also take integer matrices of size 3X3 and 5X5.Find an output matrix of size 128X128 by multipying the small matrix with the corresponding submatrix of the large matrix with the centre of the small matrix placed at the individual positions within the large matrix.Explain how you will handle the boundary values.

**APPROACH :**

First we created matrix of size provided by user. Then we created an empty matrix to store the product after multiplication. For every small matrix we considered the corresponding sub-matrix and multiplied them as we multiply two matrices.

**CODE :**

#include<stdio.h>

#include<stdlib.h>

int dimensions()

{

int r;

printf("Enter the number of rows(cols same as rows) = ");

scanf("%d",&r);

return r;

}

int\*\* dynamic\_allocation(int r)

{

int \*\*p,i,j;

p=(int \*\*)malloc(sizeof(int\*)\*r);

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

p[i]=(int \*)malloc(sizeof(int)\*r);

printf("Successful memory allocation\n");

return p;

}

int\*\* accept\_values(int\*\* p,int r)

{

int i,j;

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

{

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

{

printf("Enter element at position p[%d][%d] = ",i,j);

scanf("%d",\*(p+i)+j);

}

}

printf("Successful accepted value\n");

return p;

}

int out\_of\_range(int i,int B\_MAT\_N){

if(i<0 || i>=B\_MAT\_N) return 1;

else return 0;

}

int\*\* mul(int\*\* n,int \*\*m,int\*\*p,int B\_MAT\_N,int S\_MAT\_N)

{

printf("Inside mul\n");

int row,col,s\_col,val = 0,r\_index;

int s\_mat\_mid = (S\_MAT\_N-1)/2;

for(row=0; row<B\_MAT\_N; row++){

for(col=0; col<B\_MAT\_N; col++){

val = 0;

for(s\_col=0; s\_col<S\_MAT\_N;s\_col++){

r\_index = row-s\_mat\_mid+s\_col;

if(!out\_of\_range(r\_index,B\_MAT\_N))

val = val+n[r\_index][col]\*m[s\_mat\_mid][s\_col];

}

p[row][col] = val;

printf("%d \n",p[row][col]);

}

}

return p;

}

/\*void multiply(int mat[B\_MAT\_N][B\_MAT\_N], int sub\_m[S\_MAT\_N][S\_MAT\_N], int out[B\_MAT\_N][B\_MAT\_N]){

int s\_mat\_mid = (S\_MAT\_N-1)/2;

printf("Mid = %d\n",s\_mat\_mid);

for(int row=0; row<B\_MAT\_N; row++){

for(int col=0; col<B\_MAT\_N; col++){

int val = 0;

for(int s\_col=0; s\_col<S\_MAT\_N;s\_col++){

int r\_index = row-s\_mat\_mid+s\_col;

if(!out\_of\_range(r\_index))

val = val+mat[r\_index][col]\*sub\_m[s\_mat\_mid][s\_col];

}

out[row][col] = val;

}

}

}\*/

void print(int\*\* p,int r)

{

int i,j;

printf("The matrix elements are = \n");

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

{

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

printf("%d ",\*(\*(p+i)+j));

printf("\n");

}

printf("\n");

}

int main()

{

int matrix\_size,mask\_width;

int \*\*n,\*\*m,\*\*p;

matrix\_size = dimensions();//matrix size

mask\_width = dimensions();//mask or kernel size;

n = dynamic\_allocation(matrix\_size);

m = dynamic\_allocation(mask\_width);

p = dynamic\_allocation(matrix\_size);

n = accept\_values(n,matrix\_size);

m = accept\_values(m,mask\_width);

p = mul(n,m,p,matrix\_size,mask\_width);

print(n,matrix\_size);

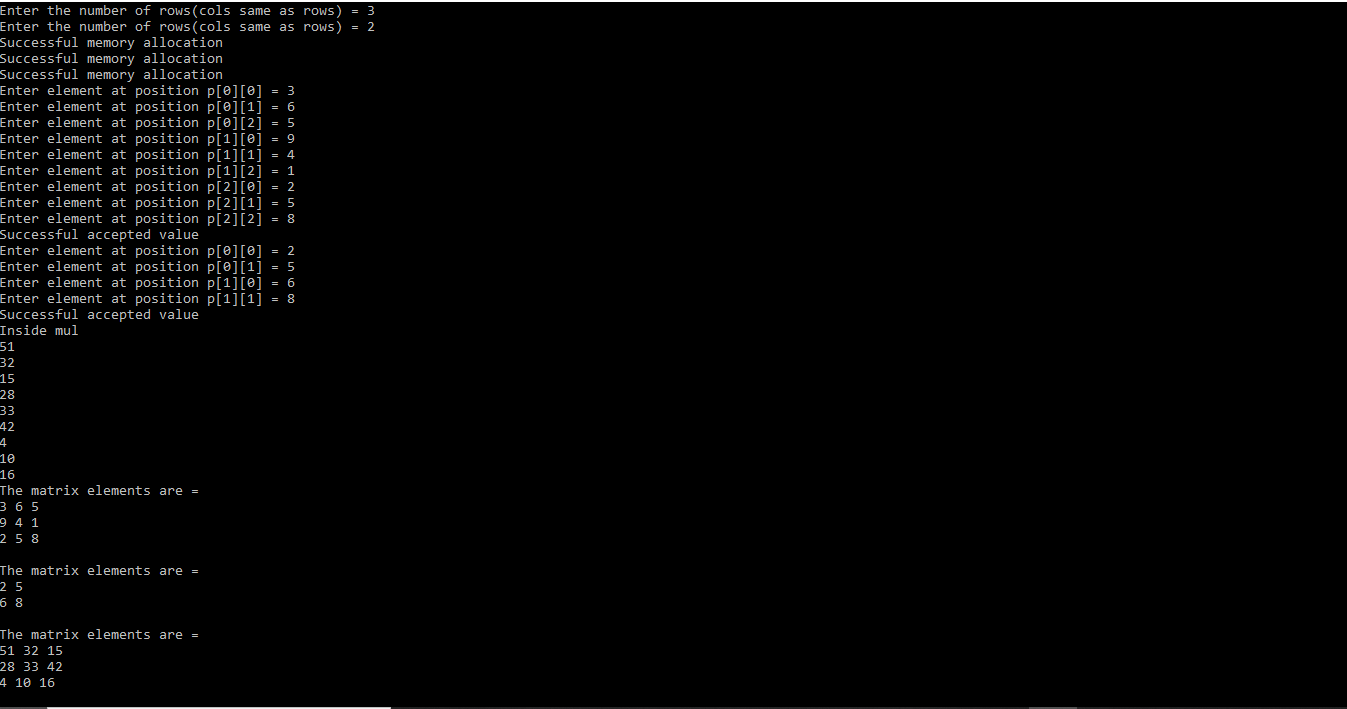
print(m,mask\_width);

print(p,matrix\_size);

return 0;

}

**OUTPUT :**

****8)

**PROBLEM STATEMENT :**

Find whether an array is sorted or not and the sorting order.

**APPROACH :**

First we sorted the given array and store it into anther array. So, now we have the original and the sorted array. Then we check the original array with the sorted array elements in normal direction and in reverse direction. If the given array is sorted then it will match with the sorted version of it's either in forward direction i.e., ascending order or reverse direction i.e.,in decreasing order , and from here we can also see whether the elements of the given array are equal or not.

**ALGORITHM :**

Sorting Algorithm:

for i =0 to n-1:

for j=i+1 to n-1:

if(a[i]>a[j]) swap(a[i],a[j])

**CODE :**

#include <stdio.h>

#include <stdlib.h>

#define MAX 10

int\* sorta(int a[],int n){

int i,j,temp;

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

for(j=i+1; j<n; j++){

if(a[i]>a[j]){

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

}

return a;

}

void compare(int a[],int b[],int n){

int i,k1=0,k2=0;

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

if(a[i]!=b[i]){

k1 = 1;

break;

}

}

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

if(a[i]!=b[n-1-i]){

k2 = 1;

break;

}

}

if(k1 == 1 && k2 == 1) printf("Not sorted.\n");

if(k1 == 0 && k2 == 1) printf("Ascending Order Sorted.\n");

if(k2 == 0 && k1 == 1) printf("Decending Order Sorted.\n");

if(k1 == 0 && k2 == 0) printf("All the elements are same.\n");

}

int main(){

int i,n,a[MAX],\*b,c[MAX];

int j;

printf("Enter the number of elements :\n");

scanf("%d",&n);

j = 0;

while(j<4){

printf("Enter the elements:\n");

for(i=0; i<n; i++) scanf("%d",&a[i]);

for(i=0; i<n; i++) c[i] = a[i];

b = sorta(c,n);

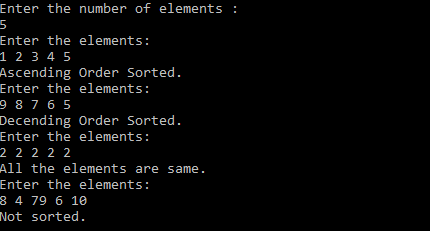
compare(a,b,n);

j++;

}

}

**OUTPUT :**

****

9)

**PROBLEM STATEMENT :**

Given two sorted arrays, write a function to merge the arrays in sorting order.

**APPROACH :**

After recieving the two arrays from the user, we created another empty array , whose size is equal to the size of sum of size of two input arrays. Then we sorted the final array and displayed it.

**ALGORITHM :**

Sorting Algorithm:

for i =0 to n-1:

for j=i+1 to n-1:

if(a[i]>a[j]) swap(a[i],a[j])

**CODE :**

#include <stdio.h>

#include <stdlib.h>

#define MAX 10

void insert(int a[],int n){

int i;

printf("Enter the elements\n");

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

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

}

}

void sort(int a[],int n){

int i,j,temp;

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

for(j=i+1; j<n; j++){

if(a[i]>a[j]){

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

}

}

void copy(int a[],int b[],int c[],int m,int n){

int i,k = 0;

for(i=0; i<m; i++) c[k++] = a[i];

for(i=0; i<n; i++) c[k++] = b[i];

}

void display(int a[],int n){

int i;

printf("The array after sorting is:\n");

for(i=0; i<n; i++) printf("%d ",a[i]);

printf("\n");

}

int main(){

int s1,s2;

printf("Enter the number of entries of the two arrays:\n");

scanf("%d %d",&s1,&s2);

int a[MAX],b[MAX],c[2\*MAX];

insert(a,s1);

insert(b,s2);

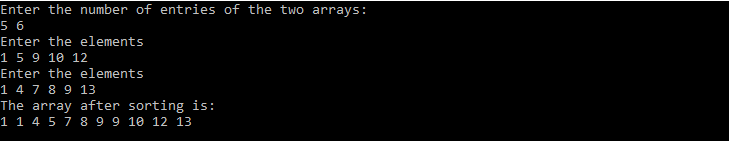
copy(a,b,c,s1,s2);

sort(c,s1+s2);

display(c,s1+s2);

}

**OUTPUT :**

****

**ASSIGNMENT 3**

1)

**PROBLEM STATEMENT :**

Implement the following functions of ADT Linked List using singly linked list as a header

file:

init\_l(cur) – initialise a list

empty\_l(head) – boolean function to return true if list pointed to by head is empty

atend\_l(cur) – boolean function to return true if cur points to the last node in the list

insert\_front(target, head) – insert the node pointed to by target as the first node of the list

pointed to by head

insert\_after(target, prev) – insert the node pointed to by target after the node pointed to by prev

delete\_front(head) – delete the first element of the list pointed to by head

delete\_after(prev) – delete the node after the one pointed to by prev.

