# A picture containing text, clipart, vector graphics Description automatically generatedDSA LAB REPORT

# Section-F11-SLOT2

|  |  |  |  |
| --- | --- | --- | --- |
| **EXERCISE NO:** | **All** | **DATE OF EXERCISE:** | **20.12.2021** |
| **ROLL NUMBER:** | **20051685** | **GROUP NO.:** | **11111** |
| **NAME IN CAPITAL :** | **ASHISH KUMAR MISHRA** | | |

Lab Assignment (LA):

1. WAP to find out the smallest and largest element stored in an array of n integers.

#include <stdio.h>

int main()

{

int j[30];

int i,n,large,small;

printf("Enter no. of elements: ");

scanf("%d",&n);

printf("Enter elements in array: ");

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

{

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

}

large=small=j[0];

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

{

if(j[i]>large)

large=j[i];

if(j[i]<small)

small=j[i];

}

printf("Smallest element is %d\n",small);

printf("Largest element is %d\n",large);

return 0;

}

2. WAP to reverse the contents of a dynamic array of n elements.

#include <stdio.h>

#include<stdlib.h>

int main()

{

int \*p,n,i;

printf("Enter no. of elements: ");

scanf("%d",&n);

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

printf("\nEnter %d numbers :\n\n",n);

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

{

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

}

printf("\nArray in Reverse Order:\n\n");

for(i=n-1;i>=0;i--)

{

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

}

return 0;

}

3.WAP to search an element in a dynamic array of n numbers.

#include <stdio.h>

int main()

{

int a[10],n,i,search,f;

printf("\n Enter size of a array :");

scanf("%d",&n);

printf("\nEnter elements of an array :\n",n);

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

{

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

}

printf("\n Enter the search element :");

scanf("%d",&search);

f=0;

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

{

if(a[i] == search)

{

f=1;

break;

}

}

if(f==1)

{

printf("\nFound the search element %d at position %d",search,i+1);

}

else

{

printf("\nNot Found the search element %d",search,i+1);

}

return 0;

}

1. WAP to sort a dynamic array of n numbers.

#include<stdio.h>

#include<stdlib.h>

int main()

{

int \*p,n,i,j,t;

printf("Enter no. of elements to be sorted :");

scanf("%d",&n);

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

printf("\n Enter %d numbers :\n\n",n);

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

{

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

}

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

{

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

{

if(\*(p+i)<\*(p+j))

{

t=\*(p+i);

\*(p+i)=\*(p+j);

\*(p+j)=t;

}

}

}

printf("After sorting in ascending order:\n");

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

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

return 0;

}

5.Given an unsorted dynamic array of size n, WAP to find and display the number of elements between two elements a and b (both inclusive). E.g Input : arr = [1, 2, 2, 7, 5, 4], a=2 and b=5, Output : 4 and the numbers are:2, 2, 5, 4.

#include <conio.h>

#include <stdio.h>

int main()

{

int n,i=0,a,b;

printf("ENTER SIZE OF ARRAY : ");

scanf("%d",&n);

int array[n];

while(i<n)

{

printf("Enter Element No-%d\n",i);

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

i++;

}

printf("Enter Upper limit a: ");

scanf("%d",&a);

printf("Enter Lower limit b: ");

scanf("%d",&b);

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

{

if((array[i]>=a)&&(array[i]<=b))

{

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

}

}

return 0;

}

6.Given a dynamic array, WAP to print the next greater element (NGE) for every element. The next greater element for an element x is the first greater element on the right side of x in array. Elements for which no greater element exist, consider next greater element as -1. E.g. For the input array [2, 5, 3, 9, 7], the next greater elements for each elements are as follows.

|  |  |
| --- | --- |
| Element | NGE |
| 2 | 5 |
| 5 | 9 |
| 3 | 9 |
| 9 | -1 |
| 7 | -1 |

#include<stdio.h>

int main()

{

int temp,n,j=0,nextgreat;

printf("enter the size of array :");

scanf("%d",&n);

int a[n];

printf("enter the values in array :");

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

{

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

}

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

{

nextgreat = 0;

for(int x=y;x<n;x++)

{

if(a[y]<a[x])

{

nextgreat = a[x];

break;

}

}

if(nextgreat == 0)

{

printf("\nnext greater number for %d is :%d",a[y],-1);

}

else

{

printf("\nnext greater number for %d is :%d",a[y],nextgreat);

}

}

return 0;

}

1. Let A be nXn square dynamic matrix. WAP by using appropriate user defined functions for the following:

a) Find the number of nonzero elements in A

b) Find the sum of the elements above the leading diagonal.

c) Display the elements below the minor diagonal.

d) Find the product of the diagonal elements.

#include<stdio.h>

int main()

{

int a[100][100];

int i,j,r1,c1,c=0,p=1,s=0;

printf(" Enter Rows : ");

scanf("%d",&r1);

printf(" Enter Column : ");

scanf("%d",&c1);

for(i=0;i<r1;i++)

{

for(j=0;j<c1;j++)

{

printf(" Enter Elements a%d%d: ",i+1,j+1);

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

if(a[i][j]>0 || a[i][j]<0)

c=c+1;

}

}

for(i=0;i<r1;i++)

{

for(j=0;j<c1;j++)

{

if(i==j)

p=p\*a[i][j];

}

}

for(i=0;i<r1;i++)

{

for(j=0;j<c1;j++)

{

if(j>i)

s=s+a[i][j];

}

}

printf("\n\tMatrix Form : \n");

for(i=0;i<r1;i++)

{

for(j=0;j<c1;j++)

{

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

if(j==c1-1)

printf("\n");

}

}

printf(" Number of Non Zero Elements : %d\n",c);

printf(" Sum of Elements above Leading Diagonal : %d\n",s);

printf(" Product of Diagonal : %d\n",p);

printf(" Elements Below Leading Diagonal : \n");

for(i=0;i<r1;i++)

{

for(j=0;j<c1;j++)

{

if(j<i)

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

}

}

return 0;

}

**8.Given an unsorted dynamic array arr and two numbers x and y, find the minimum distance between x and y in arr. The array might also contain duplicates. You may assume that both x and y are different and present in arr.**

**Input: arr[] = {3, 5, 4, 2, 6, 5, 6, 6, 5, 4, 8, 3}, x = 3, y = 6**

**Output: Minimum distance between 3 and 6 is 4.**

#include <stdio.h>

int main() {

int arr[12] = {3,5,4,2,6,5,6,6,5,4,8,3}, x, y, dis=0, distance=0;

printf("Array: ");

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

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

}

printf("\nEnter x: ");

scanf("%d", &x);

printf("Enter y: ");

scanf("%d", &y);

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

if(arr[i] == x){

distance = 1;

}

if(arr[i]==y){

break;}

if(distance=1){

dis++;

}

}

printf("\nDistance=%d",dis);

return 0;

}

**9.WAP to find out the second smallest and second largest element stored in a dynamic array.**

**#include<stdio.h>**

**int main(){**

**int number[30];**

**int i,j,a,n;**

**printf("Enter size of array\n:");**

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

**printf("Enter elements in array:");**

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

**{**

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

**}**

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

**{**

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

**{**

**if(number[i]<number[j])**

**{**

**a=number[i];**

**number[i]=number[j];**

**number[j]=a;**

**}**

**}**

**}**

**printf("Numbers arranged in descending order are :\n");**

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

**{**

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

**}**

**printf("Second largest no. is :%d\n",number[1]);**

**printf("Second smallest no. is :%d\n",number[n-2]);**

**}**

**10.WAP to arrange the elements of a dynamic array such that all even numbers are followed by all odd numbers.**

**#include<stdio.h>**

**int main()**

**{**

**int n;**

**printf("Enter the size of the array :");**

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

**int x=n-1;**

**int c=0;**

**int arr[100];**

**int a[100];**

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

**{**

**printf("Enter the array element in position %d: ",i);**

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

**}**

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

**{**

**if(arr[i]%2==0)**

**{**

**a[c]=arr[i];**

**c++;**

**}**

**else**

**{**

**a[x]=arr[i];**

**x--;**

**}**

**}**

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

**{**

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

**}**

**}**

**11.Write a program to replace every element in the dynamic array with the next greatest element present in the same array.**

**#include<stdio.h>**

**void nextGreatest(int arr[], int n)**

**{**

**int max=arr[n-1];**

**arr[n-1]=-1;**

**int i;**

**for(i =n-2; i>=0; i--)**

**{**

**int t=arr[i];**

**arr[i]=max;**

**if(max<t)**

**max=t;**

**}**

**}**

**void printArray(int arr[], int n)**

**{**

**int i;**

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

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

**}**

**int main()**

**{**

**int arr[] = {16,17,4,3,5,2};**

**int n=sizeof(arr) /sizeof(arr[0]);**

**nextGreatest(arr, n);**

**printf("Modified array is: \n");**

**printArray(arr, n);**

**return (0);**

**}**

**12.WAP to replace every dynamic array element by multiplication of previous and next of an n element.**

**#include<stdio.h>**

**void newArryPrevNext(int arr1[], int n)**

**{**

**if (n <= 1)**

**return;**

**int pre\_elem = arr1[0];**

**arr1[0] = arr1[0] \* arr1[1];**

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

**{**

**int cur\_elem = arr1[i];**

**arr1[i] = pre\_elem \* arr1[i+1];**

**pre\_elem = cur\_elem;**

**}**

**arr1[n-1] = pre\_elem \* arr1[n-1];**

**}**

**int main()**

**{**

**int arr1[] = {1,2, 3, 4, 5, 6};**

**int n = sizeof(arr1)/sizeof(arr1[0]);**

**int i = 0;**

**printf("The given array is: \n");**

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

**{**

**printf("%d ", arr1[i]);**

**}**

**printf("\n");**

**printf("The new array is: \n");**

**newArryPrevNext(arr1, n);**

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

**printf("%d ", arr1[i]);**

**return 0;**

**}**

**13.WAP to sort rows of a dynamic matrix having m rows and n columns in ascending and columns in descending order.**

**#include <stdio.h>**

**void main()**

**{**

**static int array1[10][10], array2[10][10];**

**int i, j, k, a, m, n;**

**printf("Enter the order of the matrix \n");**

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

**printf("Enter co-efficients of the matrix \n");**

**for (i = 0; i < m; ++i)**

**{**

**for (j = 0; j < n; ++j)**

**{**

**scanf("%d", &array1[i][j]);**

**array2[i][j] = array1[i][j];**

**}**

**}**

**printf("The given matrix is \n");**

**for (i = 0; i < m; ++i)**

**{**

**for (j = 0; j < n; ++j)**

**{**

**printf(" %d", array1[i][j]);**

**}**

**printf("\n");**

**}**

**printf("After arranging rows in ascending order\n");**

**for (i = 0; i < m; ++i)**

**{**

**for (j = 0; j < n; ++j)**

**{**

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

**{**

**if (array1[i][j] > array1[i][k])**

**{**

**a = array1[i][j];**

**array1[i][j] = array1[i][k];**

**array1[i][k] = a;**

**}**

**}**

**}**

**}**

**for (i = 0; i < m; ++i)**

**{**

**for (j = 0; j < n; ++j)**

**{**

**printf(" %d", array1[i][j]);**

**}**

**printf("\n");**

**}**

**printf("After arranging the columns in descending order \n");**

**for (j = 0; j < n; ++j)**

**{**

**for (i = 0; i < m; ++i)**

**{**

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

**{**

**if (array2[i][j] < array2[k][j])**

**{**

**a = array2[i][j];**

**array2[i][j] = array2[k][j];**

**array2[k][j] = a;**

**}**

**}**

**}**

**}**

**for (i = 0; i < m; ++i)**

**{**

**for (j = 0; j < n; ++j)**

**{**

**printf(" %d", array2[i][j]);**

**}**

**printf("\n");**

**}**

**}**

**14.WAP to find out the kth smallest and kth largest element stored in a dynamic array of n integers, where k<n.**

**#include<stdio.h>**

**int main()**

**{**

**int n;**

**printf("Enter the size of the array :");**

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

**int a[100];**

**int max=0;**

**int min=9999999;**

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

**{**

**printf("Enter the array element in position %d: ",i);**

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

**}**

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

**if(a[i]<n){**

**if(min>a[i])**

**{**

**min=a[i];**

**}**

**else if(max<a[i])**

**{**

**max=a[i];**

**}**

**else**

**continue;**

**}**

**}**

**printf("kth largest element is %d\n",max);**

**printf("kth smallest element is %d\n",min);**

**}**

**15.WAP to find the largest number and counts the occurrence of the largest number in a dynamic array of n integers using a single loop.**

**#include<stdio.h>**

**int main()**

**{**

**int n,l=0,occ=0;**

**printf("Enter the size of the array :");**

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

**int ar[n];**

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

**{**

**printf("enter the elements in position %d :",i);**

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

**}**

**l=ar[0];**

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

**{**

**if (l==ar[i])**

**{**

**occ++;**

**}**

**if (l<ar[i])**

**{**

**l=ar[i];**

**occ=0;**

**occ++;**

**}**

**}**

**printf("The largest element is %d and its occurence is: %d",l,occ);**

**return 0;**

**}**

**16.You are given an array of 0s and 1s in random order. Segregate 0s on left side and 1s on right side of the array. Traverse array only once.**

**#include <stdio.h>**

**void segregate0and1(int arr[], int n)**

**{**

**int left=0,right=n-1;**

**while(left<right)**

**{**

**while(arr[left]==0 && left<right)**

**left++;**

**while(arr[right]==1 && left<right )**

**right--;**

**if(left<right)**

**{**

**arr[left]=0;**

**arr[right]=1;**

**left++;**

**right--;**

**}**

**}**

**}**

**int main()**

**{**

**int arr[]={0,1,0,1,1,0};**

**int arr\_n=6, i=0;**

**segregate0and1(arr, arr\_n);**

**printf("segregated array is :");**

**for(i=0; i<6; i++)**

**printf("%d ",arr[i]);**

**getchar();**

**return 0;**

**}**

**17.WAP to swap all the elements in the 1st column with all the corresponding elements in the last column, and 2nd column with the second last column and 3rd with 3rd last etc. of a 2-Dynamic array. Display the matrix.**

**#include<stdio.h>**

**int main()**

**{**

**int r;int c;**

**printf("Enter row value= ");**

**scanf("%d",&r);**

**printf("\n");**

**printf("Enter coulumn value= ");**

**scanf("%d",&c);**

**printf("\n");**

**printf("Enter elements =");**

**int a[r][c];**

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

**{**

**for(int j=0;j<c;j++)**

**{**

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

**}**

**}**

**int t;**

**for(int j=0;j<r;j++){**

**for(int i=0;i<c/2;i++)**

**{**

**t=a[j][i];**

**a[j][i]=a[j][c-1-i];**

**a[j][c-1-i]=t;**

**}**

**}**

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

**{**

**for(int j=0;j<c;j++)**

**{**

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

**}**

**printf("\n");**

**}**

**}**

**18.WAP to arrange the elements of a dynamic array such that all even numbersare followed by all odd numbers using a single loop.**