**APPROACH :**

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.

A linked list is represented by a pointer to the first node of the linked list. The first node is called the head. If the linked list is empty, then the value of the head is NULL.  
Each node in a list consists of at least two parts:  
1) data  
2) Pointer (Or Reference) to the next node  
In C, we can represent a node using structures. Below is an example of a linked list node with integer data.

**ALGORITHM :**

**nodetype\* init\_l()** //initialize a head node

**intempty\_l(nodetype \*head)**//Return True(1) if head ==NULL else False(0)

**intatend\_l(nodetype \*cur)** //Create a new node and put value in it and attach it at the last

**void insert\_front(nodetype \*target, nodetype \*\*phead)** // Create a new node and put value in it and attach it at the front of the list

**void insert\_after (nodetype \*target, nodetype \*prev)** // Create a new node and put value in it and attach it after a given target

**void delete\_front( nodetype \*\*phead)**// Delete a node from the front and necessarily update the head pointer

}

**CODE :**

#include <stdio.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE 0

typedefstructnodetag{

float val;

structnodetag\* next;

}node;

void init\_l(node\* head){

head = NULL;

return;

}

intempty\_l(node\* head){

if(head == NULL) return TRUE;

else return FALSE;

}

intatend\_l(node\* tail){

if(tail->next == NULL) return TRUE;

else return FALSE;

}

void insert\_front(node\* target, node\*\* head){

target->next = \*head;

(\*head) = target;

}

void insert\_after(node\* target, node\* prev){

target->next = prev->next;

prev->next = target;

}

void delete\_front(node\*\* head){

node\* temp = \*head;

(\*head) = (\*head)->next;

free(temp);

}

void delete\_after(node\* prev){

node\* temp = prev->next;

prev->next = temp->next;

free(temp);

}

node\* create(float var){

node\* temp = (node\*)malloc(sizeof(node));

temp->next = NULL;

temp->val =var;

return temp;

}

void print\_l(node\* head){

node\* temp = head;

printf("\n");

while(temp != NULL){

printf("%f<->",temp->val);

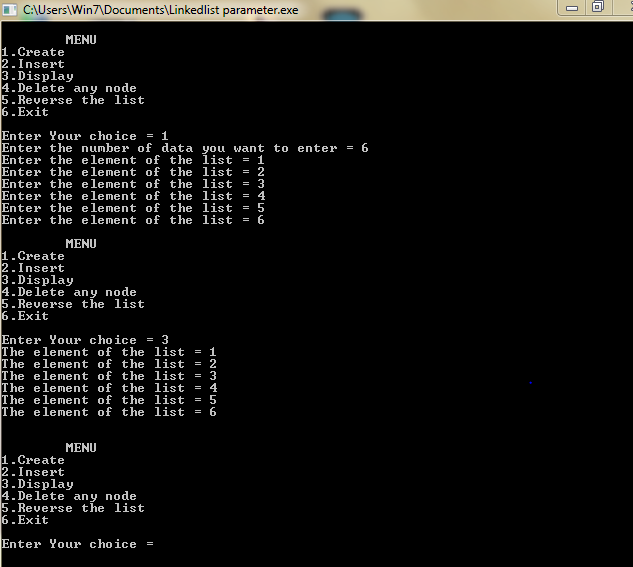
temp = temp->next;

}

printf("\n");

}

**OUTPUT :**

****

2)

**PROBLEM STATEMENT :**

Read integers from a file and arrange them in a linked list (a) in the order they are read, (b) in reverse order. Show the lists by printing.

**APPROACH :**

A file is given containing a list of numbers. Our task is to create two lists one by inserting number one after another and other by one before another using the functions insert\_after( for the same order ) and insert\_front ( for reverse order ).

**ALGORITHM :**

1. Open the given file.
2. Create two head pointer for two lists.
3. Insert the values in the list using the insert\_after() and insert\_front() functions.
4. Close the file.

**CODE :**

#include<stdio.h>

#include<stdlib.h>

struct node{

int num;

struct node\* link;

};

void print(struct node\* head)

{

struct node \*temp = head;

printf("List is :\n");

while(temp!=NULL)

{

printf(" %d",temp->num);

temp = temp->link;

}

printf("\n");

}

void reverseprint(struct node\* head)

{

if(head==NULL) return;

reverseprint(head->link);

printf(" %d",head->num);

}

int main()

{

int num;

struct node\* head = NULL;

struct node\* temp;

FILE \*fp;

fp = fopen("list.txt","r");

while(fscanf(fp,"%d",&num)!=EOF){

if(head==NULL)

{

head = (struct node \*)malloc(sizeof(struct node));

head->num = num;

head->link = NULL; // links the address field to NULL

temp = head;

}

else{struct node\* temp1 = (struct node \*)malloc(sizeof(struct node));

temp1->num = num; // links the num field of fnNode with num

temp1->link = NULL; // links the address field of fnNode with NULL

temp->link = temp1; // links previous node i.e. tmp to the fnNode

temp = temp->link;

}

}

print(head);

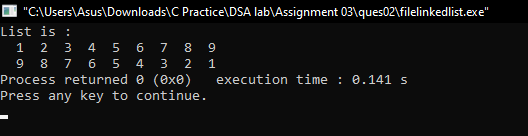
reverseprint(head);

}

**LIMITATIONS :**

1. If we do not close the file after theoperation, it may cause error later.
2. If the file does not exist, it will show error.
3. Only float type is applicable for this type, other types are not available.

**OUTPUT :**

****

3)

**PROBLEM STATEMENT :**

Implement the following functions in a menu-driven C program using the data structure operation of Singly Linked List in the header file developed in problem 1:

a) print a list (i) in the same order, (ii) in the reverse order.

b) find the size of a list in number of nodes

c) check whether two lists are equal

d) search for a key in (i) an unordered list, (ii) an ordered list( Return the node if key found and delete the node from original list)

e) append a list at the end of another list. f) delete the nth Node, last node and first node of a list.

g) check whether a list is ordered h) merge two sorted lists

i) insert a target node in the beginning, before a specified node and at the end of the list (sorted and unsorted).

j) remove duplicates from a linked list (sorted and unsorted)

k) swap elements of a list pairwise l) move last element to front of a list

m) delete alternate nodes of a list

n) rotate a list

o) delete a list.

p) reverse a list.

q) sort a list.

**APPROACH :**

1. **Print the list:** To print the list in the same order just traverse through the list and print it. But to print it in reverse order put the elements in a stack the pop the elements and print it.
2. **Size of a list:** Initialize a list node counter to zero, then traverse through the list and increment the counter accordingly.
3. **Check whether two lists are equal:** Initialise two pointer for two list and traverse through it. Whenever they are unequal return false otherwise true.
4. **Search for a key:** Weather the list is sorted or unsorted we have to traverse through it and find the key.
5. **Append a list at the end of another list:** Set the pointer to the end of one list then link the last node to the head of second list;
6. **Delete the nth Node, last node and first node of a list:** To delete the nth node just link (n-1)th node and (n+1)th node and free the nth node. To delete last node, the last but one node is linked to null pointer and make the last node free. To delete the first node, we have to pass the reference of head. Here we have to update the address of head pointer as well.
7. **Check whether a list is ordered:** First check whether the list is ascending or descending using the first two elements, then check the whole list.
8. **Merge two sorted lists:** Traverse two pointer, in two list check the elements, which is grater insert that in the new list and move the pointer to the next node and proceed.
9. **Insert a target node in the beginning, before a specified node and at the end of the list:** To insert a node before a specified target, we have to link the previous node to the new node and new node to the target node. But to insert in the beginning we have to replace the address of head pointer, so we hae to pass the reference of the head pointer.
10. **Remove duplicates from a linked list (sorted and unsorted):** To remove the duplicate element of a sorted list remove the same elements except the first one. For unsorted list traverse two pointer. First pointer any element and second one will find and delete the duplicates.
11. **Swap elements of a list pairwise:** There will be two pointers one ahead another and they will swap once and incremented twice till end.
12. **Move last element to front of a list:** Move a pointer to the last and one to the last but one. Then link the second last node to null and link the last node to the head and set the head address to the current pointer.
13. **Delete alternate nodes of a list:** Make to pointer one ahead another. The second pointer will delete the node and then the first node will be incremented once and set the second pointer one ahead of the first one and repeat.
14. **Delete a list:** Use two pointers, first one will point to the node to be deleted and second one will point to the node linked with it. First pointer will free the memory then first pointer will take the value of second pointer and the second pointer will be incremented.
15. **Reverse a list:** It will take three pointer pointing to three succeeding nodes. Now second will indicate the first and then first will take the value of second and second will take the value of third and third will be incremented.
16. **Sort a list:** We can swap the nodes of a list using two pointers. So now we can apply any sorting algorithm.

**CODE:**

DECLARE.h

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

struct node{

int data;

struct node\* next;

}\*temp1,\*temp,\*head;

typedef struct node NODE;

struct node\* create\_list(struct node\* head);

void print\_same(struct node\* head);

void reverse\_print(struct node\* head);

void size(struct node\* head);

bool equality(struct node\*head1,struct node\*head2);

bool search\_sorted(struct node\* head, int x);

void split(NODE\* source, NODE\*\* frontsp, NODE\*\* backsp);

NODE \* merge\_sorted(NODE \* headd1, NODE\* headd2);

NODE \* merge\_sort(NODE \* head);

bool search\_unsorted(NODE\* head, int x);

void append\_list(NODE\* head1,struct node\* head2);

NODE\* delete\_nth(NODE\* head,int n);

struct node\* delete\_front(struct node\* head);

struct node\* delete\_last(struct node\* head);

bool ascend(struct node\* head);

bool descend(struct node \* head);

bool is\_sorted(struct node\* head);

struct node\* insert\_front(struct node\* head,int x);

NODE\* insert\_n(NODE\* head,int data,int position);

NODE \* insertAtEnd(NODE\* head1,int x);

NODE \* remove\_dup(NODE \* head);

NODE \* moveLtoF(NODE \* head);

NODE \* del\_alt(NODE \* head);

NODE \* moveFtoL(NODE \* head);

NODE \* rotate(NODE\* head);

void del\_list(NODE\* head);

struct node\* recursion(struct node\* head);

bool not\_equal(struct node \* head1,struct node \* head2);

void deleteByValue(NODE \* head , int value );

Linkedlist.h

#include"declare.h"

struct node\* create\_list(struct node\* head)

{

int num,i;

while(i)

{

printf("Enter data.\n");

scanf("%d",&num);

if(head==NULL)

{

head = (struct node \*)malloc(sizeof(struct node));

head->data = num;

head->next = NULL; // nexts the address field to NULL

temp = head;