**#include<stdio.h>**

**int main()**

**{**

**int n;**

**printf("Enter the size of the array:");**

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

**int x=n-1;**

**int c=0;**

**int arr[100];**

**int a[100];**

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

**{**

**printf("Enter the array element in position %d: ",i);**

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

**}**

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

**{**

**if(arr[i]%2==0)**

**{**

**a[c]=arr[i];**

**c++;**

**}**

**else**

**{**

**a[x]=arr[i];**

**x--;**

**}**

**}**

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

**{**

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

**}**

**}**

**19.WAP to perform complex number arithmetic operations including addition, subtraction, multiplication, and division of two complex numbers using different functions.**

#include<stdio.h>

#include<stdlib.h>

struct complex

{

int real\_no, img\_no;

};

int main()

{

int choice, x, y, z;

struct complex a, b, c;

while(1)

{

printf("Enter 1 to add 2 complex numbers:\n");

printf("Enter 2 to subtract 2 complex numbers:\n");

printf("Enter 3 to multiply 2 complex numbers:\n");

printf("Enter 4 to divide 2 complex numbers:\n");

printf("Enter 5 to exit:\n");

printf("Enter your choice:\n");

scanf("%d", &choice);

if(choice ==5)

exit(0);

if(choice>=1 && choice<=4)

{

printf("Enter a and b where a+ib is the first complex number:");

printf("\na = ");

scanf("%d", &a.real\_no);

printf("b = ");

scanf("%d", &a.img\_no);

printf("Enter c and d where c+id is the second complex number:");

printf("\nc = ");

scanf("%d", &b.real\_no);

printf("d = ");

scanf("%d", &b.img\_no);

}

if(choice == 1)

{

c.real\_no = a.real\_no + b.real\_no;

c.img\_no = a.img\_no + b.img\_no;

if(c.img\_no>=0)

printf("Sum of complex no. : %d + %di", c.real\_no, c.img\_no);

else

printf("Sum of complex no. : %d %di", c.real\_no, c.img\_no);

}

else if (choice == 2)

{

c.real\_no = a.real\_no - b.real\_no;

c.img\_no = a.img\_no - b.img\_no;

if(c.img\_no>=0)

printf("Difference of complex no. : %d + %di", c.real\_no, c.img\_no);

else

printf("Difference of complex no. : %d %di", c.real\_no, c.img\_no);

}

else if (choice == 3)

{

c.real\_no = a.real\_no\*b.real\_no - a.img\_no\*b.img\_no;

c.img\_no = a.img\_no\*b.real\_no + a.real\_no\*b.img\_no;

if(c.img\_no>=0)

printf("Multiplication of complex no. : %d + %di", c.real\_no, c.img\_no);

else

printf("Multiplication of complex no. : %d %di", c.real\_no, c.img\_no);

}

else if(choice == 4)

{

if(b.real\_no == 0 && b.img\_no == 0)

printf("Division by 0 + 0i is not allowed.");

else

{

x = a.real\_no\*b.real\_no + a.img\_no\*b.img\_no;

y = a.img\_no\*b.real\_no - a.real\_no\*b.img\_no;

z = b.real\_no\*b.real\_no + b.img\_no\*b.img\_no;

if(x%z == 0 && y%z == 0)

{

if(y/z >= 0)

printf("Division of complex no.: %d + %di", x/z, y/z);

else

printf("Division of complex no.: %d %di", x/z, y/z);

}

else if(x%z == 0 && y%z != 0)

{

if(y/z >= 0)

printf("Division of complex no.: %d + %d/%di", x/z, y, z);

else

printf("Division of complex no.: %d %d/%di", x/z, y, z);

}

else if(x%z != 0 && y%z == 0)

{

if(y/z >= 0)

printf("Division of complex no.: %d/%d + %di", x, z, y/z);

else

printf("Division of complex no.: %d/%d %di", x, z, y/z);

}

else

{

if(y/z >= 0)

printf("Division of complex no.: %d/%d + %d/%di",x, z, y, z);

else

printf("Division of complex no.: %d/%d %d/%di",x, z, y, z);

}

}

}

else

printf("Invalid Choice.");

printf("\n Enter any key to enter choice again...\n");

}

}

**20.Define a structure for representing a point in two-dimensional Cartesian co-ordinate system.**

**a. Write a function to compute the distance between two given points.**

**b. Write a function to compute the middle point of the line segment joining two given points .**

**c. Write a function to compute the area of a triangle, given the co-ordinates of its three vertices.**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

struct Point {

float x, y;

};

double getDistance(struct Point a, struct Point b)

{

double distance;

distance = sqrt((a.x - b.x) \* (a.x - b.x) + (a.y-b.y) \*(a.y-b.y));

return distance;

}

struct Point midpoint(struct Point a, struct Point b)

{

struct Point middle;

middle.x = (a.x + b.x) / 2;

middle.y = (a.y + b.y) / 2;

return middle;

}

double areaoftriangle(struct Point a, struct Point b,struct Point c)

{

double area;

area=0.5 \* (a.x \* (b.y-c.y) + c.x \* (c.y-a.y) + c.x \* (a.y-b.y));

return area;

}

int main()

{

struct Point a, b, c;

printf("Enter coordinate of point a: ");

scanf("%f %f", &a.x, &a.y);

printf("Enter coordinate of point b: ");

scanf("%f %f", &b.x, &b.y);

printf("Enter coordinate of point c: ");

scanf("%f %f", &c.x, &c.y);

printf("Distance between a and b: %lf\n", getDistance(a, b));

struct Point m = midpoint(a, b);

printf("The midpoint between two points a and b: (%f, %f)\n", m.x,m.y);

printf("Area of the triangle formed by three coordinates a,b&c: %lf \n", areaoftriangle(a,b,c));

return 0;

}

**21.WAP to store n employee’s data such as employee name, gender, designation, department,basic pay. Calculate the gross pay of each employee as follows:**

**Gross pay = basic pay + HR + DA**

**HR=25% of basic and DA=75% of basic.**

#include <stdio.h>

#include <stdlib.h>

struct employee{

char name[50];

char gender;

char designation[50];

char department[50];

int basic\_pay;

int gross\_pay;

};

int main()

{

int n,i,j;

struct employees \*ptr;

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

scanf("%d",&n);

struct employee e[n];

ptr=(struct employee\*)malloc(n \* sizeof(struct employee));

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

{

printf("Enter employee %d's name: \n",i+1);

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

printf("Enter employee %d's gender: \n",i+1);

scanf("%s",&e[i].gender);

printf("Enter employee %d's designation: \n",i+1);

scanf("%s",&e[i].designation);

printf("Enter employee %d's department: \n",i+1);

scanf("%s",&e[i].department);

printf("Enter employee %d's basic pay: \n",i+1);

scanf("%d",&e[i].basic\_pay);

}

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

e[i].gross\_pay = e[i].basic\_pay + 0.25\*e[i].basic\_pay + 0.75\*e[i].basic\_pay;

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

{

printf("Employee %d's name: %s\n",i+1,e[i].name);

printf("Employee %d's gender: %c\n",i+1,e[i].gender);

printf("Employee %d's designation: %s\n",i+1,e[i].designation);

printf("Employee %d's department: %s\n",i+1,e[i].department);

printf("Employee %d's basic pay: %d\n",i+1,e[i].basic\_pay);

printf("Employee %d's gross pay is : %d\n",i+1,e[i].gross\_pay);

printf("\n");

}

}

**22.WAP to display the contents of a linked list in reverse order.**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num;

struct node \*next;

};

void create(struct node \*\*);

void reversedisplay(struct node \*);

void release(struct node \*\*);

void display(struct node \*);

int main()

{

struct node \*p = NULL;

struct node\_occur \*head = NULL;

int n;

printf("Enter data into the list\n");

create(&p);

printf("Displaying the nodes in the list:\n");

display(p);

printf("Displaying the list in reverse:\n");

reversedisplay(p);

release(&p);

return 0;

}

void reversedisplay(struct node \*head)

{

if (head != NULL)

{

reversedisplay(head->next);

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

}

}

void create(struct node \*\*head)

{

int c, ch;

struct node \*temp, \*rear;

do

{

printf("Enter number: ");

scanf("%d", &c);

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

temp->num = c;

temp->next = NULL;

if (\*head == NULL)

{

\*head = temp;

}

else

{

rear->next = temp;

}

rear = temp;

printf("Do you wish to continue [YES=1/NO=0]: ");

scanf("%d", &ch);

} while (ch != 0);

printf("\n");

}

void display(struct node \*p)

{

while (p != NULL)

{

printf("%d\t", p->num);

p = p->next;

}

printf("\n");

}

void release(struct node \*\*head)

{

struct node \*temp = \*head;

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

while ((\*head) != NULL)

{

free(temp);

temp = \*head;

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

}

}

**23.WAP to create a double linked list of n nodes and display the linked list by using suitable user defined functions for create and display operations.**

#include <stdio.h>

#include <stdlib.h>

struct node {

int num;

struct node \* preptr;

struct node \* nextptr;

}\*stnode, \*ennode;

void DoublelinkedListcreation(int n);

void displayDoublelinkedList();

int main()

{

int n;

stnode = NULL;

ennode = NULL;

printf("\n\n Doubly Linked List : Create and display a doubly linked list :\n");

printf("-------------------------------------------------------------------\n");

printf(" Enter the number of nodes : ");

scanf("%d", &n);

DoublelinkedListcreation(n);

displayDoublelinkedList();

return 0;

}

void DoublelinkedListcreation(int n)

{

int i, num;

struct node \*fnNode;

if(n >= 1)

{

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

if(stnode != NULL)

{

printf(" Entered data for node 1 : ");

scanf("%d", &num);

stnode->num = num;

stnode->preptr = NULL;

stnode->nextptr = NULL;

ennode = stnode;

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

{

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

if(fnNode != NULL)

{

printf(" Entered data for node %d : ", i);

scanf("%d", &num);

fnNode->num = num;

fnNode->preptr = ennode;

fnNode->nextptr = NULL;

ennode->nextptr = fnNode;

ennode = fnNode;

}

else

{

printf(" Memory can not be allocated.");

break;

}

}

}

else

{

printf(" Memory can not be allocated.");

}

}

}

void displayDoublelinkedList()

{

struct node \* tmp;

int n = 1;

if(stnode == NULL)

{

printf(" No data found in the List yet.");

}

else

{

tmp = stnode;

printf("\n\n Data entered on the list are :\n");

while(tmp != NULL)

{

printf(" node %d : %d\n", n, tmp->num);

n++;

tmp = tmp->nextptr;

}

}

}

**24.WAP to reverse the sequence elements in a double linked list.**

#include <stdio.h>

#include <stdlib.h>

struct node {

int num;

struct node \* preptr;

struct node \* nextptr;

}\*stnode, \*ennode;

void DoublelinkedListcreation(int n);

void displayDoublelinkedListRev();

int main()

{

int n;

stnode = NULL;

ennode = NULL;

printf("\n\n Doubly Linked List : Create and display a doubly linked list in reverse order :\n");

printf("------------------------------------------------------------------------------------\n");

printf(" Enter the number of nodes : ");

scanf("%d", &n);

DoublelinkedListcreation(n);

displayDoublelinkedListRev();

return 0;

}

void DoublelinkedListcreation(int n)

{

int i, num;

struct node \*fnNode;

if(n >= 1)

{

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

if(stnode != NULL)

{

printf(" Entered data for node 1 : ");

scanf("%d", &num);

stnode->num = num;

stnode->preptr = NULL;

stnode->nextptr = NULL;

ennode = stnode;

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

{

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

if(fnNode != NULL)

{

printf(" Entered data for node %d : ", i);

scanf("%d", &num);

fnNode->num = num;

fnNode->preptr = ennode;

fnNode->nextptr = NULL;

ennode->nextptr = fnNode;

ennode = fnNode;

}

else

{

printf(" Memory can not be allocated.");

break;

}

}

}

else

{

printf(" Memory can not be allocated.");

}

}

}

void displayDoublelinkedListRev()

{

struct node \* tmp;

int n = 0;

if(ennode == NULL)

{

printf(" No data found in the List yet.");

}

else

{

tmp = ennode;

printf("\n Data in reverse order are :\n");

while(tmp != NULL)

{

printf(" Data in node %d : %d\n", n+1, tmp->num);

n++;

tmp = tmp->preptr;

}

}

}

**25.Write a menu driven program to perform the following operations in a double linked list by using suitable user defined functions for each case.**

**a) Traverse the list forward,**

**b) Traverse the list backward,**

**c) Check if the list is empty**

**d) Insert a node at the certain position (at beginning/end/any**

**position)**

**e) Delete a node at the certain position (at beginning/end/any**

**position)**

**f) Delete a node for the given key, g) Count the total number of**

**nodes,**

**h) Search for an element in the linked list**

**Verify & validate each function from main method**

#include <stdio.h>

#include <stdlib.h>

struct node

{

struct node \*prev;

int n;

struct node \*next;

}\*h,\*temp,\*temp1,\*temp2,\*temp4;

void insert1();

void insert2();

void insert3();

void traversebeg();

void traverseend(int);

void search();

void delete();

int check(struct node \*h);

int count = 0;

void main()

{

int ch;

h = NULL;

temp = temp1 = NULL;

printf("\n 1 - Insert at beginning");

printf("\n 2 - Insert at end");

printf("\n 3 - Insert at position i");

printf("\n 4 - Delete at i");

printf("\n 5 - Display from beginning");

printf("\n 6 - Display from end");

printf("\n 7 - Search for element");

printf("\n 8 - Count the number of nodes");

printf("\n 9 - Exit");

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

insert1();

break;

case 2:

insert2();

break;

case 3:

insert3();

break;

case 4:

delete();

break;

case 5:

traversebeg();

break;

case 6:

temp2 = h;

if (temp2 == NULL)

printf("\n Error : List empty to display ");

else

{

printf("\n Backward order of linked list is : ");

traverseend(temp2->n);

}

break;

case 7:

search();

break;

case 8:

check(h);

break;

case 9:

exit(0);

default:

printf("\n Wrong choice menu");

}

}

}

void create()

{

int data;

temp =(struct node \*)malloc(1\*sizeof(struct node));

temp->prev = NULL;

temp->next = NULL;

printf("\n Enter value to node : ");

scanf("%d", &data);

temp->n = data;

count++;

}

void insert1()

{

if (h == NULL)

{

create();

h = temp;

temp1 = h;

}

else

{

create();

temp->next = h;

h->prev = temp;

h = temp;

}

}

void insert2()

{

if (h == NULL)

{

create();

h = temp;

temp1 = h;

}

else

{

create();

temp1->next = temp;

temp->prev = temp1;

temp1 = temp;

}

}

void insert3()

{

int pos, i = 2;

printf("\n Enter position to be inserted : ");

scanf("%d", &pos);

temp2 = h;

if ((pos < 1) || (pos >= count + 1))

{

printf("\n Position out of range to insert");

return;

}

if ((h == NULL) && (pos != 1))

{

printf("\n Empty list cannot insert other than 1st position");

return;

}

if ((h == NULL) && (pos == 1))

{

create();

h = temp;

temp1 = h;

return;

}

else

{

while (i < pos)

{

temp2 = temp2->next;

i++;

}

create();

temp->prev = temp2;

temp->next = temp2->next;

temp2->next->prev = temp;

temp2->next = temp;

}

}

void delete()

{

int i = 1, pos;

printf("\n Enter position to be deleted : ");

scanf("%d", &pos);

temp2 = h;

if ((pos < 1) || (pos >= count + 1))

{

printf("\n Error : Position out of range to delete");

return;

}

if (h == NULL)

{

printf("\n Error : Empty list no elements to delete");

return;

}

else

{

while (i < pos)

{

temp2 = temp2->next;

i++;

}

if (i == 1)

{

if (temp2->next == NULL)

{

printf("Node deleted from list");

free(temp2);

temp2 = h = NULL;

return;

}

}

if (temp2->next == NULL)

{

temp2->prev->next = NULL;

free(temp2);

printf("Node deleted from list");

return;

}

temp2->next->prev = temp2->prev;

if (i != 1)

temp2->prev->next = temp2->next;

if (i == 1)

h = temp2->next;

printf("\n Node deleted");

free(temp2);

}

count--;

}

void traversebeg()

{

temp2 = h;

if (temp2 == NULL)

{

printf("List empty to display \n");

return;

}

printf("\n Linked list elements from begining : ");

while (temp2->next != NULL)

{

printf(" %d ", temp2->n);

temp2 = temp2->next;

}

printf(" %d ", temp2->n);

}

void traverseend(int i)

{

if (temp2 != NULL)

{

i = temp2->n;

temp2 = temp2->next;

traverseend(i);

printf(" %d ", i);

}

}

void search()

{

int data, count = 0;

temp2 = h;

if (temp2 == NULL)

{

printf("\n Error : List empty to search for data");

return;

}

printf("\n Enter value to search : ");

scanf("%d", &data);

while (temp2 != NULL)

{

if (temp2->n == data)

{

printf("\n Data found in %d position",count + 1);

return;

}

else

temp2 = temp2->next;

count++;

}

printf("\n Error : %d not found in list", data);

}

int check(struct node \*h)

{

struct node\* temp=h;

int cnt = 0;

while (temp != NULL)

{

temp=temp->next;

cnt++;

}

printf("The total number of nodes is: %d",cnt);

}

**26.WAP to create a single circular double linked list of n nodes and display the linked list by using suitable user defined functions for create and display operations.**

#include <stdio.h>

#include <stdlib.h>

struct node {

int num;

struct node \* nextptr;

}\*stnode;

void CircularlinkedListcreation(int n);

void displayCircularlinkedList();

int main()

{

int n;

stnode = NULL;

printf("\n\n Circular Linked List : Create and display a circular linked list :\n");

printf("-----------------------------------------------------------------------\n");

printf(" Enter the number of nodes : ");

scanf("%d", &n);

CircularlinkedListcreation(n);

displayCircularlinkedList();

return 0;

}

void CircularlinkedListcreation(int n)

{

int i, num;

struct node \*preptr, \*newnode;

if(n >= 1)

{

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

printf(" Entered data for node 1 : ");

scanf("%d", &num);

stnode->num = num;

stnode->nextptr = NULL;

preptr = stnode;

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

{

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

printf(" Entered data for node %d : ", i);

scanf("%d", &num);

newnode->num = num;

newnode->nextptr = NULL;

preptr->nextptr = newnode;

preptr = newnode;

}

preptr->nextptr = stnode;

}

}

void displayCircularlinkedList()

{

struct node \*tmp;

int n = 1;

if(stnode == NULL)

{

printf(" No data found in the List yet.");

}

else

{

tmp = stnode;

printf("\n\n Data entered in the list are :\n");

do {

printf(" Data %d = %d\n", n, tmp->num);

tmp = tmp->nextptr;

n++;

}while(tmp != stnode);

}

}

1. **WAP to remove the duplicates in a sorted double linked list.**

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node\* next;

};

void removeDuplicates(struct Node\* head)

{

struct Node\* current = head;

struct Node\* next\_next;

if (current == NULL)

return;

while (current->next != NULL)

{

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

{

next\_next = current->next->next;

free(current->next);

current->next = next\_next;

}

else

{

current = current->next;

}

}

}

void push(struct Node\*\* head\_ref, int new\_data)

{

struct Node\* new\_node =

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

new\_node->data = new\_data;

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

void printList(struct Node \*node)

{

while (node!=NULL)

{

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

node = node->next;

}

}

int main()

{

struct Node\* head = NULL;

push(&head, 2);

push(&head, 3);

push(&head, 3);

push(&head, 1);

push(&head, 1);

push(&head, 5);

printf("\n Linked list before duplicate removal ");

printList(head);

removeDuplicates(head);

printf("\n Linked list after duplicate removal ");

printList(head);

return 0;

}

**28.Write a menu driven program to perform the following operations**

**in a circular singly linked list by using suitable user defined**

**functions for each case.**

**a) Traverse the list**

**b) Insert a node at the certain position**

**c) Delete a node at the certain position**

**d) Delete a node for the given key**

**e) Count the total number of nodes**

**f) Search for an element in the linked list**

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int data;

struct node \*next;

}Node;

Node \*head = NULL;

Node \*tail = NULL;

int main()

{

int choice;

void create\_list();

void insertAtBeg();

void insertAtEnd();

void insertAtPos();

void deleteAtBeg();

void deleteAtEnd();

void deleteAtPos();

void display\_list();

void display\_list\_pos();

void search();

int countNodes(Node \*head);

printf("\n\*\*\*\*\* Circular Singly Linked List \*\*\*\*\*\n");

printf("\n1.Create List");

printf("\n2.Display List");

printf("\n3.Insert at beginning");

printf("\n4.Insert at end");

printf("\n5.Insert after a node");

printf("\n6.Delete at beginning");

printf("\n7.Delete at end");

printf("\n8.Delete at position");

printf("\n9.Display List from a position");

printf("\n10.Search for an element");

printf("\n11.Count no. of nodes");

printf("\n12.Exit\n");

while(1)

{

printf("\nEnter your choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

create\_list();

break;

case 2:

display\_list();

break;

case 3:

insertAtBeg();

break;

case 4:

insertAtEnd();

break;

case 5:

insertAtPos();

break;

case 6:

deleteAtBeg();

break;

case 7:

deleteAtEnd();

break;

case 8:

deleteAtPos();

break;

case 9:

display\_list\_pos();

break;

case 10:

search();

break;

case 11:

countNodes(head);

break;

case 12:

exit(0);

}

}

return(0);

}

void create\_list()

{

int c;

Node \*temp, \*n;

do{

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

printf("\nEnter the data: ");

scanf("%d", &n->data);

n->next = n;

if(head == NULL)

{

head = tail = temp = n;

}

else

{

temp->next = n;

temp = temp->next;

}

temp->next = head;

tail = temp;

printf("\nEnter your choice to continue press 1 otherwise 0: ");

scanf("%d",&c);

}while(c!=0);

}

void display\_list()

{

Node \*temp;

temp = head;

if (temp == NULL)

printf("\nList is empty");

else

{

while (temp->next != head)

{

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

temp = temp->next;

}

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

}

}

void insertAtBeg()

{

Node \*temp;

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

printf("Enter the element: ");

scanf("%d",&temp->data);

temp->next = tail->next;

tail->next = temp;

head = temp;

}

void insertAtEnd()

{

Node \*temp;

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

printf("\nEnter the element: ");

scanf("%d",&temp->data);

temp->next = tail->next;

tail->next = temp;

tail = temp;

}

void insertAtPos()

{

Node \*temp,\*t;

int key;

t = tail->next;

printf("Enter the key after which new node to be inserted: ");

scanf("%d",&key);

do{

if(t->data == key)

{

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

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

scanf("%d",&temp->data);

temp->next = t->next;

t->next = temp;

if(t == tail)

tail = temp;

return;

}

t = t->next;

}while(t != tail->next);

printf("\n%d not present in the list\n",key);

}

void deleteAtBeg()

{

Node \*temp = tail->next;

head = temp->next;

tail->next = temp->next;

free(temp);

}

void deleteAtEnd()

{

Node \*temp, \*p;

p = tail->next;

while(p->next != tail)

{

p = p->next;

}

p->next = tail->next;

temp = tail;

tail = p;

free(temp);

}

void deleteAtPos()

{

Node \*prev, \*temp;

int i=1,pos;

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

scanf("%d",&pos);

temp = tail->next;

while(i < pos)

{

prev = temp;

temp = temp->next;

++i;

}

prev->next = temp->next;

if(temp == tail)

tail = prev;

free(temp);

}

void display\_list\_pos()

{

int n, i=1;

Node \*temp, \*halt;

temp = head;

printf("Enter the start position: ");

scanf("%d",&n);

if (temp == NULL)

printf("\nList is empty");

else

{

while(i<n)

{

temp = temp->next;

++i;

}

halt = temp;

while (temp->next != halt)

{

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

temp = temp->next;

}

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

}

}

void search()

{

Node \*temp;

int item,i=0,flag=1;

temp = head;

if(temp == NULL)

{

printf("\nEmpty List\n");

}

else

{

printf("\nEnter item which you want to search?\n");

scanf("%d",&item);

if(head ->data == item)

{

printf("item found at location %d",i+1);

flag=0;

}

else

{

while (temp->next != head)

{

if(temp->data == item)

{

printf("item found at location %d ",i+1);

flag=0;

break;

}

else

{

flag=1;

}

i++;

temp = temp -> next;

}

}

if(flag != 0)

{

printf("Item not found\n");

}

}

}

int countNodes(Node\* head)

{

Node\* temp = head;

int result = 0;

if (head != NULL) {

do {

temp = temp->next;

result++;

} while (temp != head);

}

printf("Total number of node is :%d",result);

}

**29.Write a menu driven program to perform the following operations**

**in a circular doubly linked list by using suitable user defined**

**functions for each case.**

**a) Traverse the list**

**b) Insert a node at the certain position**

**c) Delete a node at the certain position**

**d) Delete a node for the given key**

**e) Count the total number of nodes**

**f) Search for an element in the linked list**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int val;

struct node \*next;

struct node \*prev;

};

typedef struct node n;

n\* create\_node(int);

void add\_node();

void insert\_at\_first();

void insert\_at\_end();

void insert\_at\_position();

void delete\_node\_position();

void search();

void display\_from\_beg();

void display\_in\_backward();

int getListLength(n \*first);

n \*new, \*ptr, \*prev;

n \*first = NULL, \*last = NULL;

int number = 0;

void main()

{

int ch;

printf("\n-----Circular Doubly Linked List-----\n");

printf("\n1.Insert at beginning \n2.Insert at end \n3.Insert at position \n4.Delete node at position \n5.Search element \n6.Display list from beginning \n7.Display list from end \n8.Count number of nodes \n9.Exit ");

printf("\n------------------------------\n");

while (1)

{

printf("\n Enter your choice :");

scanf("%d", &ch);

switch (ch)

{

case 1 :

insert\_at\_first();

break;

case 2 :

insert\_at\_end();

break;

case 3 :

insert\_at\_position();

break;

case 4 :

delete\_node\_position();

break;

case 5 :

search();

break;

case 6 :

display\_from\_beg();

break;

case 7 :

display\_in\_backward();

break;

case 8:

getListLength(first);

break;

case 9 :

exit(0);

case 10 :

add\_node();

break;

default:

printf("\ninvalid choice");

}

}

}

n\* create\_node(int info)

{

number++;

new = (n \*)malloc(sizeof(n));

new->val = info;

new->next = NULL;

new->prev = NULL;

return new;

}

void add\_node()

{

int info;

printf("\nEnter the value you would like to add:");

scanf("%d", &info);

new = create\_node(info);

if (first == last && first == NULL)

{

first = last = new;

first->next = last->next = NULL;

first->prev = last->prev = NULL;

}

else

{

last->next = new;

new->prev = last;

last = new;

last->next = first;

first->prev = last;

}

}

void insert\_at\_first()

{

int info;

printf("\nEnter the value to be inserted at first :");

scanf("%d",&info);

new = create\_node(info);

if (first == last && first == NULL)

{

printf("\nInitially it is empty linked list later insertion is done");

first = last = new;

first->next = last->next = NULL;

first->prev = last->prev = NULL;

}

else

{

new->next = first;

first->prev = new;

first = new;

first->prev = last;

last->next = first;

printf("\n The value is inserted at beginning");

}

}

void insert\_at\_end()

{

int info;

printf("\nEnter the value that has to be inserted at last :");

scanf("%d", &info);

new = create\_node(info);

if (first == last && first == NULL)

{

printf("\nInitially the list is empty and now new node is inserted but at first");

first = last = new;

first->next = last->next = NULL;

first->prev = last->prev = NULL;

}

else

{

last->next = new;

new->prev = last;

last = new;

first->prev = last;

last->next = first;

}

}

void insert\_at\_position()

{

int info, pos, len = 0, i;

n \*prevnode;

printf("\n Enter the value that you would like to insert :");

scanf("%d", &info);

printf("\n Enter the position where you have to enter :");

scanf("%d", &pos);

new = create\_node(info);

if (first == last && first == NULL)

{

if (pos == 1)

{

first = last = new;

first->next = last->next = NULL;

first->prev = last->prev = NULL;

}

else

printf("\n Empty linked list you cant insert at that particular position");

}

else

{

if (number < pos)

printf("\n Node cant be inserted as position is exceeding the linked list length");

else

{

for (ptr = first, i = 1;i <= number;i++)

{

prevnode = ptr;

ptr = ptr->next;

if (i == pos-1)

{

prevnode->next = new;

new->prev = prevnode;

new->next = ptr;

ptr->prev = new;

printf("\nInserted at position %d succesfully", pos);

break;

}

}

}

}

}

void delete\_node\_position()