}

else{struct node\* temp1 = (struct node \*)malloc(sizeof(struct node));

temp1->data = num; // nexts the num field of fnNode with num

temp1->next = NULL; // nexts the address field of fnNode with NULL

temp->next = temp1; // nexts previous node i.e. tmp to the fnNode

temp = temp->next;

}

printf("Enter 1 to enter more.\n");

scanf("%d",&i);

}

return head;

}

void print\_same(struct node\* head)

{

struct node \*temp = head;

printf("List is :\n");

while(temp!=NULL)

{

printf(" %d",temp->data);

temp = temp->next;

}

printf("\n");

}

void reverse\_print(struct node\* head)

{

if(head==NULL) return;

reverse\_print(head->next);

printf(" %d",head->data);

return ;

}

void size(struct node\* head)

{

int i =0;

while(head!=NULL)

{

head = head->next;

i++;

}

printf("%d",i);

}

bool equality(struct node\*head1,struct node\*head2)

{

int flag =0;

while(head1!=NULL && head2!= NULL)

{

if(head1->data!=head2->data){ return false;}

head1 = head1->next;

head2 = head2->next;

}

return (head1 ==NULL && head2 == NULL);

}

bool search\_sorted(struct node\* head, int x)

{

struct node\* current = head; // Initialize current

while (current != NULL)

{

if (current->data == x)

return true;

current = current->next;

}

return false;

}

void split(NODE\* source, NODE\*\* frontsp, NODE\*\* backsp)

{

NODE\* fast;

NODE\* slow;

slow = source;

fast = source->next;

while (fast != NULL) {

fast = fast->next;

if (fast != NULL) {

slow = slow->next;

fast = fast->next;

}

}

\*frontsp = source;

\*backsp = slow->next;

slow->next = NULL;

}

NODE \* merge\_sorted(NODE \* headd1, NODE\* headd2)

{

NODE \*s;

NODE \*new\_head=NULL;

if(headd1->data>=headd2->data)

{

s = headd2;

headd2 = s->next;

}

else{

s = headd1;

headd1 = s->next;

}

new\_head = s;

while(headd1!=NULL && headd2!=NULL)

{

if(headd1->data>=headd2->data) //big

{

s->next= headd2;

s = headd2;

headd2 = headd2->next;

}

else {//small

s->next = headd1; // s->next = head2;

s = headd1; // s = head2;

headd1 = headd1->next;

}

}

if(headd1==NULL) s->next = headd2;

if(headd2==NULL) s->next = headd1;

// print\_same(new\_head);

return new\_head;

}

NODE \* merge\_sort(NODE \* head)

{

NODE \* temp = head;

NODE \* a ;

NODE \* b;

if ((temp == NULL) || (temp->next == NULL)) {

return head;}

split(head,&a,&b);

a = merge\_sort(a);

b = merge\_sort(b);

head = merge\_sorted(a,b);

return head;

}

bool search\_unsorted(NODE\* head, int x)

{

head = merge\_sort(head);

return search\_sorted(head,x);

}

void append\_list(NODE\* head1,struct node\* head2)

{

NODE \*temp=head1;

while(temp->next!=NULL) {temp = temp->next;}

temp->next = head2;

print\_same(head1);

}

NODE\* delete\_nth(NODE\* head,int n)

{

int i;

NODE\* temp1 = head;

if(n==1)

{

head=head->next;

return head;

}

for(i=0;i<n-2;i++)

{

temp1 = temp1->next;

}

NODE\* temp2 = temp1->next;

temp1->next=temp2->next;

free(temp2);

return head;

}

struct node\* delete\_front(struct node\* head)

{

head=head->next;

return head;

}

struct node\* delete\_last(struct node\* head)

{

int i;

struct node\* temp1 = head;

while(temp1->next->next!=NULL)

{

temp1 = temp1->next;

}

struct node\* temp2 = temp1->next;

temp1->next=temp2->next;

free(temp2);

return head;

}

bool ascend(struct node\* head){

while(head!=NULL && head->next != NULL)

{

if(head->data>head->next->data) {return true;} //NOt Ascend

head = head->next;

}

}

bool descend(struct node \* head){

while(head!=NULL && head->next != NULL)

{

if(head->data<head->next->data) {return true;} //NOt DEScend

head = head->next;

}

}

bool is\_sorted(struct node\* head)

{

int i;

if(ascend(head))

{ if(descend(head))

{

printf("Array unordered.\n");

}

else{printf("Array in Descending order.");}

}

else{

printf("Array in Ascending order.");

}

}

struct node\* insert\_front(struct node\* head,int x)

{

struct node\* temp = (struct node\*)malloc(sizeof(struct node));

// \*(temp).data = x;

temp->data = x;

temp->next = NULL;

// temp->next = head;

// \*(temp).next = NULL;

if(head!=NULL) temp->next = head;

head = temp;

// temp-> = x;

return head;

}

NODE\* insert\_n(NODE\* head,int data,int position)

{

int i;

NODE\* temp1 = (NODE\*)malloc(sizeof(NODE));

temp1->data=data;

temp1->next = NULL;

if(position==1)

{

temp1->next = head;

head = temp1;

return head;

}

NODE\* temp2 = head;

for(i=0;i<position-2;i++)

{

temp2= temp2->next;

}

temp1->next =temp2->next;

//if(temp2->next == NULL)// printf("yes");

temp2->next = temp1;

// if(temp1->next == NULL)// printf("not");

return (head);

}

NODE \* insertAtEnd(NODE\* head1,int x)

{

NODE \* temp1= head1;

NODE\* temp = (NODE\*)malloc(sizeof(NODE));

temp->data = x;

temp->next = NULL;

while(temp1->next!=NULL)

{

temp1= temp1->next;

}

temp1->next = temp;

return (head1);

}

NODE \* remove\_dup(NODE \* head)

{

if(head == NULL) { printf("List Empty."); return head; }

NODE \* temp = head;

NODE \*temp1 = NULL;

while(temp->next !=NULL)

{

if(temp->data == temp->next->data)

{

temp1 = temp->next->next;

free(temp->next);

temp->next = temp1;

}

else{

temp = temp->next;

}

}

print\_same(head);

return head;

}

NODE \* moveLtoF(NODE \* head)

{

NODE \* temp1= head;

NODE \* temp2= head->next;

NODE \* temp3 = head;

while(temp2->next!=NULL)

{

temp2= temp2->next;

temp1= temp1->next;

}

head = temp2;

temp2->next = temp3;

temp1->next = NULL;

print\_same(head);

return head;

}

NODE \* del\_alt(NODE \* head)

{

NODE \*temp = head;

NODE \*temp1 = head->next;

while(temp!=NULL && temp1!=NULL)

{

temp->next = temp1->next;

free(temp1);

temp= temp->next;

if(temp!=NULL)

{

temp1= temp->next;

}

}

return head;

}

NODE \* moveFtoL(NODE \* head)

{

NODE \* temp1 = head;

NODE \* temp2 = head->next;

while(temp1->next!=NULL)

{

temp1= temp1->next;

}

temp1->next = head;

head = temp2;

temp1->next->next =NULL;

return head;

}

NODE \* rotate(NODE\* head)

{

int n,i;

printf("Enter key value:\n");

scanf("%d",&n);

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

head = moveFtoL(head);

}

print\_same(head);

}

void del\_list(NODE\* head)

{

if(head==NULL) return ;

del\_list(head->next);

free(head);

printf("List deleted.");

}/\*

NODE \*recursion(NODE \*head) {

NODE \*temp1 = (NODE\*)malloc(sizeof(NODE));

NODE \*temp2 = (NODE\*)malloc(sizeof(NODE));

if (head->next == NULL) {

return head;

}

else {

temp1 =recursion(head->next);

temp2 =head->next;

temp2->next = head;

head->next = NULL;

}

return temp1;

};\*/

struct node\* recursion(struct node\* head){

if(head->next == NULL){

return head;

}

struct node\* newHead = recursion(head->next);

head->next->next = head;

head->next = NULL;

return newHead;

}

bool not\_equal(struct node \* head1,struct node \* head2)

{

return !equality(head1,head2);

}

void deleteByValue(NODE \* head , int value )

{

print\_same(head);

NODE \* temp= head;

while(temp->next->data != value)

{

temp = temp->next;

}

NODE \* temp2 = temp->next;

temp->next = temp2->next;

free(temp2);

print\_same(head);

}

/\*

NODE\* insert\_beg(NODE\* head,int x)

{

NODE\* temp = (NODE\*)malloc(sizeof(NODE));

temp->data = x;

temp->next = NULL;

if(head!=NULL) temp->next = head;

head = temp;

print\_same(head);

return head;

}

NODE\* delete(NODE\* head,int n)

{

// n=1;

int i;

NODE\* temp1 = head;

if(n==1)

{

head=head->next;

return head;

}

for(i=0;i<n-2;i++)

{

temp1 = temp1->next;

}

NODE\* temp2 = temp1->next;

temp1->next=temp2->next;

free(temp2);

return head;

}\*/

Main.c

#include<stdio.h>

#include<stdlib.h>

#include"linklist.h"

NODE \*head1,\*head2,\*head3,\*head4;

int main()

{

printf("Enter data to create list.\n");

head1 = insert\_front(head1,90);

head1 = insert\_front(head1,90);

head1 = insert\_front(head1,70);

head1 = insert\_front(head1,70);

head1 = insert\_front(head1,30);

head1 = insert\_front(head1,10);

head2 = insert\_front(head2,100);

head2 = insert\_front(head2,3);

head2 = insert\_front(head2,56);

head2 = insert\_front(head2,25);

head2 = insert\_front(head2,5);

head2 = insert\_front(head2,58);

head2 = insert\_front(head2,173);

head2 = insert\_front(head2,957);

head2 = insert\_front(head2,463);

head3 = insert\_front(head3,110);

head3 = insert\_front(head3,90);

head3 = insert\_front(head3,70);

head3 = insert\_front(head3,70);

head3 = insert\_front(head3,30);

head3 = insert\_front(head3,10);

head4 = insert\_front(head4,100);

head4 = insert\_front(head4,60);

head4 = insert\_front(head4,40);

head4 = insert\_front(head4,20);

int choice,n=2;

while(1)

{

printf("\n1. Enter 1 to print in same order.\n2."

"Enter 2 to print in reverse order.\n3."

"Enter 3 to know size.\n40."

"Enter 4 to check Equality.\n5."

"Enter 5 to search in unsorted.\n6."

"Enter 6 to search in sorted.\n7."

"Enter 6 to search in sorted.\n7."

"Enter 7 to append lists.\n8."

"Enter 8 to delete nth node.\n9."

"Enter 9 to delete front.\n10."

"Enter 10 to delete last.\n11."

"Enter 11 to check sorted or not.\n12."

"Enter 12 to sort .\n13."

"Enter 13 to insert at n.\n14."

"Enter 14 to insert front.\n15."

"Enter 15 to insert at end.\n16."

"Enter 16 to remove duplicate.\n17."

"Enter 17 to remove duplicate.\n18."

"Enter 18 to Move last node to front.\n19."

"Enter 19 to delete alternative node.\n20."

"Enter 20 to rotate.\n21."

"Enter 21 to delete list.\n22."