{

int pos, count = 0, i;

n \*temp, \*prevnode;

printf("\n Enter the position which u wanted to delete :");

scanf("%d", &pos);

if (first == last && first == NULL)

printf("\n Empty linked list you cant delete");

else

{

if (number < pos)

printf("\n Node cant be deleted at position as it is exceeding the linked list length");

else

{

for (ptr = first,i = 1;i <= number;i++)

{

prevnode = ptr;

ptr = ptr->next;

if (pos == 1)

{

number--;

last->next = prevnode->next;

ptr->prev = prevnode->prev;

first = ptr;

printf("%d is deleted", prevnode->val);

free(prevnode);

break;

}

else if (i == pos - 1)

{

number--;

prevnode->next = ptr->next;

ptr->next->prev = prevnode;

printf("%d is deleted", ptr->val);

free(ptr);

break;

}

}

}

}

}

void search()

{

int count = 0, key, i, f = 0;

printf("\nEnter the value to be searched:");

scanf("%d", &key);

if (first == last && first == NULL)

printf("\nList is empty no elemnets in list to search");

else

{

for (ptr = first,i = 0;i < number;i++,ptr = ptr->next)

{

count++;

if (ptr->val == key)

{

printf("\n The value is found at position at %d", count);

f = 1;

}

}

if (f == 0)

printf("\n The value is not found in linked list");

}

}

void display\_from\_beg()

{

int i;

if (first == last && first == NULL)

printf("\nList is empty no elemnts to print");

else

{

printf("\n Element in Nodes are : ", number);

for (ptr = first, i = 0;i < number;i++,ptr = ptr->next)

printf("%2d", ptr->val);

}

}

void display\_in\_backward()

{

int i;

if (first == last && first == NULL)

printf("\nList is empty there are no elments: ");

else

{

printf("\n Element in Nodes are : ", number);

for (ptr = last, i = 0;i < number;i++,ptr = ptr->prev)

{

printf("%2d", ptr->val);

}

}

}

int getListLength(n\* first)

{

n\* temp=first;

int count = 0;

if(first != NULL)

do

{

count++;

temp = temp->next;

} while(temp != first);

printf("The total number of nodes : %d",count);

}

**30.How to represent a polynomial using linked list? Write the program to add polynomials.**

#include<stdio.h>

#include<stdlib.h>

typedef struct polynomial

{

int coef;

int expo;

struct polynomial \*next;

}poly;

poly \*create\_poly(poly \*);

poly \*insert\_poly\_node(poly \*, int, int);

poly \*add\_poly(poly \*, poly \*);

void show\_poly(poly \*);

int main() {

int choice;

do{

poly \*poly1, \*poly2, \*poly3;

poly1 = NULL;

poly2 = NULL;

poly3 = NULL;

printf("\nCreate 1st expression\n");

poly1 = create\_poly(poly1);

printf("\nDisplay the 1st expression:\n");

show\_poly(poly1);

printf("\nCreate 2nd expression\n");

poly2 = create\_poly(poly2);

printf("\nDisplay the 2nd expression:\n");

show\_poly(poly2);

poly3 = add\_poly(poly1, poly2);

printf("\nAdditon of two polynomial is:\n");

show\_poly(poly3);

printf("\nDo you want to add two more polynomials?\n");

printf("Enter 1 for Yes, 0 for No: ");

scanf("%d", &choice);

}while(choice);

return 0;

}

poly \*create\_poly(poly \*start)

{

int i,n,ex,co;

printf("How many terms u want to enter : ");

scanf("%d",&n);

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

{

printf("Enter coeficient for term %d : ",i);

scanf("%d",&co);

printf("Enter exponent for term %d : ",i);

scanf("%d",&ex);

start = insert\_poly\_node(start, co, ex);

}

return(start);

}

poly \*insert\_poly\_node(poly \*start, int co, int ex)

{

poly \*ptr, \*tmp;

tmp = (poly \*)malloc(sizeof(poly));

tmp->coef = co;

tmp->expo = ex;

tmp->next = NULL;

if(start == NULL)

start = tmp;

else

{

ptr = start;

while(ptr->next != NULL)

ptr = ptr->next;

ptr->next = tmp;

}

return(start);

}

poly \*add\_poly(poly \*p1, poly \*p2)

{

poly \*p3 = NULL;

if(p1==NULL && p2==NULL)

return p3;

while(p1!=NULL && p2!=NULL)

{

if(p1->expo > p2->expo)

{

p3 = insert\_poly\_node(p3, p1->coef, p1->expo);

p1 = p1->next;

}

else if(p2->expo > p1->expo)

{

p3 = insert\_poly\_node(p3, p2->coef, p2->expo);

p2 = p2->next;

}

else

{

p3 = insert\_poly\_node(p3, p1->coef + p2->coef, p1->expo);

p1 = p1->next;

p2 = p2->next;

}

}

while(p1 != NULL)

{

p3 = insert\_poly\_node(p3, p1->coef, p1->expo);

p1 = p1->next;

}

while(p2 != NULL)

{

p3 = insert\_poly\_node(p3, p2->coef, p2->expo);

p2 = p2->next;

}

return p3;

}

void show\_poly(poly \*ptr)

{

if(ptr==NULL)

{

printf("Empty\n");

return;

}

while(ptr->next != NULL)

{

printf("(%dx^%d) + ", ptr->coef,ptr->expo);

ptr=ptr->next;

}

printf("(%dx^%d)", ptr->coef,ptr->expo);

printf("\n");

}

**31.How to represent a polynomial using linked list? Write the the program to add three polynomials.**

**Hint: polyad(polyad(poly1, poly2),poly3);**

#include <stdio.h>

#include <stdlib.h>

typedef struct polynomial

{

int coef;

int expo;

struct polynomial \*next;

} poly;

poly \*create\_poly(poly \*);

poly \*insert\_poly\_node(poly \*, int, int);

poly \*add\_poly(poly \*, poly \*);

void show\_poly(poly \*);

int main()

{

int choice;

do

{

poly \*poly1, \*poly2, \*poly3, \*poly4;

poly1 = NULL;

poly2 = NULL;

poly3 = NULL;

printf("\nCreate 1st expression\n");

poly1 = create\_poly(poly1);

printf("\nDisplay the 1st expression:\n");

show\_poly(poly1);

printf("\nCreate 2nd expression\n");

poly2 = create\_poly(poly2);

printf("\nDisplay the 2nd expression:\n");

show\_poly(poly2);

printf("\nCreate 3rd expression\n");

poly3 = create\_poly(poly3);

printf("\nDisplay the 3rd expression:\n");

show\_poly(poly3);

poly4 = add\_poly((add\_poly(poly1, poly2)), poly3);

printf("\nAdditon of three polynomial is:\n");

show\_poly(poly4);

printf("\nDo you want to add three more polynomials?\n");

printf("Enter 1 for Yes, 0 for No: ");

scanf("%d", &choice);

} while (choice);

return 0;

}

poly \*create\_poly(poly \*start)

{

int i, n, ex, co;

printf("How many terms u want to enter : ");

scanf("%d", &n);

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

{

printf("Enter coeficient for term %d : ", i);

scanf("%d", &co);

printf("Enter exponent for term %d : ", i);

scanf("%d", &ex);

start = insert\_poly\_node(start, co, ex);

}

return (start);

}

poly \*insert\_poly\_node(poly \*start, int co, int ex)

{

poly \*ptr, \*tmp;

tmp = (poly \*)malloc(sizeof(poly));

tmp->coef = co;

tmp->expo = ex;

tmp->next = NULL;

if (start == NULL)

start = tmp;

else

{

ptr = start;

while (ptr->next != NULL)

ptr = ptr->next;

ptr->next = tmp;

}

return (start);

}

poly \*add\_poly(poly \*p1, poly \*p2)

{

poly \*p3 = NULL;

if (p1 == NULL && p2 == NULL)

return p3;

while (p1 != NULL && p2 != NULL)

{

if (p1->expo > p2->expo)

{

p3 = insert\_poly\_node(p3, p1->coef, p1->expo);

p1 = p1->next;

}

else if (p2->expo > p1->expo)

{

p3 = insert\_poly\_node(p3, p2->coef, p2->expo);

p2 = p2->next;

}

else

{

p3 = insert\_poly\_node(p3, p1->coef + p2->coef, p1->expo);

p1 = p1->next;

p2 = p2->next;

}

}

while (p1 != NULL)

{

p3 = insert\_poly\_node(p3, p1->coef, p1->expo);

p1 = p1->next;

}

while (p2 != NULL)

{

p3 = insert\_poly\_node(p3, p2->coef, p2->expo);

p2 = p2->next;

}

return p3;

}

void show\_poly(poly \*ptr)

{

if (ptr == NULL)

{

printf("Empty\n");

return;

}

while (ptr->next != NULL)

{

printf("(%dx^%d) + ", ptr->coef, ptr->expo);

ptr = ptr->next;

}

printf("(%dx^%d)", ptr->coef, ptr->expo);

printf("\n");

}

**32.Design a suitable data structure to efficiently represent a sparse matrix ? Write the program to add the original sparse matrix with the transpose of the same matrix.**

#include<stdio.h>

#include<stdlib.h>

typedef struct tr{

int row,col,val;

}tr;

int main()

{

printf("Give row and column of the matrix: ");

int i,j;

int a,b;

scanf("%d %d",&a,&b);

if(a != b)

{

printf("Not possible as not a square matrix!!!!!");

return 0;

}

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

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

{

sparse[i] = (int\*)malloc(a\*sizeof(int));

}

int n\_z = 0;

printf("Give values:\n");

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

{

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

{

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

if(sparse[i][j])

{

n\_z ++;

}

}

}

tr\* tr1;

tr\* tr2;

tr1 = (tr\*)malloc((n\_z + 1)\*sizeof(tr));

tr2 = (tr\*)malloc((n\_z + 1)\*sizeof(tr));

tr1[0].row = tr1[0].col = a;

tr1[0].val = n\_z;

int k = 1;

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

{

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

{

if(sparse[i][j]!=0)

{

tr1[k].row = i;tr1[k].col = j;tr1[k].val = sparse[i][j];

k++;

}

}

}

free(sparse);

for(i=0;i<n\_z+1;++i)

{

tr2[i].row = tr1[i].col;

tr2[i].col = tr1[i].row;

tr2[i].val = tr1[i].val;

}

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

{

for(j=i+1;j<n\_z+1;++j)

{

if(tr2[i].row > tr2[j].row)

{

tr temp = tr2[i];

tr2[i] = tr2[j];

tr2[j] = temp;

}

if(tr2[i].row == tr2[j].row)

{

if(tr2[i].col > tr2[j].col)

{

tr temp = tr2[i];

tr2[i] = tr2[j];

tr2[j] = temp;

}

}

}

}

printf("\nSparse matrix:\n");

for(i=0;i<n\_z+1;++i)

{

printf("%d %d %d\n",tr1[i].row,tr1[i].col,tr1[i].val);

}

printf("\nTranspose of sparse matrix:\n");

printf("\n\n");

for(i=0;i<n\_z+1;++i)

{

printf("%d %d %d\n",tr2[i].row,tr2[i].col,tr2[i].val);

}

tr sum[(2\*n\_z) + 1];

sum[0].row = tr1[0].row;

sum[0].col = tr1[0].col;

sum[0].val = tr1[0].val;

i = 1,j = 1;

k = 1;

printf("\nSum is:\n");

while(i < n\_z+1 || j < n\_z + 1)

{

if(tr1[i].row < tr2[j].row)

{

sum[k] = tr1[i];

k++;

i++;

}

if(tr1[i].row > tr2[j].row)

{

sum[k] = tr2[j];

k++;

j++;

}

if(tr1[i].row == tr2[j].row)

{

if(tr1[i].col < tr2[j].col)

{

sum[k] = tr1[i];

k++;

i++;

}

else if(tr1[i].col > tr2[j].col)

{

sum[k] = tr2[j];

k++;

j++;

}

else

{

sum[k] = tr2[j];

sum[k].val = tr1[i].val + tr2[j].val;

i++;

k++;

j++;

}

}

}

while(i < n\_z+1)

{

sum[k] = tr1[i];

k++;

i++;

}

while(j < n\_z+1)

{

sum[k] = tr2[j];

k++;

j++;

}

printf("\n\n");

printf("%d %d %d\n",k-1,k-1,k-1);

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

{

printf("%d %d %d\n",sum[i].row,sum[i].col,sum[i].val);

}

}

**33.WAP Write a menu driven program to perform the following operations of a stack using array by using suitable user defined functions for each case.**

1. **Check if the stack is empty**
2. **b) Display the contents of stack**

**c) Push d) Pop**

#include<stdio.h>

#include<stdlib.h>

#define MAX 10

int top=-1,stack[MAX];

void push();

void pop();

void display();

void main()

{

int ch;

while(1)

{

printf("\n\*\* Stack Menu \*\*");

printf("\n\n1.Push\n2.Pop\n3.Display\n4.Exit");

printf("\n\nEnter your choice(1-4):");

scanf("%d",&ch);

switch(ch)

{

case 1: push();

break;

case 2: pop();

break;

case 3: display();

break;

case 4: exit(0);

default: printf("\nWrong Choice!!");

}

}

}

void push()

{

int val;

if(top==MAX-1)

{

printf("\nStack is full!!");

}

else

{

printf("\nEnter element to push:");

scanf("%d",&val);

top=top+1;

stack[top]=val;

}

}

void pop()

{

if(top==-1)

{

printf("\nStack is empty!!");

}

else

{

printf("\nDeleted element is %d",stack[top]);

top=top-1;

}

}

void display()

{

int i;

if(top==-1)

{

printf("\nStack is empty!!");

}

else

{

printf("\nStack is...\n");

for(i=top;i>=0;--i)

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

}

}

**34.WAP Write a menu driven program to perform the following**

**operations of a stack using linked list by using suitable user defined**

**functions for each case.**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*ptr;

}\*top,\*top1,\*temp;

int topelement();

void push(int data);

void pop();

void empty();

void display();

void destroy();

void stack\_count();

void create();

int count = 0;

void main()

{

int no, ch, e;

printf("\n 1 - Push");

printf("\n 2 - Pop");

printf("\n 3 - Top");

printf("\n 4 - Empty");

printf("\n 5 - Exit");

printf("\n 6 - Display");

printf("\n 7 - Stack Count");

printf("\n 8 - Destroy stack");

create();

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter data : ");

scanf("%d", &no);

push(no);

break;

case 2:

pop();

break;

case 3:

if (top == NULL)

printf("No elements in stack");

else

{

e = topelement();

printf("\n Top element : %d", e);