"Enter 22 to to do recursion.\n23."

"Enter 23 to sort.\n24."

"Enter 24 to exit.\n");

printf("\nEnter choice.\n");

scanf("%d",&choice);

switch(choice)

{

case 1: print\_same(head1);

break;

case 2: reverse\_print(head1);

break;

case 3: size(head1);

break;

case 4: equality(head1,head1)?printf("Same"):

printf("Not same");

break;

case 5 : search\_unsorted(head2,300)?printf("Found"):

printf("Not Found");

break;

case 6 : search\_sorted(head4,20)?printf("Found"):

printf("Not Found");

break;

case 7: append\_list(head1,head2);

break;

case 8: delete\_nth(head1,n);

break;

case 9: delete\_front(head1);

break;

case 10: delete\_last(head1);

break;

case 11 : is\_sorted(head1);

break;

case 12 : head3 = merge\_sorted(head3,head4);

break;

case 13 : head1 = insert\_n(head1,5,7); print\_same(head1);

break;

case 14: head1 = insert\_front(head2,499); print\_same(head1);

break;

case 15: insertAtEnd(head1,10);

break; // add end

case 16 : remove\_dup(head1);

break;

case 17: remove\_dup(head1);

break;

case 18 : moveLtoF(head1);

break;

case 19 : del\_alt(head1);

print\_same(head1);

break;

case 20 : rotate(head1);

break;

case 21: del\_list(head1);

break;

case 22: head2 = recursion(head2);

print\_same(head2);

break;

case 23 :head2 = merge\_sort(head2);

print\_same(head2);

break;

case 24: exit(0);

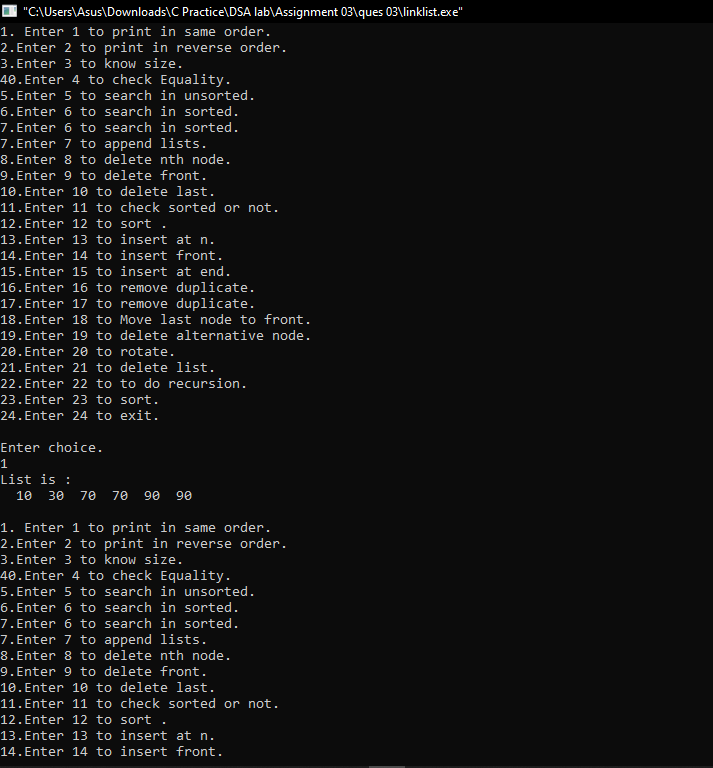
}

}

return 0;

}

**OUTPUT :**

****

**DOUBLY LINK LIST**

**Double.h**

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

struct Node{

int data;

struct Node \*next;

struct Node \*prev;

};struct Node \*head1,\*last,\*head2,\*head,\*head3;

typedef struct Node node;

struct Node \* GetNewNode(int x)

{

struct Node\* NewNode= (struct Node \*)malloc(sizeof(struct Node));

NewNode->data = x;

NewNode->prev=NULL;

NewNode->next=NULL;

return NewNode;

}

void create()

{

int n;

printf("How many nodes do you want.\n");

scanf("%d",&n);

int i, data;

node \*newNode;

if(n >= 1)

{

head = (node \*)malloc(sizeof(node));

printf("Enter data of 1 node: ");

scanf("%d", &data);

head->data = data;

head->prev = NULL;

head->next = NULL;

last = head;

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

{

newNode = (node \*)malloc(sizeof(node));

printf("Enter data of %d node: ", i);

scanf("%d", &data);

newNode->data = data;

newNode->prev = last; // Link new node with the previous node

newNode->next = NULL;

last->next = newNode; // Link previous node with the new node

last = newNode; // Make new node as last node

}

printf("\nDOUBLY LINKED LIST CREATED SUCCESSFULLY\n");

}

}

void Print(node\* head)

{

int check = 0;

// int check = check\_empty(head);

if(check != 1)

{

struct Node \* temp = head;

printf("List is : \n");

while(temp!= NULL)

{

printf(" %d",temp->data);

temp = temp->next;

}

printf("\n");

}

}

void ReversePrint()

{

struct Node \*temp = head;

if(temp==NULL) return ;

while(temp->next!=NULL) {temp=temp->next;}

printf("Reversed list is:\n");

while(temp!=NULL)

{

printf(" %d",temp->data);

temp=temp->prev;

}

printf("\n");

}

int size(node\* head)

{

int i=0;

node \* temp = head;

while(temp!=NULL)

{

i++;

temp = temp->next;

}

return i;

}

bool equality(node \* head1, node\* head2)

{

int flag =0;

while(head1!=NULL && head2!= NULL)

{

if(head1->data!=head2->data){ return false;}

head1 = head1->next;

head2 = head2->next;

}

return (head1 ==NULL && head2 == NULL);

}

bool search\_sorted(node\* head, int x)

{

node\* current = head; // Initialize current

while (current != NULL)

{

if (current->data == x)

return true;

current = current->next;

}

return false;

}

node\* split(node\* source , node\*\* frontsp , node\*\* backsp)

{

node \* slow = source;

node \* fast = source->next;

while(fast!=NULL)

{

fast = fast->next;

if(fast!=NULL)

{

slow = slow->next;

fast = fast->next;

}

}

\*frontsp = source;

\*backsp = slow->next;

slow->next->prev = NULL;

slow->next = NULL;

}

node \* merge\_sorted(node \* head1, node\* head2)

{

node \* s;

node \* new\_head = NULL;

if(head1->data>head2->data)

{

s = head2;

head2 = head2->next;

}

else{

s = head1;

head1 = head1->next;

}

new\_head = s;

while(head1!=NULL && head2!=NULL)

{

if(head1->data > head2->data)

{

s->next = head2;

head2->prev = s;

s = head2;

head2 = head2->next;

}

else{

s->next = head1;

head1->prev = s;

s = head1;

head1 = head1->next;

}

}

if(head1==NULL) s->next = head2;

if(head2==NULL) s->next = head1;

return new\_head;

}

node \* merge\_sort(node \* head)

{

node \* temp = head;

node \* a;

node \* b;

if ((temp == NULL) || (temp->next == NULL)) {

return head;}

split(head,&a,&b);

a = merge\_sort(a);

b = merge\_sort(b);

head = merge\_sorted(a,b);

return head;

}

bool search\_unsorted(node\* head, int x)

{

head = merge\_sort(head);

return search\_sorted(head,x);

}

node\* append\_list(node\* head1,node\* head2)

{

node \*temp=head1;

while(temp->next!=NULL) {temp = temp->next;}

temp->next = head2;

head2->prev = temp;

return head1;

}

node\* delete\_front(node\* head)

{

node \* temp = head;

head=head->next;

if(head!=NULL)

{

head->prev = NULL;

}

free(temp);

return head;

}

void del\_las(node \* last)

{

printf("HII");

node \* todelete = last;

last = last->prev;

if(last!=NULL)

{

last->next = NULL;

}

free(todelete);

return ;

}

node\* delete\_last(node\* head)

{

node \* temp = head;

while(temp->next!=NULL)

{

temp = temp->next;

}

del\_las(temp);

return head;

}

node\* delete(node \* head,int n)

{

int check =0;

// int check = check\_empty(head);

if(check != 1)

{

int i;

if(n==1)

{ head = delete\_front(head);

return head;

}

struct Node\* temp = head;

for(i=0;i<n-2;i++) {temp = temp->next;}

struct Node\* temp1 = temp->next;

if(temp1->next==NULL)

{ del\_las(temp1);

return head; }

temp->next = temp1->next;

temp1->next->prev = temp1->prev;

return head;

}

}

bool ascend(node\* head){

while(head!=NULL && head->next != NULL)

{

if(head->data>head->next->data) {return true;} //NOt Ascend

head = head->next;

}

}

bool descend(node \* head){

while(head!=NULL && head->next != NULL)

{

if(head->data<head->next->data) {return true;} //NOt DEScend

head = head->next;

}

}

bool is\_sorted(node\* head)

{

int i;

if(ascend(head))

{ if(descend(head))

{

printf("Array unordered.\n");

}

else{printf("Array in Descending order.");}

}

else{

printf("Array in Ascending order.");

}

}

node\* InsertAtHead(node\* head,int x)

{

struct Node \* fnode = GetNewNode(x);

if(head==NULL)

{

head = fnode;

return head;

}

head->prev = fnode;

fnode->next = head;

head = fnode;

return head;

}

node \* InsertAtN(node\* head,int n)

{

int value,i;

node \* temp = head;

printf("Enter the value you want to insert,\n");

scanf("%d",&value);

if(n==1) {head = InsertAtHead(head,value); return head;}

node \*insert = GetNewNode(value);

for(i=0;i<n-2;i++)

{

temp = temp->next;

}

insert->prev = temp->next->prev;

insert->next = temp->next;

temp->next = insert;

return head;

}

node\* InsertAtTail(node\* head,int x)

{

struct Node \* lnode = GetNewNode(x);

if(head==NULL)

{

head = lnode;

return head;

}

struct Node \* temp = head;

while(temp->next!=NULL) {temp=temp->next;}

temp->next = lnode;

lnode->prev = temp;

return head;

}

node \* remove\_dup(node \* head)

{

if(head == NULL) { printf("List Empty."); return head; }

node \* temp = head;

node \*temp1 = NULL;

while(temp->next !=NULL)

{

if(temp->data == temp->next->data)

{

temp1 = temp->next->next;

if(temp->next->next!=NULL){

temp->next->next->prev = temp->next->prev;

}

free(temp->next);

temp->next = temp1;

}

else{

temp = temp->next;

}

}

return head;

}

node \* moveLtoF(node \* head)

{

node \* temp1= head;

node \* temp2= head->next;

node \* temp3 = head;

while(temp2->next!=NULL)

{

temp2= temp2->next;

temp1= temp1->next;

}

head = temp2;

head->next = temp3;

head->prev->next = NULL;

head->prev = NULL;

head->next->prev = head;

return head;

}

node \* del\_alt(node \* head)

{

node \*temp = head;

node \*temp1 = head->next;

while(temp!=NULL && temp1!=NULL)

{

temp->next = temp1->next;

temp1->next->prev = temp1->prev ;

free(temp1);

temp= temp->next;

if(temp!=NULL)

{

temp1= temp->next;

}

}

return head;

}

node \* moveFtoL(node \* head)

{

node \* temp1 = head;

node \* temp2 = head->next;

while(temp1->next!=NULL)

{

temp1= temp1->next;

}

temp1->next = head;

head = temp2;

head->prev = NULL;

temp1->next->next =NULL;

temp1->next->prev = temp1;

return head;

}

node\* rotate(node\* head)

{

int n,i;

printf("Enter key value:\n");

scanf("%d",&n);

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

head = moveFtoL(head);

}

return head;

}

void del\_list(node\* head)

{

if(head==NULL) return ;

del\_list(head->next);

free(head);

printf("List deleted.");

}

bool check\_empty(node\* head)

{

if(head==NULL)

return true;

return false;

}

bool is\_last(struct Node\* curr )

{

if(curr->next == NULL)

return true;

return false;

}

/\*

bool search\_unsorted(node\* head, int x)

{

node\* current = head; // Initialize current

while (current != NULL)

{

if (current->data == x)

return true;

current = current->next;

}

return false;

}

bool not\_equal(node \* head1,node \* head2)

{

return !equality(head1,head2);

}\*/

**Circular linked list**

**Circular.h**

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

struct Node{

int data;

struct Node \*next;

};struct Node \*head1,\*last,\*head2,\*head,\*head3,\*head4;

typedef struct Node node;

node \* GetNewNode(int x)

{

struct Node\* NewNode= (struct Node \*)malloc(sizeof(struct Node));

NewNode->data = x;

NewNode->next=NULL;

return NewNode;

}

node\* create\_list(node\* head)

{

int i, num,n=5;

node \*last, \*newnode;

if(n >= 1)

{

head = (node \*)malloc(sizeof(node));

printf(" Input data for node 1 : ");

scanf("%d", &num);

head->data = num;

head->next = NULL;

last = head;

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

{

newnode = (node \*)malloc(sizeof(node));

printf(" Input data for node %d : ", i);

scanf("%d", &num);

newnode->data = num;

newnode->next = NULL; // next address of new node set as NULL

last->next = newnode; // previous node is linking with new node

last = newnode; // previous node is advanced

}

last->next = head; //last node is linking with first node

}

return head;

}

void print\_same(node\* head)

{

node \*temp = head;

printf("List is :\n");

do{

printf(" %d",temp->data);

temp = temp->next;

}while(temp!=head);

printf("\n");

}

int size(node\* head)

{

node \*temp = head;

int i =0;

do{

temp = temp->next;

i++;

}while(temp!=head);

//printf("%d",i);

return i;

}

bool equality(node\*head1,node\*head2)

{

int i =0;

node \* temp1= head1;

node \* temp2= head2;

if(size(temp1)== size(temp2))

{

do

{

if(temp1->data!=temp2->data){ return false;}

temp1 = temp1->next;

temp2 = temp2->next;

}while(temp1!=head1 && temp2!= head2);

return (temp1 == head1 && temp2 == head2);

}

else {

return false;

}

}

bool search\_sorted(node\* head, int x)

{

int i=0;

node\* current = head; // Initialize current

do{

if(x<current->data)

{

return false;

}

else{

if(current->data==x){ return true;}

current = current->next;

}}while(current!=head);

return false;

}

bool search\_unsorted(node\* head, int x)

{

int i=0;

node\* current = head; // Initialize current

do{

if(current->data==x){ return true;}

current = current->next;

}while(current!=head);

return false;

}

node \* append\_list(node\* head1,node\* head2)

{

node \*temp=head1;

node \*temp1=head2;

do{

temp = temp->next;

}while(temp->next!=head1);

temp->next = head2;

do{

temp1 = temp1->next;

}while(temp1->next!=head2);

temp1->next = head1;

return head1;

}

node\* insert\_front(node\* head,int x)

{

node\* temp1 = (node\*)malloc(sizeof(node));

// \*(temp).data = x;

temp1->data = x;

temp1->next = head;

// temp->next = head;

// \*(temp).next = NULL;

if(head==NULL){ temp1->next = temp1; head = temp1; return head;}

node \*temp2 = head;

while(temp2->next != head)

{

temp2 = temp2->next;

}

temp2->next = temp1;

head = temp1;

// head->next = temp2;

// temp-> = x;

return head;

}

node\* insert\_n(node\* head,int data,int position)

{

printf("holy");

int i;

node\* temp1 = (node\*)malloc(sizeof(node));

temp1->data=data;

temp1->next = NULL;

if(position==1)

{

temp1->next = head;

head = temp1;

return head;

}

node\* temp2 = head;

for(i=0;i<position-2;i++)

{

temp2= temp2->next;

}

temp1->next =temp2->next;

temp2->next = temp1;

return (head);

}

node \* insertAtEnd(node\* head1,int x)

{

node \* temp1= head1;

node\* temp = (node\*)malloc(sizeof(node));

temp->data = x;

while(temp1->next!=head1)

{

temp1= temp1->next;

}

temp1->next = temp;

temp->next = head1;

return (head1);

}

node\* delete\_front(node\* head)

{

node\* temp = head;

node\* temp1 = head;

head=head->next;

temp = head;

do{

temp = temp->next;

}while(temp->next!=temp1);

temp->next = head;

free(temp1); //Issue

return head;

}

node\* delete\_nth(node\* head,int n)

{

int i;

node\* temp1 = head;

if(n==1)

{

head= delete\_front(head);

return head;

}

for(i=0;i<n-2;i++)

{

temp1 = temp1->next;

}

node\* temp2 = temp1->next;

temp1->next=temp2->next;

free(temp2);

return head;

}

node\* delete\_last(node\* head)

{

node\* temp = head;

node\* temp1 = head->next;

node\* temp2 = head;

do{

temp1 = temp1->next;

temp2 = temp2->next;

}while(temp1->next!=temp);

temp2->next=temp1->next;

free(temp1);

return head;

}

bool ascend(node\* head){

int i=0;

node\* temp = head;

do{

if(temp->data>temp->next->data) {return true;} //NoT ascend

temp = temp->next;

}while(temp->next->next!=head);

return false;

}

bool descend(node \* head){

int i=0;

node\* temp = head;

do{

if(temp->data<temp->next->data) {return true;} //NoT ascend

temp = temp->next;

}while(temp->next->next!=head);

return false;

}

bool is\_sorted(node\* head)

{

int i;

if(ascend(head))

{ if(descend(head))

{

printf("Array unordered.\n");

}

else{printf("Array in Descending order.\n");}

}

else{

printf("Array in Ascending order.\n");

}

}

node\* remove\_dup(node \* head)

{

if(head == NULL) { printf("List Empty."); return head; }

node \* temp = head;

node \*temp1 = NULL;

do

{

if(temp->data == temp->next->data)

{

temp1 = temp->next->next;

free(temp->next);

temp->next = temp1;

}

else{

temp = temp->next;

}

}while(temp->next !=head);

return head;

}

node \* moveLtoF(node\* head)

{

node \* temp1= head;

node \* temp2= head->next;

node \* temp3 = head;

do{

temp2= temp2->next;

temp1= temp1->next;

}while(temp2->next!=head);

head = temp2;

temp2->next = temp3;

temp1->next = temp2;

return head;

}

node \* del\_alt(node \* head)

{

node \*temp = head;

node \*temp1 = head->next;

do

{

temp->next = temp1->next;

free(temp1);

temp= temp->next;

if(temp!=NULL)

{

temp1= temp->next;

}

}while(temp!=head && temp1!=head);

return head;

}

node \* moveFtoL(node \* head)

{

node \* temp1 = head;

node \* temp2 = head->next;

do

{

temp1= temp1->next;

}while(temp1->next!=head);

temp1->next = head;

head = temp2;

temp1->next->next =head;

return head;

}

node \* rotate(node\* head)

{

int n,i;

printf("Enter key value:\n");

scanf("%d",&n);

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

head = moveFtoL(head);

}

return head;

}/\*

void del\_list(node\* head)

{

if(head==NULL) return ;

del\_list(head->next);

free(head);

printf("List deleted.");

}\*/

node\* recursion(node\* head){

if(head->next == NULL){

return head;

}

node\* newHead = recursion(head->next);

head->next->next = head;

head->next = newHead;

// newHead->next = head;

return newHead;

}

6)

**PROBLEM STATEMENT :**

Implement an application to find out the Inverted Index of a set of text files.

**APPROACH :**

To solve this we will need to use the idea of hashing at the higher

level and also at the lower level. By this i mean that, to locate a word in

terms of its presence in a particular doc. We will use hashing to store the

words starting will a particular letter . All the words in a particular doc

starting with a particular letter can then be stored in sequence with the use

of linked list which helps in new addition of new found words. All after we

have stored words we can start storing their location as per paragraphs and

then as per sentences. This basically requires storing at a lot of hashing levels

, but with the use of linked list we can add a particular words in step count

proportional to the length of word.

**ALGORITHM :**

We first create an array of 26 linked lists and then treat them as if

they are the letters of alphabet.

2. Now we connect 26 more linked lists with these lists.

3. Thus now onwards if we get any word starting with these two sequence

will be appended in the list associated with that list in ascending order.

4. So we scan a word.

5. We check for its first letter.

6. We enter the list of that node.

7. Again we search for the second letter and go to its node.

8. Now we start searching for that word iteratively in the list.

9. Just as we reach the words between which the word should lie, we

insert it there and write down its doc :para :line :position and come out.

10. In case we find the exact match we just append data inside the data

block of that word in first .