}

break;

case 4:

empty();

break;

case 5:

exit(0);

case 6:

display();

break;

case 7:

stack\_count();

break;

case 8:

destroy();

break;

default :

printf(" Wrong choice, Please enter correct choice ");

break;

}

}

}

void create()

{

top = NULL;

}

void stack\_count()

{

printf("\n No. of elements in stack : %d", count);

}

void push(int data)

{

if (top == NULL)

{

top =(struct node \*)malloc(1\*sizeof(struct node));

top->ptr = NULL;

top->info = data;

}

else

{

temp =(struct node \*)malloc(1\*sizeof(struct node));

temp->ptr = top;

temp->info = data;

top = temp;

}

count++;

}

void display()

{

top1 = top;

if (top1 == NULL)

{

printf("Stack is empty");

return;

}

while (top1 != NULL)

{

printf("%d ", top1->info);

top1 = top1->ptr;

}

}

void pop()

{

top1 = top;

if (top1 == NULL)

{

printf("\n Error : Trying to pop from empty stack");

return;

}

else

top1 = top1->ptr;

printf("\n Popped value : %d", top->info);

free(top);

top = top1;

count--;

}

int topelement()

{

return(top->info);

}

void empty()

{

if (top == NULL)

printf("\n Stack is empty");

else

printf("\n Stack is not empty with %d elements", count);

}

void destroy()

{

top1 = top;

while (top1 != NULL)

{

top1 = top->ptr;

free(top);

top = top1;

top1 = top1->ptr;

}

free(top1);

top = NULL;

printf("\n All stack elements destroyed");

count = 0;

}

**35.WAP to convert an infix expression into its equivalent postfix**

**notation.**

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

#define SIZE 100

char stack[SIZE];

int top = -1;

void push(char item)

{

if(top >= SIZE-1)

{

printf("\nStack Overflow.");

}

else

{

top = top+1;

stack[top] = item;

}

}

char pop()

{

char item ;

if(top <0)

{

printf("stack under flow: invalid infix expression");

getchar();

exit(1);

}

else

{

item = stack[top];

top = top-1;

return(item);

}

}

int is\_operator(char symbol)

{

if(symbol == '^' || symbol == '\*' || symbol == '/' || symbol == '+' || symbol =='-')

{

return 1;

}

else

{

return 0;

}

}

int precedence(char symbol)

{

if(symbol == '^')

{

return(3);

}

else if(symbol == '\*' || symbol == '/')

{

return(2);

}

else if(symbol == '+' || symbol == '-')

{

return(1);

}

else

{

return(0);

}

}

void InfixToPostfix(char infix\_exp[], char postfix\_exp[])

{

int i, j;

char item;

char x;

push('(');

strcat(infix\_exp,")");

i=0;

j=0;

item=infix\_exp[i];

while(item != '\0')

{

if(item == '(')

{

push(item);

}

else if( isdigit(item) || isalpha(item))

{

postfix\_exp[j] = item;

j++;

}

else if(is\_operator(item) == 1)

{

x=pop();

while(is\_operator(x) == 1 && precedence(x)>= precedence(item))

{

postfix\_exp[j] = x;

j++;

x = pop();

}

push(x);

push(item);

}

else if(item == ')')

{

x = pop();

while(x != '(')

{

postfix\_exp[j] = x;

j++;

x = pop();

}

}

else

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

i++;

item = infix\_exp[i];

}

if(top>0)

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

if(top>0)

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

postfix\_exp[j] = '\0';

}

int main()

{

char infix[SIZE], postfix[SIZE];

printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only.\n");

printf("\nEnter Infix expression : ");

gets(infix);

InfixToPostfix(infix,postfix);

printf("Postfix Expression: ");

puts(postfix);

return 0;

}

**36.WAP to convert an infix expression into its equivalent prefix**

**notation.**

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define MAX 20

void push(int);

char pop();

void infix\_to\_prefix();

int precedence(char);

char stack[20], infix[20], prefix[20];

int top = -1;

int main()

{

printf("\nINPUT THE INFIX EXPRESSION : ");

scanf("%s", infix);

infix\_to\_prefix();

return 0;

}

void push(int pos)

{

if (top == MAX - 1)

{

printf("\nSTACK OVERFLOW\n");

}

else

{

top++;

stack[top] = infix[pos];

}

}

char pop()

{

char ch;

if (top < 0)

{

printf("\nSTACK UNDERFLOW\n");

exit(0);

}

else

{

ch = stack[top];

stack[top] = '\0';

top--;

return (ch);

}

return 0;

}

void infix\_to\_prefix()

{

int i = 0, j = 0;

strrev(infix);

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

{

if (infix[i] >= 'a' && infix[i] <= 'z')

{

prefix[j] = infix[i];

j++;

i++;

}

else if (infix[i] == ')' || infix[i] == '}' || infix[i] == ']')

{

push(i);

i++;

}

else if (infix[i] == '(' || infix[i] == '{' || infix[i] == '[')

{

if (infix[i] == '(')

{

while (stack[top] != ')')

{

prefix[j] = pop();

j++;

}

pop();

i++;

}

else if (infix[i] == '[')

{

while (stack[top] != ']')

{

prefix[j] = pop();

j++;

}

pop();

i++;

}

else if (infix[i] == '{')

{

while (stack[top] != '}')

{

prefix[j] = pop();

j++;

}

pop();

i++;

}

}

else

{

if (top == -1)

{

push(i);

i++;

}

else if (precedence(infix[i]) < precedence(stack[top]))

{

prefix[j] = pop();

j++;

while (precedence(stack[top]) > precedence(infix[i]))

{

prefix[j] = pop();

j++;

if (top < 0)

{

break;

}

}

push(i);

i++;

}

else if (precedence(infix[i]) >= precedence(stack[top]))

{

push(i);

i++;

}

}

}

while (top != -1)

{

prefix[j] = pop();

j++;

}

strrev(prefix);

prefix[j] = '\0';

printf("EQUIVALENT PREFIX NOTATION : %s ", prefix);

}

int precedence(char alpha)

{

if (alpha == '+' || alpha == '-')

{

return (1);

}

if (alpha == '\*' || alpha == '/')

{

return (2);

}

return 0;

}

**37.Two brackets are considered to be a matched pair if the an opening**

**bracket (i.e., (, [, or { ) occurs to the left of a closing bracket (i.e., ),**

**], or }) of the exact same type. There are three types of matched**

**pairs of brackets: [], {}, and (). A matching pair of brackets is not**

**balanced if the set of brackets it encloses are not matched. WAP to**

**determine whether the input sequence of brackets is balanced or not.**

**If a string is balanced, it print YES on a new line; otherwise, print**

**NO on a new line.**

**Example: Input: {[()]} and Output: YES**

**Input: {[(])} and Output: NO**

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

int same(char a,char b)

{

if(a=='['&& b==']')

return 1;

if(a=='{'&& b=='}')

return 1;

if(a=='('&& b==')')

return 1;

return 0;

}

int check(char \*a)

{

char stack[1001],top=-1;

int j;

for(j=0;j<strlen(a);j++)

{

if(a[j]=='['||a[j]=='{'||a[j]=='(')

stack[++top]=a[j];

if(a[j]==']'||a[j]=='}'||a[j]==')')

{

if(top==-1)

{

return 0;

}

else

{

if(!same(stack[top--],a[j]))

{

return 0;

}

}

}

}

if(top!=-1)

{

return 0;

}

return 1;

}

int main() {

char a[1001];

int n, i, valid;

scanf("%d",&n);

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

{

scanf("%s",a);

valid = check(a);

if(valid==1)

printf("YES\n");

else

printf("NO\n");

}

return 0;

}

**38.Write a menu driven program to implement queue operations**

**such as Enqueue, Dequeue, Peek, Display of elements, isEmpty,**

**isFull using static array.**

#include<stdio.h>

#include<stdlib.h>

#define MAX 10

int queue\_arr[MAX];

int rear=-1;

int front=-1;

void enqueue(int item);

int dequeue();

int peek();

void display();

int isFull();

int isEmpty();

int main()

{

int choice,item;

while(1)

{

printf("\n1.Enqueue\n");

printf("2.Dequeue\n");

printf("3.Display element at the front\n");

printf("4.Display all elements of the queue\n");

printf("5.Quit\n");

printf("\nEnter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("\nEnqueue the element for adding in queue : ");

scanf("%d",&item);

enqueue(item);

break;

case 2:

item=dequeue();

printf("\nDequeued element is %d\n",item);

break;

case 3:

printf("\nElement at the front is %d\n",peek());

break;

case 4:

display();

break;

case 5:

exit(1);

default:

printf("\nWrong choice\n");

}

}

return 0;

}

void enqueue(int item)

{

if( isFull() )

{

printf("\nQueue Overflow\n");

return;

}

if( front == -1 )

front=0;

rear=rear+1;

queue\_arr[rear]=item ;

}

int dequeue()

{

int item;

if( isEmpty() )

{

printf("\nQueue Underflow\n");

exit(1);

}

item=queue\_arr[front];

front=front+1;

return item;

}

int peek()

{

if( isEmpty() )

{

printf("\nQueue Underflow\n");

exit(1);

}

return queue\_arr[front];

}

int isEmpty()

{

if( front==-1 || front==rear+1 )

return 1;

else

return 0;

}

int isFull()

{

if( rear==MAX-1 )

return 1;

else

return 0;

}

void display()

{

int i;

if ( isEmpty() )

{

printf("\nQueue is empty\n");

return;

}

printf("\nQueue is :\n\n");

for(i=front;i<=rear;i++)

printf("%d ",queue\_arr[i]);

printf("\n\n");

}

**39.Write a menu driven program to implement queue operations**

**such as Enqueue, Dequeue, Peek, Display of elements, isEmpty, isFull using dynamic array.**

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

struct Queue {

int front, rear, size;

unsigned capacity;

int\* array;

};

struct Queue\* createQueue(unsigned capacity)

{

struct Queue\* queue = (struct Queue\*)malloc(

sizeof(struct Queue));

queue->capacity = capacity;

queue->front = queue->size = 0;

queue->rear = capacity - 1;

queue->array = (int\*)malloc(

queue->capacity \* sizeof(int));

return queue;

}

int isFull(struct Queue\* queue)

{

return (queue->size == queue->capacity);

}

int isEmpty(struct Queue\* queue)

{

return (queue->size == 0);

}

void enqueue(struct Queue\* queue, int item)

{

if (isFull(queue))

return;

queue->rear = (queue->rear + 1)

% queue->capacity;

queue->array[queue->rear] = item;

queue->size = queue->size + 1;

printf("%d enqueued to queue\n", item);

}

int dequeue(struct Queue\* queue)

{

if (isEmpty(queue))

return INT\_MIN;

int item = queue->array[queue->front];

queue->front = (queue->front + 1)

% queue->capacity;

queue->size = queue->size - 1;

return item;

}

int peek(struct Queue\* queue)

{

if (isEmpty(queue))

return ;

return queue->array[queue->front];

}

int rear(struct Queue\* queue)

{

if (isEmpty(queue))

return INT\_MIN;

return queue->array[queue->rear];

}

int main()

{

struct Queue\* queue = createQueue(1000);

enqueue(queue, 10);

enqueue(queue, 20);

enqueue(queue, 30);

enqueue(queue, 40);

printf("\nFront item is %d\n", peek(queue));

printf("\n%d dequeued from queue\n\n",

dequeue(queue));

printf("Rear item is %d\n", rear(queue));

return 0;

}

**40.Write a menu driven program to implement queue operations**

**such as Enqueue, Dequeue, Peek, Display of elements, isEmpty**

**using linked list.**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*ptr;

}\*front,\*rear,\*temp,\*front1;

int peek();

void enq(int data);

void deq();

void empty();

void display();

void create();

void queuesize();

int count = 0;

void main()

{

int no, ch, e;

printf("\n 1 - Enque");

printf("\n 2 - Deque");

printf("\n 3 - Front element");

printf("\n 4 - Empty");

printf("\n 5 - Exit");

printf("\n 6 - Display");

printf("\n 7 - Queue size");

create();

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter data : ");

scanf("%d", &no);

enq(no);

break;

case 2:

deq();

break;

case 3:

e = peek();

if (e != 0)

printf("Front element : %d", e);

else

printf("\n No front element in Queue as queue is empty");

break;

case 4:

empty();

break;

case 5:

exit(0);

case 6:

display();

break;

case 7:

queuesize();

break;

default:

printf("Wrong choice, Please enter correct choice ");

break;

}

}

}

void create()

{

front = rear = NULL;

}

void queuesize()

{

printf("\n Queue size : %d", count);

}

void enq(int data)

{

if (rear == NULL)

{

rear = (struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = NULL;

rear->info = data;

front = rear;

}

else

{

temp=(struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = temp;

temp->info = data;

temp->ptr = NULL;

rear = temp;

}

count++;

}

void display()

{

front1 = front;

if ((front1 == NULL) && (rear == NULL))

{

printf("Queue is empty");

return;

}

while (front1 != rear)

{

printf("%d ", front1->info);

front1 = front1->ptr;

}

if (front1 == rear)

printf("%d", front1->info);

}

void deq()

{

front1 = front;

if (front1 == NULL)

{

printf("\n Error: Trying to display elements from empty queue");

return;

}

else

if (front1->ptr != NULL)

{

front1 = front1->ptr;

printf("\n Dequed value : %d", front->info);

free(front);

front = front1;

}

else

{

printf("\n Dequed value : %d", front->info);

free(front);

front = NULL;

rear = NULL;

}

count--;

}

int peek()

{

if ((front != NULL) && (rear != NULL))

return(front->info);

else

return 0;

}

void empty()

{

if ((front == NULL) && (rear == NULL))

printf("\n Queue empty");

else

printf("Queue not empty");

}

**41.Write a menu driven program to implement circular queue operations such as Enqueue, Dequeue, Peek, Display of elements,**

**isEmpty, isFull using static array.**

#include<stdio.h>

#include<stdlib.h>

#define MAX 10

int cqueue\_arr[MAX];

int front=-1;

int rear=-1;

void display( );

void enqueue(int item);

int dequeue();

int peek();

int isEmpty();

int isFull();

int main()

{

int choice,item;

while(1)

{

printf("\n1.Enqueue\n");

printf("2.Dequeue\n");

printf("3.Peek\n");

printf("4.Display\n");

printf("5.Quit\n");

printf("\nEnter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1 :

printf("\nInput the element for enqueue : ");

scanf("%d",&item);

enqueue(item);

break;

case 2 :

printf("\nElement dequeued is : %d\n",dequeue());

break;

case 3:

printf("\nElement at the front is : %d\n",peek());

break;

case 4:

display();

break;

case 5:

exit(1);

default:

printf("\nWrong choice\n");

}

}

return 0;

}

void enqueue(int item)

{

if( isFull() )

{

printf("\nQueue Overflow\n");

return;

}

if(front == -1 )

front=0;

if(rear==MAX-1)

rear=0;

else

rear=rear+1;

cqueue\_arr[rear]=item ;

}

int dequeue()

{

int item;

if( isEmpty() )

{

printf("\nQueue Underflow\n");

exit(1);

}

item=cqueue\_arr[front];

if(front==rear)

{

front=-1;

rear=-1;

}

else if(front==MAX-1)

front=0;

else

front=front+1;

return item;

}

int isEmpty()

{

if(front==-1)

return 1;

else

return 0;

}

int isFull()

{

if((front==0 && rear==MAX-1) || (front==rear+1))

return 1;

else

return 0;

}

int peek()

{

if( isEmpty() )

{

printf("\nQueue Underflow\n");

exit(1);

}

return cqueue\_arr[front];

}

void display()

{

int i;

if(isEmpty())

{

printf("\nQueue is empty\n");

return;

}

printf("\nQueue elements :\n");

i=front;

if( front<=rear )

{

while(i<=rear)

printf("%d ",cqueue\_arr[i++]);

}

else

{

while(i<=MAX-1)

printf("%d ",cqueue\_arr[i++]);

i=0;

while(i<=rear)

printf("%d ",cqueue\_arr[i++]);

}

printf("\n");

}

**42.Write a menu driven program to implement circular queue**

**operations such as Enqueue, Dequeue, Peek, Display of elements,**

**isEmpty, isFull using dynamic array.**

#include<stdio.h>

#include<stdlib.h>

typedef struct {

int front;

int rear;

int capacity;

int \*arr;

}cqueue;

cqueue \*Q;

void initQueue()

{

Q = (cqueue\*)malloc(sizeof(cqueue));

printf("Enter the size of the queue: ");

scanf("%d", &Q->capacity);

Q->front = Q->rear = -1;

Q->arr = (int \*)malloc(sizeof(int)\*Q->capacity);

}

void enQueue()

{

int item;

if(Q->front == (Q->rear+1) % Q->capacity)

printf("Queue Overflow\n");

else

{

printf("Enter the element to be inserted in the queue: ");

scanf("%d", &item);

if(Q->front == -1 && Q->rear == -1)

{

Q->front = 0;

Q->rear = 0;

}

else

{

Q->rear = (Q->rear+1) % Q->capacity;

}

Q->arr[Q->rear] = item;

}

}

void deQueue()

{

if(Q->rear == -1 && Q->front == -1)

printf("Queue Underflow\n");

else

{

printf("Dequeued element: %d\n", Q->arr[Q->front]);

if(Q->front == Q->rear)

Q->front = Q->rear = -1;

else

Q->front = (Q->front + 1) % Q->capacity;

}

}

void display()

{

int i;

if(Q->front == -1 && Q->rear == -1)

{

printf("Queue is empty\n");

return;

}

printf("Queue elements :\n");

i = Q->front;

if(Q->front <= Q->rear)

{

while(i <= Q->rear)

printf("%d ",Q->arr[i++]);

}

else

{

while(i <= Q->capacity - 1)

printf("%d ",Q->arr[i++]);

i = 0;

while(i <= Q->rear)

printf("%d ",Q->arr[i++]);

}

printf("\n");

}

int main()

{

int choice;

initQueue();

while(1)

{

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Display\n");

printf("4. Quit\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1 :

enQueue();

break;

case 2 :

deQueue();

break;

case 3:

display();

break;

case 4:

exit(1);

default:

printf("Wrong choice, Try again!!\n");

}

}

return 0;

}

**43.Write a menu driven program to implement circular queue**

**operations such as Enqueue, Dequeue, Peek, Display of elements,**

**isEmpty using linked list.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\* next;

};

struct node \*f = NULL;

struct node \*r = NULL;

void enqueue(int d)

{

struct node\* n;

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

n->data = d;

n->next = NULL;

if((r==NULL)&&(f==NULL))

{

f = r = n;

r->next = f;

}

else

{

r->next = n;

r = n;

n->next = f;

}

}

void dequeue()

{

struct node\* t;

t = f;

if((f==NULL)&&(r==NULL))

printf("\nQueue is Empty");

else if(f == r){

f = r = NULL;

free(t);

}

else{

f = f->next;

r->next = f;

free(t);

}

}

void display(){

struct node\* t;

t = f;

if((f==NULL)&&(r==NULL))

printf("\nQueue is Empty");

else{

do{

printf("\n%d",t->data);

t = t->next;

}while(t != f);

}

}

int main()

{

int opt,n,i,data;

printf("Enter Your Choice:-");

do{

printf("\n\n1 for Enqueue the Data in Queue\n2 for show the Data in Queue \n3 for Dequeue the data from the Queue\n0 for Exit\n");

scanf("%d",&opt);

switch(opt){

case 1:

printf("\nEnter the number of data :");

scanf("%d",&n);

printf("\nEnter your data :");

i=0;

while(i<n){

scanf("%d",&data);

enqueue(data);

i++;

}

break;

case 2:

display();

break;

case 3:

dequeue();

break;

case 0:

break;

default:

printf("\nIncorrect Choice");

}

}while(opt!=0);

return 0;

}

1. **Write a menu driven program to implement Deques (both Input-restricted and Output-restricted) operations such as Enqueue, Dequeue, Peek, Display of elements, IsEmpty, IsFull using static array.**

#include<conio.h>

#include<stdio.h>

# define MAX\_SIZE 5

int deque\_arr[MAX\_SIZE];

int Left = -1;

int Right = -1;

void InsertRight()

{

int added\_item;

if((Left == 0 && Right == MAX\_SIZE-1) || (Left == Right+1))

{

printf("Queue Overflow\n");

return;

}

if (Left == -1)

{

Left = 0;

Right = 0;}

else

if(Right == MAX\_SIZE-1)

Right = 0;

else

Right = Right+1;

printf("Input the element for adding in queue : ");

scanf("%d", &added\_item);

deque\_arr[Right] = added\_item ;

}

void InsertLeft()

{ int added\_item;

if((Left == 0 && Right == MAX\_SIZE-1) || (Left == Right+1))

{

printf("Queue Overflow \n");

return;

}

if (Left == -1)

{

Left = 0;

Right = 0;

}

else

if(Left== 0)

Left=MAX\_SIZE-1;

else

Left=Left-1;

printf("Input the element for adding in queue : ");

scanf("%d", &added\_item);

deque\_arr[Left] = added\_item ;

}

void DeleteLeft()

{ if (Left == -1)

{ printf("Queue Under-flow\n");

return ; }

printf("Element has been deleted from queue is : %d\n",deque\_arr[Left]);

if(Left == Right)

{ Left = -1;

Right=-1; }

else

if(Left == MAX\_SIZE-1)

Left = 0;

else

Left = Left+1;

}

void DeleteRight()

{

if (Left == -1)

{

printf("Queue Under flow\n");

return ;

}

printf("Element has been deleted from queue is : %d\n",deque\_arr[Right]);

if(Left == Right)

{

Left = -1;

Right=-1; }

else

if(Right == 0)

Right=MAX\_SIZE-1;

else

Right=Right-1;

}

void Display()

{ int fpos = Left,rpos = Right;

if(Left == -1)

{

printf("Queue is empty\n");

return;

}

printf("Queue elements :\n");

if( fpos <= rpos )

{

while(fpos <= rpos)

{

printf("%d ",deque\_arr[fpos]);

fpos++; }

}

else

{

while(fpos <= MAX\_SIZE-1)

{

printf("%d ",deque\_arr[fpos]);

fpos++;

}

fpos = 0;

while(fpos <= rpos)

{

printf("%d ",deque\_arr[fpos]);

fpos++;

}

}

printf("\n");

}

void Input()

{

int Option=0;

do

{

printf("1.Insert at Right\n");

printf("2.Delete from Left\n");

printf("3.Delete from Right\n");

printf("4.Display\n");

printf("5.Quit\n");

printf("Enter your choice : ");

scanf("%d",&Option);

switch(Option)

{

case 1: InsertRight(); break;

case 2: DeleteLeft(); break;

case 3: DeleteRight(); break;

case 4: Display(); break;

case 5: break;

default: printf("Wrong Option\n");

}

}while(Option!=5);

}

void Output()

{ int Option=0;

do

{ printf("1.Insert at Right\n");

printf("2.Insert at Left\n");

printf("3.Delete from Left\n");

printf("4.Display\n");

printf("5.Quit\n");

printf("Enter your choice : ");

scanf("%d",&Option);

switch(Option)

{

case 1: InsertRight(); break;

case 2: InsertLeft(); break;

case 3: DeleteLeft(); break;

case 4: Display(); break;

case 5: break;

default: printf("Wrong Option\n");

}

}while(Option!=5);

}

main()

{ int Option=0;

printf("1.Input restricted dequeue\n");

printf("2.Output restricted dequeue\n");

printf("Enter your choice : ");

scanf("%d",&Option);

switch(Option)

{

case 1 : Input(); break;

case 2: Output(); break;

default: printf("Wrong Option\n");

}

}

**45.Write a menu driven program to implement Priority Queue using linked list**.

#include<stdio.h>

#include<stdlib.h>

struct node

{

int priority;

int info;

struct node \*link;

}\*front=NULL;

void insert(int item, int item\_priority);

int del();

void display();

int isEmpty();

int main()

{

int choice,item,item\_priority;

while(1)

{

printf("\n1.Insert\n");

printf("2.Delete\n");

printf("3.Display\n");

printf("4.Quit\n");

printf("\nEnter your choice : ");

scanf("%d", &choice);

switch(choice)

{

case 1:

printf("\nInput the item to be added in the queue : ");

scanf("%d",&item);

printf("\nEnter its priority : ");

scanf("%d",&item\_priority);

insert(item, item\_priority);

break;

case 2:

printf("\nDeleted item is %d\n",del());

break;

case 3:

display();

break;

case 4:

exit(1);

default :

printf("\nWrong choice\n");

}

}

return 0;

}

void insert(int item,int item\_priority)

{

struct node \*tmp,\*p;

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

if(tmp==NULL)

{

printf("\nMemory not available\n");

return;

}

tmp->info=item;

tmp->priority=item\_priority;

if( isEmpty() || item\_priority < front->priority )

{

tmp->link=front;

front=tmp;

}

else

{

p = front;

while( p->link!=NULL && p->link->priority<=item\_priority )

p=p->link;

tmp->link=p->link;

p->link=tmp;

}

}

int del()

{

struct node \*tmp;

int item;

if( isEmpty() )

{

printf("\nQueue Underflow\n");

exit(1);

}

else

{

tmp=front;

item=tmp->info;

front=front->link;

free(tmp);

}

return item;

}

int isEmpty()

{

if( front == NULL )

return 1;

else

return 0;

}

void display()

{

struct node \*ptr;

ptr=front;

if( isEmpty() )

printf("\nQueue is empty\n");

else

{ printf("\nQueue is :\n");

printf("\nPriority Item\n");

while(ptr!=NULL)

{

printf("%5d %5d\n",ptr->priority,ptr->info);

ptr=ptr->link;

}

}

}

**46.WAP to implement a queue using 2 stacks.**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

void push(struct node\*\* top, int data);

int pop(struct node\*\* top);

struct queue

{

struct node \*stack1;

struct node \*stack2;

};

void enqueue(struct queue \*q, int x)

{

push(&q->stack1, x);

}

void dequeue(struct queue \*q)

{

int x;

if (q->stack1 == NULL && q->stack2 == NULL) {

printf("queue is empty");

return;

}

if (q->stack2 == NULL) {

while (q->stack1 != NULL) {

x = pop(&q->stack1);

push(&q->stack2, x);

}

}

x = pop(&q->stack2);

printf("%d\n", x);

}

void push(struct node\*\* top, int data)

{

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

if (newnode == NULL) {

printf("Stack overflow \n");

return;

}

newnode->data = data;

newnode->next = (\*top);

(\*top) = newnode;

}

int pop(struct node\*\* top)

{

int buff;

struct node \*t;

if (\*top == NULL) {

printf("Stack underflow \n");

return;

}

else {

t = \*top;

buff = t->data;

\*top = t->next;

free(t);

return buff;

}

}

void display(struct node \*top1,struct node \*top2)

{

while (top1 != NULL) {

printf("%d\n", top1->data);

top1 = top1->next;

}

while (top2 != NULL) {

printf("%d\n", top2->data);

top2 = top2->next;

}

}

int main()

{

struct queue q = (struct queue)malloc(sizeof(struct queue));

int f = 0, a;

char ch = 'y';

q->stack1 = NULL;

q->stack2 = NULL;

while (ch == 'y'||ch == 'Y') {

printf("enter ur choice\n1.add to queue\n2.remove from queue\n3.display\n4.exit\n");

scanf("%d", &f);

switch(f) {

case 1 : printf("enter the element to be added to queue\n");

scanf("%d", &a);

enqueue(q, a);

break;

case 2 : dequeue(q);

break;

case 3 : display(q->stack1, q->stack2);

break;

case 4 : exit(1);

break;

default : printf("invalid\n");

break;

}

}

}

**47.WAP to reverse a stack using recursion.**

#include<stdio.h>

#include<stdlib.h>

#define bool int

struct sNode

{

char data;

struct sNode \*next;

};

void push(struct sNode\*\* top\_ref,

int new\_data);

int pop(struct sNode\*\* top\_ref);

bool isEmpty(struct sNode\* top);

void print(struct sNode\* top);

void insertAtBottom(struct sNode\*\* top\_ref,

int item)

{

if (isEmpty(\*top\_ref))

push(top\_ref, item);

else

{

int temp = pop(top\_ref);

insertAtBottom(top\_ref, item);

push(top\_ref, temp);

}

}

void reverse(struct sNode\*\* top\_ref)

{

if (!isEmpty(\*top\_ref))

{

int temp = pop(top\_ref);

reverse(top\_ref);

insertAtBottom(top\_ref, temp);

}

}

int main()

{

struct sNode \*s = NULL;

push(&s, 0);

push(&s, 9);

push(&s, 1);

push(&s, 9);

printf("\n Original Stack ");

print(s);

reverse(&s);

printf("\n Reversed Stack ");

print(s);

return 0;

}

bool isEmpty(struct sNode\* top)

{

return (top == NULL)? 1 : 0;

}

void push(struct sNode\*\* top\_ref,

int new\_data)

{

struct sNode\* new\_node =

(struct sNode\*) malloc(sizeof(struct sNode));

if (new\_node == NULL)

{

printf("Stack overflow \n");

exit(0);