11. Now we go on repeating the steps 4,5,6,7,8,9 and 10.

**CODE :**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX\_WORD\_SIZE 10

#define STR\_ARRAY\_LEN 50

#define SENTINAL\_CHAR '~'

#define IS\_GREATER 1

#define IS\_LESSER -1

#define IS\_EQUAL 0

typedefstruct doc\_{

char doc\_name[MAX\_WORD\_SIZE];

struct doc\_\* next\_doc;

}doc;

typedefstructnode\_tag{

structnode\_tag\* prev;

char word[MAX\_WORD\_SIZE];

doc\* docs;

structnode\_tag\* next;

}node;

void format\_doc(FILE\* fp, char\* name){

FILE\* f\_new = fopen(name,"w");

fseek(fp,0,SEEK\_SET);

char c = '\0';

while(!feof(fp)){

char str[MAX\_WORD\_SIZE] = "\0";

intstr\_index = 0;

while(isalnum(c=fgetc(fp))){

if(c>='A' && c<='Z') c = c - 'A' + 'a';

str[str\_index++] = c;

}

while(str\_index>0 &&str\_index<MAX\_WORD\_SIZE-1)

str[str\_index++] = SENTINAL\_CHAR;

str[str\_index] = '\0';

if(str\_index != 0){

//printf("In format\_doc we're writing :: %s\n",str);

fwrite(str,sizeof(char),MAX\_WORD\_SIZE,f\_new);

}

}

fclose(f\_new);

return;

}

void sort\_strings(char str[STR\_ARRAY\_LEN][MAX\_WORD\_SIZE]){

for(inti=MAX\_WORD\_SIZE-1;i>=0; i--){

intcount[37] = {0};

for(int j=0; j<STR\_ARRAY\_LEN; j++){

if(isdigit(str[j][i])) count[str[j][i]-'0'+27]++;

else if(isalpha(str[j][i])) count[str[j][i] - 'a'+1]++;

else if(str[j][i] == SENTINAL\_CHAR) count[0]++;

}

intpos[37] = {0};

for(int m=1;m<37;m++){

pos[m]=pos[m-1]+count[m-1];

}

char temp[STR\_ARRAY\_LEN][MAX\_WORD\_SIZE];

for(int k=0; k<STR\_ARRAY\_LEN; k++){

if(isdigit(str[k][i])) strcpy(temp[pos[str[k][i]-'0'+27]++],str[k]);

else if(isalpha(str[k][i])) strcpy(temp[pos[str[k][i]-'a'+1]++], str[k]);

else strcpy(temp[pos[0]++],str[k]);

}

for(int l=0; l<STR\_ARRAY\_LEN; l++)

strcpy(str[l],temp[l]);

}

return;

}

intcomp\_strings(char str1[MAX\_WORD\_SIZE], char str2[MAX\_WORD\_SIZE]){

for(inti=0; i<MAX\_WORD\_SIZE-1; i++){

if(str1[i] == SENTINAL\_CHAR || str2[i] == SENTINAL\_CHAR){

if(str2[i]!= SENTINAL\_CHAR) return IS\_LESSER;

else if(str1[i] != SENTINAL\_CHAR) return IS\_GREATER;

else continue;

}

else if(isalpha(str1[i]) &&isalpha(str2[i])){

if(str1[i]>str2[i])

return IS\_GREATER;

else if(str1[i]<str2[i]) return IS\_LESSER;

else continue;

}

else if(isdigit(str1[i]) || isdigit(str2[i])){

if(!isdigit(str2[i])) return IS\_GREATER;

else if(!isdigit(str1[i])) return IS\_LESSER;

else continue;

}

}

return IS\_EQUAL;

}

doc\* create\_doc(const char doc\_name[MAX\_WORD\_SIZE]){

doc\* doc\_head = (doc\*)malloc(sizeof(doc));

strcpy(doc\_head->doc\_name, doc\_name);

doc\_head->next\_doc = NULL;

return doc\_head;

}

void append\_doc(node\* elem, char doc\_name[MAX\_WORD\_SIZE]){

doc\* doc\_tmp = elem->docs;

while(doc\_tmp != NULL){

if(strcmp(doc\_tmp->doc\_name,doc\_name) == IS\_EQUAL)

return;

if(doc\_tmp->next\_doc != NULL) doc\_tmp = doc\_tmp->next\_doc;

else break;

}

doc\_tmp->next\_doc = create\_doc(doc\_name);

}

node\* create\_node(const char str[MAX\_WORD\_SIZE],const char doc\_name[MAX\_WORD\_SIZE]){

node\* temp = (node \*)malloc(sizeof(node));

temp->prev = NULL;

strcpy(temp->word,str);

temp->docs = create\_doc(doc\_name);

temp->next = NULL;

return temp;

}

void insert\_front(node\*\* head, char str[MAX\_WORD\_SIZE],char doc\_name[MAX\_WORD\_SIZE]){

node\* temp = create\_node(str, doc\_name);

(\*head)->prev = temp;

temp->next = \*head;

\*head = temp;

}

void insert\_after(node\* elem, char str[MAX\_WORD\_SIZE],char doc\_name[MAX\_WORD\_SIZE]){

node\* temp = create\_node(str,doc\_name);

temp->next = elem->next;

temp->prev = elem;

if(temp->next!=NULL) temp->next->prev = temp;

elem->next = temp;

return;

}

void place\_elem(node\*\* head, char str[STR\_ARRAY\_LEN][MAX\_WORD\_SIZE], char doc\_name[MAX\_WORD\_SIZE]){

char duplicate[MAX\_WORD\_SIZE] ="\0";

strcpy(duplicate,str[0]);

intstart\_index = 0;

while(str[start\_index][0] == SENTINAL\_CHAR)

start\_index++;

if(\*head == NULL){

\*head = create\_node(str[start\_index],doc\_name);

node\* curr = \*head;

for(inti=start\_index+1; i<STR\_ARRAY\_LEN; i++){

if(comp\_strings(duplicate, str[i]) != IS\_EQUAL){

insert\_after(curr,str[i],doc\_name);

strcpy(duplicate,str[i]);

curr = curr->next;

}

else continue;

}

}

else{

if(comp\_strings(str[start\_index],(\*head)->word) == IS\_LESSER)

insert\_front(head,str[start\_index++],doc\_name);

node\* curr = \*head;

for(inti=start\_index; i<STR\_ARRAY\_LEN; i++){

while(comp\_strings(str[i],curr->word) == IS\_GREATER){

if(curr->next != NULL)

curr = curr->next;

else break;

}

if(comp\_strings(str[i],curr->word) == IS\_GREATER)

insert\_after(curr,str[i],doc\_name);

else if(comp\_strings(str[i],curr->word) == IS\_EQUAL)

append\_doc(curr,doc\_name);

else insert\_after(curr->prev,str[i],doc\_name);

}

}

return;

}

void print\_dict(node\* head){

while(head!= NULL){

for(inti=0; i<MAX\_WORD\_SIZE; i++)

if(head->word[i]!=SENTINAL\_CHAR) printf("%c",head->word[i]);

else printf(" ");

printf("\t");

doc\* tmp = head->docs;

while(tmp != NULL){

printf("<---->%s",tmp->doc\_name);

tmp = tmp->next\_doc;

}

printf("\n");

head = head->next;

}

return;

}

void i\_STR\_list(char str\_arr[STR\_ARRAY\_LEN][MAX\_WORD\_SIZE]){

for(inti=0; i<STR\_ARRAY\_LEN; i++){

for(int j=0; j<MAX\_WORD\_SIZE-1; j++)

str\_arr[i][j] = SENTINAL\_CHAR;

str\_arr[i][MAX\_WORD\_SIZE-1] = '\0';

}

return;

}

void dict\_add\_from(char doc\_name[MAX\_WORD\_SIZE], node\*\* head){

FILE\* fp;

fp = fopen(doc\_name,"r");

if(fp == NULL){

printf("No such file exists, skipping....\n");

return;

}

format\_doc(fp,"formatted");

fclose(fp);

fp = fopen("formatted", "r");

char str[MAX\_WORD\_SIZE] = "\0";

char str\_arr[STR\_ARRAY\_LEN][MAX\_WORD\_SIZE];

inti = 0;

intfread\_c = 0;

i\_STR\_list(str\_arr);

\*/

while((fread\_c = fread(str,sizeof(char),MAX\_WORD\_SIZE,fp))){

strcpy(str\_arr[i%STR\_ARRAY\_LEN], str);

i++;

if(i%STR\_ARRAY\_LEN == 0 &&i!=0) {

sort\_strings(str\_arr);

place\_elem(head, str\_arr, doc\_name);

i\_STR\_list(str\_arr);

}

}

if(i%STR\_ARRAY\_LEN != 0 &&i!=0) {

sort\_strings(str\_arr);

place\_elem(head, str\_arr, doc\_name);

i\_STR\_list(str\_arr);

}

return;

}

void search\_rec(char query[MAX\_WORD\_SIZE], node\* head){

static node\* prev\_req = NULL;

if(prev\_req == NULL) prev\_req = head;

char formatted\_q[MAX\_WORD\_SIZE];

int encountered = 0;

for(inti=0; i<MAX\_WORD\_SIZE-1; i++){

if(query[i] == '\0') encountered = 1;

if(!encountered) formatted\_q[i] = query[i];

else formatted\_q[i] = SENTINAL\_CHAR;

}

if(comp\_strings(formatted\_q, prev\_req->word) == IS\_LESSER){

while(comp\_strings(formatted\_q, prev\_req->word) != IS\_EQUAL){

if(prev\_req->prev != NULL)

prev\_req = prev\_req->prev;

else{

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("Your query cannot be found in DB\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

return;

}

}

}

else if(comp\_strings(formatted\_q, prev\_req->word) == IS\_GREATER){

while(comp\_strings(formatted\_q, prev\_req->word) != IS\_EQUAL){

if(prev\_req->next != NULL)

prev\_req = prev\_req->next;

else{

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("Your query cannot be found in DB\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

return;

}

}

}

doc\* temp = prev\_req->docs;

inti=1;

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("%s\n",query);

while(temp != NULL){

printf("%d. %s\n",i++,temp->doc\_name);

temp = temp->next\_doc;

}

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

}

intmain(intargc, char \*\*argv){

node\* head = NULL;

for(inti=1; i<argc; i++){

dict\_add\_from(argv[i], &head);

}

print\_dict(head);

search\_rec("fox", head);

search\_rec("will", head);

search\_rec("me", head);

return 0;

}

**LIMITATIONS :**

We used the concept of hashing upto two levels. This hashing can

be used for more indices as well but the memory taken increases 26 times at

every level. The fn=26^n. So use can use it for at max 4 levels , and also the

code gets complex, where n denotes the level of hashing used.

7)

**PROBLEM STATEMENT :**

Write a simulation program for managing heap memory for allocation and deallocation of user memory requests.

**APPROACH :**

As from we known that heap memory is for dynamically allocated

members . So we try to allocate the memory as per the amount requested by

user. These things can be done in c by using the malloc or realloc type of

functions. And if the user tries to free some sort of heap memory , we free

that memory using the free functions.

The most important part of memory allocation is to check whether the amount

of memory is available to be allocated or noty.. Or whether the amount is

available to be freed or not.

**ALGORITHM :**

1. For allocation we will check out whether the listed amount isavailable or not.

2. If available we allocate that memory using malloc for first time.If we do this step of allocating memory heap memory as perrequest we will use realloc.

3. Now if the user demands to free a specific amount of memory,will first check whether that amount of memory is available inheap or not. If available we free that amount of memory.