}

new\_node->data = new\_data;

new\_node->next = (\*top\_ref);

(\*top\_ref) = new\_node;

}

int pop(struct sNode\*\* top\_ref)

{

char res;

struct sNode \*top;

if (\*top\_ref == NULL)

{

printf("Stack overflow \n");

exit(0);

}

else

{

top = \*top\_ref;

res = top->data;

\*top\_ref = top->next;

free(top);

return res;

}

}

void print(struct sNode\* top)

{

printf("\n");

while (top != NULL)

{

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

top = top->next;

}

}

**48.WAP to reverse a queue using recursion.**

#include<stdio.h>

#include<stdlib.h>

int f = -1,r = -1;

int q[50];

void enqueue(int data,int l)

{

if(r==l-1){

printf("Queue is full");

}

else if((f==-1)&&(r==-1)){

f = r = 0;

q[r] = data;

}

else

{

r++;

q[r] = data;

}

}

void print()

{

int i;

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

{

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

}

}

void reverse()

{

int i,j,t;

for(i=f,j=r;i<j;i++,j--){

t = q[i];

q[i] = q[j];

q[j] = t;

}

}

void main()

{

int n,i=0,t;

printf("Enter the size of Queue :");

scanf("%d",&n);

printf("\nEnter the data for Queue :");

while(i<n){

scanf("%d",&t);

enqueue(t,n);

i++;

}

printf("\nQueue which you have entered:-");

print();

reverse();

printf("\nQueue after reversing:-");

print();

}

**49.WAP to implement a queue using one stack.**

#include <stdio.h>

#include <stdlib.h>

void push1(int);

void push2(int);

int pop1();

int pop2();

void enqueue();

void dequeue();

void display();

void create();

int st1[100], st2[100];

int top1 = -1, top2 = -1;

int count = 0;

void main()

{

int ch;

printf("\n1 - Enqueue element into queue");

printf("\n2 - Dequeue element from queue");

printf("\n3 - Display from queue");

printf("\n4 - Exit");

create();

while (1)

{

printf("\nEnter choice :");

scanf("%d", &ch);

switch (ch)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

exit(0);

default:

printf("Wrong choice");

}

}

}

void create()

{

top1 = top2 = -1;

}

void push1(int data)

{

st1[++top1] = data;

}

int pop1()

{

return(st1[top1--]);

}

void push2(int data)

{

st2[++top2] = data;

}

int pop2()

{

return(st2[top2--]);

}

void enqueue()

{

int data, i;

printf("Enter data into queue :");

scanf("%d", &data);

push1(data);

count++;

}

void dequeue()

{

int i;

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

{

push2(pop1());

}

pop2();

count--;

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

{

push1(pop2());

}

}

void display()

{

int i;

for (i = 0;i <= top1;i++)

{

printf(" %d ", st1[i]);

}

}

50.WAP to implement a stack using one queue.

#include<stdio.h>

#include<stdlib.h>

int f = -1,r = -1;

int q[50];

void enqueue(int data,int l)

{

if(r==l-1){

printf("Queue is full");

}

else if((f==-1)&&(r==-1)){

f = r = 0;

q[r] = data;

}

else

{

r++;

q[r] = data;

}

}

void deque()

{

printf("%d\n",q[f]);

f++;

}

void print()

{

int i;

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

{

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

}

}

void reverse()

{

int i,j,t;

for(i=f,j=r;i<j;i++,j--){

t = q[i];

q[i] = q[j];

q[j] = t;

}

}

void push(int x)

{

enqueue(x, 50);

}

void pop(){

reverse();

deque();

reverse();

}

int main()

{

push(20);

push(30);

push(40);

pop();

push(50);

pop();

pop();

return 0;

}

**51.Write a program to implement 2 Stacks in a single array of size N. The stacks should not show overflow if there even a single slot available in the array. Write the functions popA(), popB(), pushA(), pushB() for pushing and popping from these stacks.**

#include<stdio.h>

#define MAX 10

int top1=-1;

int top2=MAX;

int stack[MAX];

void push1(int n)

{

if((top1+1)==top2 || top1==MAX-1 )

{

printf("Stack overflow!\n");

return ;

}

top1++;

stack[top1]=n;

}

void push2(int n)

{

if((top1+1)==top2 || top2==0)

{

printf("Stack overflow!\n");

return ;

}

top2--;

stack[top2]=n;

}

void pop1()

{

if(top1==-1)

{

printf("Stack underflow\n");

return ;

}

int n=stack[top1];

printf("The element popped is %d",n);

top1--;

}

void pop2()

{

if(top2==MAX+1)

{

printf("Stack underflow\n");

return ;

}

int n=stack[top2];

printf("The element popped is %d",n);

top2++;

}

void disp\_arr()

{

int i;

printf("\nDisplaying Array\n");

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

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

}

void disp\_stack1()

{

int i;

printf("\nDisplaying Stack 1\n");

for(i=top1;i>=0;i--)

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

}

void disp\_stack2()

{

int i;

printf("\nDisplaying Stack 2\n");

for(i=top2;i<MAX;i++)

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

}

int main()

{

int choice,ct=1,c,ele;

while(ct)

{

printf("\nDo you wish to continue ?(1 for yes, 0 for no): \n");

scanf("%d",&ct);

printf("Enter the stack you want to deal with:\n1 for stack 1\n2 for stack 2 \n3 to print array\n4 to display stack 1\n5 to display stack 2\n");

scanf("%d",&choice);

switch (choice)

{

case 1:

{

printf("\nPush(1) or pop(2) ? : \n");

scanf("%d",&c);

if(c==1)

{

printf("\nEnter the element you want to push: \n");

scanf("%d",&ele);

push1(ele);

}

if(c==2)

{

pop1();

}

}

break;

case 2:

{

printf("Push(1) or pop(2) ? : \n");

scanf("%d",&c);

if(c==1)

{

printf("\nEnter the element you want to push: \n");

scanf("%d",&ele);

push2(ele);

}

if(c==2)

{

pop2();

}

}

break;

case 3:

disp\_arr();

break;

case 4:

disp\_stack1();

break;

case 5:

disp\_stack2();

break;

}

}

return 0;

}

**52.WAP to create a binary tree using Array and perform pre-order, in-order, and post-order traversals.**

#include <stdio.h>

int complete\_node = 15;

char tree[] = {'\0', 'D', 'A', 'F', 'E', 'B', 'R', 'T', 'G', 'Q', '\0', '\0', 'V', '\0', 'J', 'L'};

int get\_right\_child(int index)

{

if(tree[index]!='\0' && ((2\*index)+1)<=complete\_node)

return (2\*index)+1;

return -1;

}

int get\_left\_child(int index)

{

if(tree[index]!='\0' && (2\*index)<=complete\_node)

return 2\*index;

return -1;

}

void preorder(int index)

{

if(index>0 && tree[index]!='\0')

{

printf(" %c ",tree[index]);

preorder(get\_left\_child(index));

preorder(get\_right\_child(index));

}

}

void postorder(int index)

{

if(index>0 && tree[index]!='\0')

{

postorder(get\_left\_child(index));

postorder(get\_right\_child(index));

printf(" %c ",tree[index]);

}

}

void inorder(int index)

{

if(index>0 && tree[index]!='\0')

{

inorder(get\_left\_child(index));

printf(" %c ",tree[index]);

inorder(get\_right\_child(index));

}

}

int main()

{

printf("Preorder:\n");

preorder(1);

printf("\nPostorder:\n");

postorder(1);

printf("\nInorder:\n");

inorder(1);

printf("\n");

return 0;

}

**53.Write a menu driven program to implement binary tree using linked list and different traversals.**

#include<stdio.h>

#include<conio.h>

struct tree

{

int info;

struct tree \*left;

struct tree \*right;

};

struct tree \*insert(struct tree \*root, int d)

{

if(!root)

{

root=(struct tree\*)malloc(sizeof(struct tree));

root->info = d;

root->left = NULL;

root->right = NULL;

return(root);

}

if(root->info > d)

root->left = insert(root->left,d);

else

{

if(root->info < d)

root->right = insert(root->right,d);

}

return(root);

}

void inorder(struct tree \*root)

{

if(root != NULL)

{

inorder(root->left);

printf(" %d",root->info);

inorder(root->right);

}

return;

}

void postorder(struct tree \*root)

{

if(root != NULL)

{

postorder(root->left);

postorder(root->right);

printf(" %d",root->info);

}

return;

}

void preorder(struct tree \*root)

{

if(root != NULL)

{

printf(" %d",root->info);

preorder(root->left);

preorder(root->right);

}

return;

}

int main(void)

{

struct tree \*root;

int ch, item,item\_no;

root = NULL;

do {

do

{

printf("\n-----MENU------");

printf("\n1. Insert in Binary Tree ");

printf("\n2. Inorder traversal of Binary tree");

printf("\n3. Postorder traversal of Binary tree");

printf("\n4. Preorder traversal of Binary tree");

printf("\n5. Exit ");

printf("\n-----MENU------");

printf("\n");

printf("\nEnter choice : ");

scanf(" %d",&ch);

if(ch<1 || ch>5)

printf("\n Invalid Entry");

}

while (ch<1 || ch>5);

switch(ch)

{

case 1:

printf("\n Enter new element: ");

scanf("%d", &item);

root= insert(root,item);

printf("\n root is %d",root->info);

printf("\n Binary tree is : ");

inorder(root);

break;

case 2:

printf("\n Inorder traversal: ");

inorder(root);

break;

case 3:

printf("\n Postorder traversal: ");

postorder(root);

break;

case 4:

printf("\n Preorder traversal: ");

preorder(root);

break;

default:

printf("\n EXIT ");

}

}

while(ch !=5);

return(0);

}

**54.WAP to implement non-recursive pre-order and post-order traversals using Stack data structure for binary tree.**

#include<stdio.h>

#include<stdlib.h>

#define MAX 50

struct node

{

struct node \*lchild;

int info;

struct node \*rchild;

};

struct node \*insert\_nrec(struct node \*root, int ikey );

void nrec\_pre(struct node \*root);

void nrec\_in(struct node \*root);

void nrec\_post(struct node \*root);

void display(struct node \*ptr,int level);

struct node \*queue[MAX];

int front=-1,rear=-1;

void insert\_queue(struct node \*item);

struct node \*del\_queue();

int queue\_empty();

struct node \*stack[MAX];

int top=-1;

void push\_stack(struct node \*item);

struct node \*pop\_stack();

int stack\_empty();

int main( )

{

struct node \*root=NULL, \*ptr;

int choice,k;

while(1)

{

printf("\n");

printf("1.Insert\n");

printf("2.Display\n");

printf("3.Preorder Traversal\n");

printf("4.Postorder Traversal\n");

printf("5.Quit\n");

printf("\nEnter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("\nEnter the key to be inserted : ");

scanf("%d",&k);

root = insert\_nrec(root, k);

break;

case 2:

printf("\n");

display(root,0);

printf("\n");

break;

case 3:

nrec\_pre(root);

break;

case 4:

nrec\_post(root);

break;

case 5:

exit(1);

default:

printf("\nWrong choice\n");

}

}

return 0;

}

struct node \*insert\_nrec(struct node \*root, int ikey)

{

struct node \*tmp,\*par,\*ptr;

ptr = root;

par = NULL;

while( ptr!=NULL)

{

par = ptr;

if(ikey < ptr->info)

ptr = ptr->lchild;

else if( ikey > ptr->info )

ptr = ptr->rchild;

else

{

printf("\nDuplicate key");

return root;

}

}

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

tmp->info=ikey;

tmp->lchild=NULL;

tmp->rchild=NULL;

if(par==NULL)

root=tmp;

else if( ikey < par->info )

par->lchild=tmp;

else

par->rchild=tmp;

return root;

}

void nrec\_pre(struct node \*root)

{

struct node \*ptr = root;

if( ptr==NULL )

{

printf("Tree is empty\n");

return;

}

push\_stack(ptr);

while( !stack\_empty() )

{

ptr = pop\_stack();

printf("%d ",ptr->info);

if(ptr->rchild!=NULL)

push\_stack(ptr->rchild);

if(ptr->lchild!=NULL)

push\_stack(ptr->lchild);

}

printf("\n");

}

void nrec\_post(struct node \*root)

{

struct node \*ptr = root;

struct node \*q;

if( ptr==NULL )

{

printf("Tree is empty\n");

return;

}

q = root;

while(1)

{

while(ptr->lchild!=NULL)

{

push\_stack(ptr);

ptr=ptr->lchild;

}

while( ptr->rchild==NULL || ptr->rchild==q )

{

printf("%d ",ptr->info);

q = ptr;

if( stack\_empty() )

return;

ptr = pop\_stack();

}

push\_stack(ptr);

ptr = ptr->rchild;

}

printf("\n");

}

void insert\_queue(struct node \*item)

{

if(rear==MAX-1)

{

printf("Queue Overflow\n");

return;

}

if(front==-1)

front=0;

rear=rear+1;

queue[rear]=item ;

}

struct node \*del\_queue()

{

struct node \*item;

if(front==-1 || front==rear+1)

{

printf("Queue Underflow\n");

return 0;

}

item=queue[front];

front=front+1;

return item;

}

int queue\_empty()

{

if(front==-1 || front==rear+1)

return 1;

else

return 0;

}

void push\_stack(struct node \*item)

{

if(top==(MAX-1))

{

printf("Stack Overflow\n");

return;

}

top=top+1;

stack[top]=item;

}

struct node \*pop\_stack()

{

struct node \*item;

if(top==-1)

{

printf("Stack Underflow....\n");

exit(1);

}

item=stack[top];

top=top-1;

return item;

}

int stack\_empty()

{

if(top==-1)

return 1;

else

return 0;

}

void display(struct node \*ptr,int level)

{

int i;

if(ptr == NULL )

return;

else

{

display(ptr->rchild, level+1);

printf("\n");

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

printf(" ");

printf("%d", ptr->info);

display(ptr->lchild, level+1);

}

}

1. **WAP to implement level-order traversal using single Queue data structure**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_Q\_SIZE 500

struct node

{

int data;

struct node\* left;

struct node\* right;

};

struct node\*\* createQueue(int \*, int \*);

void enQueue(struct node \*\*, int \*, struct node \*);

struct node deQueue(struct node \*, int \*);

void printLevelOrder(struct node\* root)

{

int rear, front;

struct node \*\*queue = createQueue(&front, &rear);

struct node \*temp\_node = root;

while (temp\_node)

{

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

if (temp\_node->left)

enQueue(queue, &rear, temp\_node->left);

if (temp\_node->right)

enQueue(queue, &rear, temp\_node->right);

temp\_node = deQueue(queue, &front);

}

}

struct node\*\* createQueue(int \*front, int \*rear)