4. Now whenever the user requests again, we follow through thesteps of 1B , 2 and 3.

**CODE :**

#include <stdio.h>

#include <stdlib.h>

#define DEF\_USR\_SPACE 200

#define TRUE 1

#define FALSE 0

#define MAX\_FREE\_BLOCKS 20

#define SENTINEL\_INDEX -1

typedefstructblock\_tag{

structblock\_tag\* next;

short intinitialSize;

short intmaxFreeSize;

void\* FreeSpaceRoot;

}block;

typedefstructblock\_info\_tag{

block\* tail;

short intnum\_blocks;

} block\_info;

block\_infoBlocksInUse;

block\_infoBlocksFree;

void init\_usr\_space(block \*b){

char\* block\_point = (char \*)b;

short int\* block\_write = NULL;

block\_point = b->FreeSpaceRoot;

block\_write = (short \*)block\_point;

\*block\_write = (b->initialSize)<<1;

block\_write += 1;

\*block\_write = SENTINEL\_INDEX;

block\_write += 1;

\*block\_write = SENTINEL\_INDEX;

}

block\* create\_block(int size){

if(size < DEF\_USR\_SPACE) size = DEF\_USR\_SPACE;

block\* new\_block = (block \*)malloc(sizeof(block) + size);

new\_block->next = NULL;

new\_block->initialSize = size;

new\_block->maxFreeSize = size;

new\_block->FreeSpaceRoot = (void \*)((char \*)new\_block + sizeof(block));

init\_usr\_space(new\_block);

return new\_block;

}

void add\_block(block\_info\* info\_block, block\* target){

info\_block->num\_blocks++;

if(info\_block->tail == NULL){

info\_block->tail = target;

target->next = target;

}

else{

target->next = info\_block->tail->next;

info\_block->tail->next = target;

info\_block->tail = target;

}

}

void del\_block(block\_info \*info\_block, block\* prev){

info\_block->num\_blocks --;

block\* temp = prev->next;

prev->next = temp->next;

if(info\_block->tail == temp)

info\_block->tail = prev;

free(temp);

}

block\* rem\_block(block\_info\* info\_block, block\* prev){

info\_block->num\_blocks--;

block\* temp = prev->next;

prev->next = temp->next;

if(info\_block->tail == temp)

info\_block->tail = prev;

return temp;

}

block\* fetch\_block(int size){

if(BlocksInUse.num\_blocks != 0){

block\* iterator = BlocksInUse.tail->next;

do{

if(iterator->maxFreeSize>= size)

return iterator;

iterator = iterator->next;

}while(iterator != BlocksInUse.tail->next);

}

block\* block\_to\_use = NULL;

if(BlocksFree.num\_blocks != 0){

block\* iterator = BlocksFree.tail;

intmin\_size = -1;

block\* prev = iterator;

do{

if(iterator->maxFreeSize>= size){

if(min\_size == -1){

min\_size = iterator->maxFreeSize;

block\_to\_use = prev;

}

else if(iterator->maxFreeSize<min\_size){

min\_size = iterator->maxFreeSize;

block\_to\_use = prev;

}

}

prev = iterator;

iterator = iterator->next;

}while(iterator != BlocksFree.tail);

}

if(block\_to\_use == NULL) block\_to\_use = create\_block(size);

else block\_to\_use = rem\_block(&BlocksFree, block\_to\_use);

add\_block(&BlocksInUse, block\_to\_use);

return block\_to\_use;

}

void mem\_alloc(int size){

}

intmain(){

fetch\_block(10);

}

**LIMITATIONS :**

The amount of memory allocation varies from ram to ram… so we need to be careful for memory leak as well as memory overflow.

Memory available for allocation is around 4 or 8 gb.. So users upper limit is limited at quite low level.

**POLYGON INTERSECTION**

#include <stdio.h>

#include<stdbool.h>

#define max(x, y) (((x) > (y)) ? (x) : (y))

#define min(x, y) (((x) < (y)) ? (x) : (y))

#define INF 10000

struct point

{

int x;

int y;

};

typedef struct point Point;

bool onSegment(Point p, Point q, Point r)

{

if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&

q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))

return true;

return false;

}

int orientation(Point p, Point q, Point r)

{

int val = (q.y - p.y) \* (r.x - q.x) -

(q.x - p.x) \* (r.y - q.y);

if (val == 0) return 0;

return (val > 0)? 1: 2;

}

bool doIntersect(Point p1, Point q1, Point p2, Point q2)

{

int d1 = orientation(p1, q1, p2);

int d2 = orientation(p1, q1, q2);

int d3 = orientation(p2, q2, p1);

int d4 = orientation(p2, q2, q1);

if (d1 != o2 && o3 != o4)

return true;

if (d1 == 0 && onSegment(p1, p2, q1)) return true;

if (d2 == 0 && onSegment(p1, q2, q1)) return true;

if (d3 == 0 && onSegment(p2, p1, q2)) return true;

if (d4 == 0 && onSegment(p2, q1, q2)) return true;

return false;

}

bool isInside(Point polygon[], int n, Point p)

{

if (n < 3) return false;

Point extreme = {INF, p.y};

int count = 0, i = 0;

do

{

int next = (i+1)%n;

if (doIntersect(polygon[i], polygon[next], p, extreme))

{

if (orientation(polygon[i], p, polygon[next]) == 0)

return onSegment(polygon[i], p, polygon[next]);

count++;

}

i = next;

} while (i != 0);

// Return true if count is odd, false otherwise

return count&1; // Same as (count%2 == 1)

}

// Driver program to test above functions

int main()

{

Point polygon1[] = {{0, 0}, {10, 0}, {10, 10}, {0, 10}};

int n = sizeof(polygon1)/sizeof(polygon1[0]);

Point p1 = {20, 20};

Point p2 = {50, 50};

Point p3 = {30, 16};

Point p4 = {50, 50};

if(isInside(polygon1, n, p1)==0) // p1 not inside polygon

{

// printf("P1 is not inside polygon.\n");

if(isInside(polygon1, n, p2)==0) // p2 not inside polygon

{

// printf("P2 is not inside polygon.\n");

if(isInside(polygon1, n, p3)==0)

{

// printf("P3 is not inside polygon.\n");

if(isInside(polygon1, n, p4)==0)

{

printf("Polygons don't intersect.\n");

}

else{

// printf("P4 inside polygon.\n");

printf("Polygons intersect.\n");

}

}

else{

// printf("P3 inside polygon.\n");

printf("Polygons intersect.\n");

}

}

else{

// printf("P2 inside polygon.\n");

printf("Polygons intersect.\n");

}

}

else{

printf("Polygons intersect.\n");

}

return 0;

}

**ASSIGNMENT 4**

1)

**PROBLEM STATEMENT :**

Implement a greedy function to implement the coin change problem with coins of denominations 1p, 5p, 10p, 25p and 50p. You need to minimize the number of coins for a change. The input to the function is the amount in paise to be changed and the output is a string containing the denominations and the numbers against each denomination. Call the function from the main program. How would test the correctness of your program?

**APPROACH :**

The idea is simple Greedy Algorithm. Start from largest possible denomination and keep adding denominations while remaining value is greater than 0.

**ALGORITHM :**

1. Initialize Result as empty.
2. Find the largest denomination available that is less than the given amount.
3. Add denominations to result and subtract the corresponding values from the amount.
4. If amount reduces to zero print the result. Else repeat steps 2 and 3 till then.

**CODE :**

// C program to find minimum number of denominations

#include <stdio.h>

#include<string.h>

#define MAX 20

// All denominations available.

int coins[5] = {1,5,10,25,50};

void findMin(int cost)

{

char coinList[5] = {'0','0','0','0','0'};

int i, k = 0;

for (i = 4; i >= 0; i--) {

while (cost >= coins[i]) {

cost -= coins[i] ;

coinList[i]++; // increment count of the particular coin in the list

}

}

for (i = 0; i < 5; i++) {

{

printf("%d -> %c\n",coins[i],coinList[i]);

}

}

return;

}

int main(void)

{

int n;

printf("Enter the amount in paise : ");

scanf("%d",&n);

printf("Following is the list of denomination with their respective counts : \n");

findMin(n);

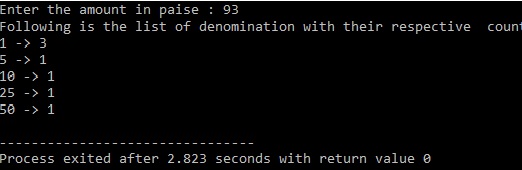
return 0;

}

**LIMITATIONS :**

The above approach may not work for all denominations. For example, it doesn’t work for denominations {9, 6, 5, 1} and V = 11. The above approach would print 9, 1 and 1. But we can use 2 denominations 5 and 6.

**OUTPUT :**



2)

**PROBLEM STATEMENT :**

Write a program to solve the n-queens problem. Choose the data structure you will use to solve the problem. Run the program for 100, 1000 and 10000 queens. Note the time required in each case

**APPROACH :**

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

**ALGORITHM :**

1). Start in the leftmost column.

2) If all queens are placed return true.

3) Try all rows in current column and do the following

a) If queen can safely be placed in this row then mark this cell as 1 and recursively check if placing the queen there leads to a solution.

b) If above condition is satisfied return true.

c) Else unmark this cell and go to step a to try other rows.

If all rows tried for a particular column with no place for a queen return false.

**CODE :**

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

int ld[100] = { 0 };

int rd[100] = { 0 };

int cl[100] = { 0 };

/\* A utility function to print solution \*/

void printSolution(int \*\*board,int r)

{

int i,j;

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

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

printf(" %d ", board[i][j]);

printf("\n");

}

}

bool solveNQUtil(int \*\*board, int col,int r)

{

int i;

if (col >= r)

return true;

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

if ((ld[i - col + r - 1] != 1 &&

rd[i + col] != 1) && cl[i] != 1) {

board[i][col] = 1;

ld[i - col + r - 1] = rd[i + col] = cl[i] = 1;

if (solveNQUtil(board, col + 1,r))

return true;

board[i][col] = 0; // BACKTRACK

ld[i - col + r - 1] = rd[i + col] = cl[i] = 0;

}

}

return false;

}

bool solveNQ(int \*\*board,int r)

{

if (solveNQUtil(board, 0,r) == false) {

printf("Solution does not exist");

return false;

}

printSolution(board,r);

return true;

}

// driver program to test above function

int main()

{

int \*\*board;

int r,i,j;

printf("Enter rows: ");

scanf("%d",&r);

board=(int \*\*)malloc(sizeof(int \*)\*r);

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

board[i]=(int \*)malloc(sizeof(int)\*r);

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

{

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

{

board[i][j] = 0;

}

}

solveNQ(board,r);

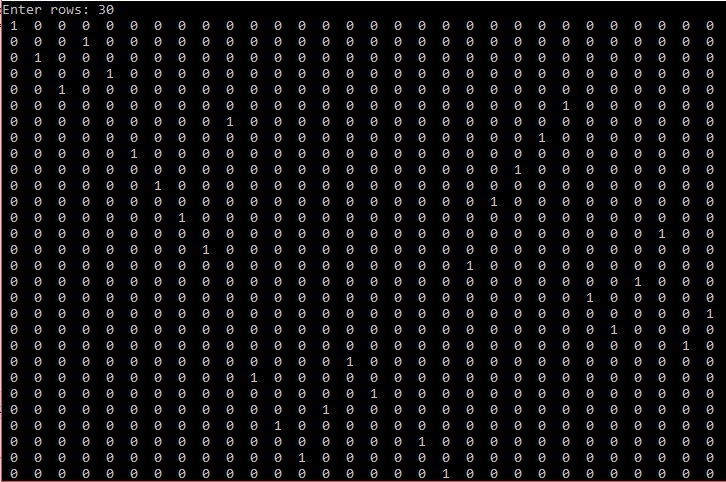
return 0;

}

**LIMITATIONS :**

The above code is exponential in order providing best output for n= 30 at 25 seconds. N > 40 time taken for execution rises exponentially.

**OUTPUT :**



3)

**PROBLEM STATEMENT :**

Write a program to solve the “Rat-in-a-Maze” problem for various dimensions of the maze. How do you propose to present the output for a large maze.

**APPROACH :**

The approach is to start from the top-left cell. For each cell check whether it can be included in path or not. If so try moving along x axis till we come across a barrier. So move one cell back try to move downwards if so continue recursively else backtrack and return 0.

**ALGORITHM :**

If destination is reached

Print the solution matrix

Else

a)Mark the current cell in solution matrix as 1

b) Move forward in horizontal direction and recursively check if it leads to a solution.

c)If not then backtrack and try to move downwards and check recursively for solution.

d)If non of the above two steps yield a solution backtrack and return false.

**CODE :**

#include <stdio.h>

#include<stdlib.h>

int solveMazeUtil(int \*\*maze, int x, int y, int \*\*sol,int r,int c);

void printSolution(int \*\*sol,int x,int y)

{

int i,j;

printf("\nThe required solution : \n");

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

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

printf(" %d ", sol[i][j]);

printf("\n");

}

}

int checkCell(int \*\*maze, int x, int y,int r,int c)

{

if (x >= 0 && x < r && y >= 0 && y < c && maze[x][y] == 1)

return 1;

return 0;

}

int solveMaze(int \*\*maze,int \*\*sol,int r,int c)

{

if (!solveMazeUtil(maze, 0, 0, sol,r,c)) {

printf("Solution doesn't exist");

return 0;

}

return 1;

}

/\* A recursive utility function to solve Maze problem \*/

int solveMazeUtil(int \*\*maze, int x, int y, int \*\*sol,int r,int c)

{

if (x == r - 1 && y == c - 1) {

sol[x][y] = 1;

return 1;

}

if (checkCell(maze, x, y,r,c) == 1) {

sol[x][y] = 1;

if (solveMazeUtil(maze, x + 1, y, sol,r,c) == 1)

return 1;

if (solveMazeUtil(maze, x, y + 1, sol,r,c) == 1)

return 1;

sol[x][y] = 0;

return 0;

}

return 0;

}

int main()

{

int \*\*maze,\*\*sol;

int r,c,i,j;

printf("Enter rows and columns : ");

scanf("%d %d",&r,&c);

maze=(int \*\*)malloc(sizeof(int \*)\*r);

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

maze[i]=(int \*)malloc(sizeof(int)\*c);

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

{

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

{

printf("Enter element at position p[%d][%d] = ",i,j);

scanf("%d",\*(maze+i)+j);

}

}

sol=(int \*\*)malloc(sizeof(int \*)\*r);

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

sol[i]=(int \*)malloc(sizeof(int)\*c);

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

{

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

{

sol[i][j] = 0;

}

}

if(solveMaze(maze,sol,r,c))

printSolution(sol,r,c);

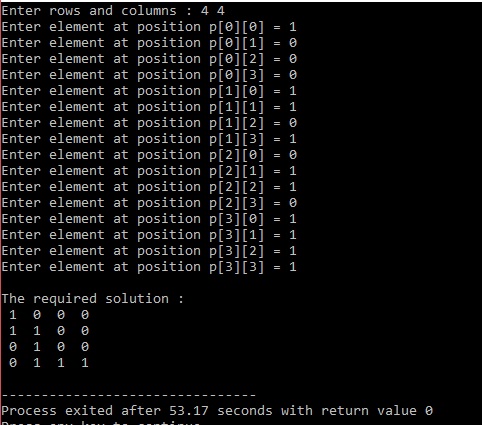
return 0;

}

**LIMITATIONS:**

The code takes exponentially high time for larger values of row and column

**OUTPUT:**



4)

**PROBLEM STATEMENT :**

Implement the following sorting algorithms with their different variants for files of integer and strings:

Bubble sort, Insertion sort, Selection sort, Merge sort, Quick sort and Heap sort.

Tabulate the timing analysis data as discussed in the lab and plot the curves with the timing data to ascertain the time complexity of the algorithms.

**APPROACH :**

Read the data from file(integer or string ) and store them in a suitable array. Now ask from the user which sorting algorithm they want to implement and send the array to that sorting function to do it’s job.

**CODE :**

#include <stdio.h>

#include<stdlib.h>

#define MAX 100000

int time;

void swap(int \*x,int \*y)

{

int temp;

temp = \*x;

\*x = \*y;

\*y = temp;

}

void BubbleSort(int arr[],int n)

{

int i,j,temp;

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

{

for(j = 0;j<n-i-1;j++)

{

if(arr[j]>arr[j+1])

{

swap(&arr[j],&arr[j+1]);

}

}

}

}

void SelectionSort(int arr[], int n)

{

int i,j,temp;

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

{

for(j = i+1;j<n;j++)

{

if(arr[i]>arr[j])

{

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

}

}

}

}

void InsertionSort(int arr[],int n)

{

int i,pos,k;

for(i=1;i<n;i++)

{

pos = i;

k = arr[i];

while(pos > 0 && k < arr[pos-1])

{

arr[pos] = arr[pos-1];

pos--;

}

arr[pos] = k;

}

}

void Merge(int arr[],int l,int r,int mid)

{

int \*t,i=l,j=mid+1,k=0;

t = (int \*)malloc(sizeof(int)\*(r - l + 1));

while(i<=mid && j<=r)

{

if(arr[i]<=arr[j])

t[k++] = arr[i++];

else

t[k++] = arr[j++];

}

for(;i<=mid;i++)

t[k++] = arr[i];

for(;j<=r;j++)

t[k++] = arr[j];

for(i=l,k=0;i<=r;i++,k++)

arr[i] = t[k];

}

void MergeSort(int v[],int l,int r)

{

int mid;

if(l<r)

{

int mid = (l+r)/2;

MergeSort(v,l,mid);

MergeSort(v,mid+1,r);

Merge(v,l,r,mid);

}

}

void QuickSort(int arr[],int l,int r)

{

int p;

if(l<r)

{

p = getPivot(arr,l,r);

QuickSort(arr,l,p-1);

QuickSort(arr,p+1,r);

}

}

int getPivot(int arr[],int l,int r)

{

int p = l,temp;

while(l<r)

{

if(l<r && arr[p] < arr[r])

r--;

if(l<r && arr[p] > arr[r])

{

swap(&arr[p],&arr[r]);

p = r;

l++;

}

if(l<r && arr[p] > arr[l])

l++;

if(l<r && arr[p] < arr[l])

{

swap(&arr[p],&arr[l]);

p = l;

r--;

}

}

return p;

}

void Heapify(int arr[],int n,int pos)

{

int i = pos;

int l = 2\*i + 1;

int r = 2\*i + 2;

int temp;

if(l < n && arr[l] > arr[i])

i = l;

if(r < n && arr[r] > arr[i])

i = r;

if(pos != i)

{

swap(&arr[pos],&arr[i]);

Heapify(arr,n,i);

}

}

void HeapSort(int arr[],int n)

{

int i;

for(i = 0;i< n/2 - 1;i++)

Heapify(arr,n,i);

for(i = n-1;i>=0;i--)

{

swap(&arr[0],&arr[i]);

Heapify(arr,i,0);

}

}

void print(int numbers[],int n)

{

int i;

printf("The sorted array = \n");

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

{

printf("%d --> %d\n", i, numbers[i]);

}

}

int main()

{

FILE \*f;

int numbers[MAX], num, i=0,choice,x;

f=fopen("C:\\Users\\PC\\Documents\\Inteiros.txt", "r");

if(f==NULL)

{

printf("Error\n");

return 0;

}

While(fscanf(f, "%d", &num)>0)

{

numbers[i]=num;

i++;

}

fclose(f);

printf("numbers were read\n");

printf("1.Bubble Sort\n2.Insertion sort\n3.Selection sort\n4.Merge sort\n5.Quick Sort\n6.Heap sort\n7.Exit\n");

do(1)

{

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1: BubbleSort(numbers,i);

print(numbers,i);

break;

case 2: InsertionSort(numbers,i);

print(numbers,i);

break;

case 3: SelectionSort(numbers,i);

print(numbers,i);

break;

case 4: MergeSort(numbers,0,i-1);

print(numbers,i);

break;

case 5: QuickSort(numbers,0,i-1);

print(numbers,i);

break;

case 6: HeapSort(numbers,i-1);

print(numbers,i);

break;

case 7: exit(0);

}

printf("Do you want to continue ?(1/0) ");

scanf("%d",&x);

}while(x == 1);

return 0;

}

**LIMITATIONS :**

Depending upon the input size of array some sorting algorithms mentioned above may not efficiently with respect to time ,like others.

**A screenshot of a cell phone

Description automatically generated**

A screenshot of a cell phone

Description automatically generated

5)

PROBLEM STATEMENT :

Implement the ADT Stack using array and linked list in separate header files. Use the header files to solve the following problems:

Print the elements of a stack .

a) From top to bottom

b) From bottom to top with the following conditions

Stack becoming empty

Stack unchanged

Write a boolean function to return true if two stacks are equal.

Implement the Conversion of Infix expression to Postfix representation.

Implement the evaluation of Postfix expressions.

The above problems are to be solved by taking into account all error conditions in the input

APPROACH : We are writing the required functions in the stack header file along with the declaration of the array itself to store the stack data members. We are presenting an interface to user in the main.c code from where user can call the required function as per requirement.

CODE :

**STACK.H**

#include<stdio.h>

#define MAX 100

int A[MAX];

int a\_top = - 1, b\_top = -1;

int B[MAX];

void push(int x){

if(a\_top == MAX-1){

printf("Overflow\n");

return ;

}

A[++a\_top] = x;

}

void pop(){

if(a\_top == -1){

printf("Underflow\n");

return;

}

printf("popped element = %d",A[a\_top--]);

}

void peep()

{

if(a\_top < 0)

{

printf("NO Element to Display\n");

return;

}

int i;

for(i = a\_top;i>=0;i--)

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

printf("\n");

}

int top(){

return A[a\_top];

}

int IsEmpty(){

if(a\_top == -1) return 1;

else return 0;

}

void bottom\_To\_Top\_a()

{

while(a\_top>=0)

{

B[++b\_top] = A[a\_top--];

}

while(b\_top> = 0)

printf("%d ",B[b\_top--]);

printf("\n");

}

void bottom\_To\_Top\_b()

{

int i;

for(i = 0;i<=a\_top;i++)

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

printf("\n");

}

**MAIN.C**

#include <stdio.h>

#include "stack1.h"

int main() {

int v,ch,i;

printf("Press\n1.Push\n2.Pop\n3.Top to Bottom\n4.Bottom to Top(stack becoming empty)\n5.Bottom to top (stack remains unchanged)\n6.IsEqual\n7.Infix to postfix\n8.Postfix Evaluation\n9.Is\_Empty\n10.Exit\n");

while(1)

{

printf("Enter your choice : ");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("Enter element to push : ");

scanf("%d",&v);

push(v);

break;

case 2: pop();

break;

case 3: peep();

break;

case 4: bottom\_To\_Top\_a();

break;

case 5: bottom\_To\_Top\_b() ;

break;

case 6: IsEqual();

break;

/\*case 7: Infix\_to\_Postfix();

break;

case 8:Postfix\_Evaluation();

break;\*/

case 9:IsEmpty();

break;

case 10: exit(0);

}

}

return;

}

LIMITATIONS :

Our code uses array data structure to store the stack members which requires contiguous memory allocation and may fail to allocate data.

**----------------END-----------------**