{

struct node \*\*queue =

(struct node \*)malloc(sizeof(struct node)\*MAX\_Q\_SIZE);

\*front = \*rear = 0;

return queue;

}

void enQueue(struct node \*\*queue, int \*rear, struct node \*new\_node)

{

queue[\*rear] = new\_node;

(\*rear)++;

}

struct node \*deQueue(struct node \*\*queue, int \*front)

{

(\*front)++;

return queue[\*front - 1];

}

struct node\* newNode(int data)

{

struct node\* node = (struct node\*)

malloc(sizeof(struct node));

node->data = data;

node->left = NULL;

node->right = NULL;

return(node);

}

int main()

{

struct node \*root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

printf("Level Order traversal of binary tree is : ");

printLevelOrder(root);

return 0;

}

**56.Write a menu driven program to create binary tree using linked list for computing following information.**

**a) To count number of leaf nodes**

**b) To count number of non-leaf nodes**

**c) To find total number of nodes**

**d) To compute height of the binary tree**

**e) To find sum of all nodes**

**f) To find the minimum element**

**g) To find the maximum element**

#include <stdio.h>

#include <stdlib.h>

typedef struct BT

{

int data;

struct BT \*left, \*right;

} BT;

void insertBT(BT \*\*root, int data)

{

BT \*newBT = (BT \*)malloc(sizeof(BT));

newBT->data = data;

newBT->left = newBT->right = NULL;

if (!(\*root))

\*root = newBT;

else

{

BT \*prev = NULL, \*tmp = \*root;

while (tmp)

{

prev = tmp;

if (tmp->data > data)

tmp = tmp->left;

else if (tmp->data < data)

tmp = tmp->right;

}

if (prev->data > data)

prev->left = newBT;

else

prev->right = newBT;

}

}

void createBT(BT \*\*r)

{

int ch = 1;

do

{

int data = 0;

printf("\nEnter a number to enter in BT: ");

scanf("%d", &data);

insertBT(r, data);

printf("\nPress 1 to enter a value to BT or 0 to exit: ");

scanf("%d", &ch);

} while (ch == 1);

}

int countLeaf(BT \*r)

{

if (r == NULL)

return 0;

if (r->left == NULL && r->right == NULL)

return 1;

return countLeaf(r->left) + countLeaf(r->right);

}

int countNonLeafNodes(BT \*r)

{

return countNodes(r) - countLeaf(r);

}

int countNodes(BT \*r)

{

if (!r)

return 0;

return countNodes(r->left) + countNodes(r->right) + 1;

}

int sumOfNodes(BT \*r)

{

if (!r)

return 0;

return r->data + sumOfNodes(r->right) + sumOfNodes(r->left);

}

void findSmallest(BT \*r)

{

if (r == NULL)

printf("\nBT EMPTY!\n");

else

{

BT \*tmp = r;

while (tmp->left)

tmp = tmp->left;

printf("\nSmallest element in BT = %d\n", tmp->data);

}

}

void findLargest(BT \*r)

{

if (r == NULL)

printf("\nBT EMPTY!\n");

else

{

BT \*tmp = r;

while (tmp->right)

tmp = tmp->right;

printf("\nLargest element in BT = %d\n", tmp->data);

}

}

int maxDepth(BT \*r)

{

if (!r)

return 0;

else

{

int ldep = maxDepth(r->left);

int rdep = maxDepth(r->right);

if (ldep > rdep)

return ldep + 1;

else

return rdep + 1;

}

}

int main()

{

BT \*r = NULL;

while (1)

{

int ch;

printf(

"\tBinary Tree Menu\n"

"----------------------------------------\n"

"0.Quit\n"

"1.Create\n"

"2.To count number of leaf nodes\n"

"3.To count number of non-leaf nodes\n"

"4. To find total number of nodes\n"

"5. To compute height of the binary tree\n"

"6. To find sum of all nodes\n"

"7. To find the minimum element\n"

"8. To find the maximum element\n"

"----------------------------------------\n"

"Enter your choice:");

scanf("%d", &ch);

switch (ch)

{

case 0:

return 0;

break;

case 1:

createBT(&r);

break;

case 2:

printf("\nNumber of leaf nodes = %d\n", countLeaf(r));

break;

case 3:

printf("\nNumber of non-leaf nodes = %d\n", countNonLeafNodes(r));

break;

case 4:

printf("\nTotal Number of nodes = %d\n", countNodes(r));

break;

case 5:

printf("\nDepth of BT = %d\n", maxDepth(r));

break;

case 6:

printf("\nSum of data of all nodes = %d\n", sumOfNodes(r));

break;

case 7:

findSmallest(r);

break;

case 8:

findLargest(r);

break;

default:

printf("\nEnter a choice!\n");

break;

}

}

return 0;

}

**57.WAP Write the following menu driven program for the binary search tree**

**----------------------------------------**

**Binary Search Tree Menu**

**----------------------------------------**

**1. Create**

**2. In-Order Traversal**

**3. Pre-Order Traversal**

**4. Post-Order traversal**

**5. Search**

**6. Find Smallest Element**

**7. Find Largest Element**

**8. Deletion of Tree**

**9. Quit**

#include <stdio.h>

#include <stdlib.h>

typedef struct BST

{

int data;

struct BST \*left, \*right;

} BST;

void insertBST(BST \*\*root, int data)

{

BST \*newBST = (BST \*)malloc(sizeof(BST));

newBST->data = data;

newBST->left = newBST->right = NULL;

if (!(\*root))

\*root = newBST;

else

{

BST \*prev = NULL, \*tmp = \*root;

while (tmp)

{

prev = tmp;

if (tmp->data > data)

tmp = tmp->left;

else if (tmp->data < data)

tmp = tmp->right;

}

if (prev->data > data)

prev->left = newBST;

else

prev->right = newBST;

}

}

void createBST(BST \*\*root)

{

int ch = 1;

do

{

int data = 0;

printf("\nEnter a number to enter in BST: ");

scanf("%d", &data);

insertBST(root, data);

printf("\nPress 1 to enter a value to BST or 0 to exit: ");

scanf("%d", &ch);

} while (ch == 1);

}

void Inorder(BST \*root)

{

if (!root)

return;

Inorder(root->left);

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

Inorder(root->right);

}

void Preorder(BST \*root)

{

if (!root)

return;

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

Preorder(root->left);

Preorder(root->right);

}

void Postorder(BST \*root)

{

if (!root)

return;

Postorder(root->left);

Postorder(root->right);

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

}

void search(BST \*root, int data)

{

BST \*tmp = root;

while (tmp)

{

if (tmp->data > data)

tmp = tmp->left;

else if (tmp->data < data)

tmp = tmp->right;

else

{

printf("\n%d found in BST!\n", data);

return;

}

}

printf("\nElement not found in BST!\n");

}

void findSmallest(BST \*root)

{

if (root == NULL)

printf("\nBST EMPTY!\n");

else

{

BST \*tmp = root;

while (tmp->left)

tmp = tmp->left;

printf("\nSmallest element in BST = %d\n", tmp->data);

}

}

void findLargest(BST \*root)

{

if (root == NULL)

printf("\nBST EMPTY!\n");

else

{

BST \*tmp = root;

while (tmp->right)

tmp = tmp->right;

printf("\nLargest element in BST = %d\n", tmp->data);

}

}

BST\* getMinimumKey(BST \*curr)

{

while (curr->left != NULL)

curr = curr->left;

return curr;

}

void searchKey(BST \*\*curr, int key, BST \*\*parent)

{

while (\*curr != NULL && (\*curr)->data != key)

{

\*parent = (\*curr);

if (key < (\*curr)->data)

(\*curr) = (\*curr)->left;

else

(\*curr) = (\*curr)->right;

}

}

void \_deleteTree(BST \*root)

{

if (root == NULL)

return;

\_deleteTree(root->left);

\_deleteTree(root->right);

free(root);

}

void deleteTree(BST \*\*root)

{

\_deleteTree(\*root);

\*root = NULL;

}

int main()

{

BST \*root = NULL;

while (1)

{

int ch;

printf("\n----------------------------------------\n"

"\tBinary Search Tree Menu\n"

"----------------------------------------\n"

"1. Create\n"

"2. In-Order Traversal\n"

"3. Pre-Order Traversal\n"

"4. Post-Order traversal\n"

"5. Search\n"

"6. Find Smallest Element\n"

"7. Find Largest Element\n"

"8. Deletion of Tree\n"

"9. Quit\n"

"----------------------------------------\n"

"Enter your choice: ");

scanf("%d", &ch);

switch (ch)

{

case 1:

createBST(&root);

break;

case 2:

if (root == NULL)

{

printf("\nBST EMPTY!\n");

break;

}

printf("In-order traversal: ");

Inorder(root);

printf("\n");

break;

case 3:

if (root == NULL)

{

printf("\nBST EMPTY!\n");

break;

}

printf("Pre-order traversal: ");

Preorder(root);

printf("\n");

break;

case 4:

if (root == NULL)

{

printf("\nBST EMPTY!\n");

break;

}

printf("Post-order traversal: ");

Postorder(root);

printf("\n");

break;

case 5:

{

int d;

printf("Enter number to search: ");

scanf("%d", &d);

search(root, d);

}

break;

case 6:

findSmallest(root);

break;

case 7:

findLargest(root);

break;

case 8:

deleteTree(&root);

break;

default:

printf("\nEnter a valid ch!\n");

break;

case 9:

return 0;

break;

}

}

return 0;

}

**58.WAP to sort an array of n integers in an ascending order using Heap sort.**

#include <stdio.h>

void main()

{

int heap[10], no, i, j, c, root, temp;

printf("\nEnter no of elements :");

scanf("%d", &no);

printf("\nEnter the nos : ");

for (i = 0; i < no; i++)

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

for (i = 1; i < no; i++)

{

c = i;

do

{

root = (c - 1) / 2;

if (heap[root] < heap[c])

{

temp = heap[root];

heap[root] = heap[c];

heap[c] = temp;

}

c = root;

} while (c != 0);

}

printf("\nHeap array : ");

for (i = 0; i < no; i++)

printf("%d\t ", heap[i]);

for (j = no - 1; j >= 0; j--)

{

temp = heap[0];

heap[0] = heap[j];

heap[j] = temp;

root = 0;

do

{

c = 2 \* root + 1;

if ((heap[c] < heap[c + 1]) && c < j-1)

c++;

if (heap[root]<heap[c] && c<j)

{

temp = heap[root];

heap[root] = heap[c];

heap[c] = temp;

}

root = c;

} while (c < j);

}

printf("\nThe sorted array is : ");

for (i = 0; i < no; i++)

printf("\t %d", heap[i]);

}

**59.WAP to sort an array of n integers in an ascending order using merge sort.**

#include<stdio.h>

void merge(int arr[],int min,int mid,int max)

{

int tmp[30];

int i,j,k,m;

j=min;

m=mid+1;

for(i=min; j<=mid && m<=max ; i++)

{

if(arr[j]<=arr[m])

{

tmp[i]=arr[j];

j++;

}

else

{

tmp[i]=arr[m];

m++;

}

}

if(j>mid)

{

for(k=m; k<=max; k++)

{

tmp[i]=arr[k];

i++;

}

}

else

{

for(k=j; k<=mid; k++)

{

tmp[i]=arr[k];

i++;

}

}

for(k=min; k<=max; k++)

arr[k]=tmp[k];

}

void sortm(int arr[],int min,int max)

{

int mid;

if(min<max)

{

mid=(min+max)/2;

sortm(arr,min,mid);

sortm(arr,mid+1,max);

merge(arr,min,mid,max);

}

}

int main()

{

int arr[30];

int i,size;

printf("\tMerge sort\n");

printf("-----------------------------------\n");

printf("How many numbers you want to sort?: ");

scanf("%d",&size);

printf("\n Enter %d elements :\n ",size);

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

{

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

}

sortm(arr,0,size-1);

printf("\nSorted elements after using merge sort:\n\n");

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

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

return 0;

}

1. **WAP to sort an array of n doubles in a descending order using quick sort.**

#include<stdio.h>

void quicksort(double number[25],int first,int last){

int i, j, pivot, temp;

if(first<last){

pivot=first;

i=first;

j=last;

while(i<j){

while(number[i]>=number[pivot]&&i<last)

i++;

while(number[j]<number[pivot])

j--;

if(i<j){

temp=number[i];

number[i]=number[j];

number[j]=temp;

}

}

temp=number[pivot];

number[pivot]=number[j];

number[j]=temp;

quicksort(number,first,j-1);

quicksort(number,j+1,last);

}

}

int main(){

int i,count;

double number[25];

printf("Enter some elements (Max. - 25): ");

scanf("%d",&count);

printf("Enter %d elements: ", count);

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

scanf("%lf",&number[i]);

quicksort(number,0,count-1);

printf("The Sorted Order is: ");

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

printf(" %lf",number[i]);

return 0;

}

**61.WAP to sort an array of n integers in a descending order using insertion sort.**

#include <stdio.h>

int main()

{

int n, i, j, temp;

int arr[64];

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

scanf("%d", &n);

printf("Enter %d integers\n", n);

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

{

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

}

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

{

j = i;

while ( j > 0 && arr[j-1] < arr[j])

{

temp = arr[j];

arr[j] = arr[j-1];

arr[j-1] = temp;

j--;

}

}

printf("Sorted list in descending order:\n");

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

{

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

}

return 0;

}

**62.WAP to store n floats in linked list and sort them using selection sort.**

#include<stdio.h>

#include<stdlib.h>

typedef struct Node

{

float data;

struct Node \*link;

}node;

node \*head = NULL;

void print();

void swap(node \*p1, node\*p2);

void SelectionSort(node \*head);

void insert(float data, float position);

float main()

{

insert(4,1);

insert(2,2);

insert(3,3);

insert(1,4);

insert(0,5);

printf("\n Before sorting = ");

print();

SelectionSort(head);

printf("\n After sorting = ");

print();

return 0;

}

void SelectionSort(node \*head)

{

node \*start = head;

node \*traverse;

node \*min;

while(start->link)

{

min = start;

traverse = start->link;

while(traverse)

{

if( min->data > traverse->data )

{

min = traverse;

}

traverse = traverse->link;

}

swap(start,min);

start = start->link;

}

}

void swap(node \*p1, node\*p2)

{

float temp = p1->data;

p1->data = p2->data;

p2->data = temp;

}

void insert(float data, float position)

{

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

temp->data = data;

temp->link = NULL;

if(position==1)

{

temp->link = head;

head = temp;

return ;

}

node \*traverse = head;

float i;

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

{

traverse = traverse->link;

}

temp->link = traverse->link;

traverse->link = temp;

}

void print()

{

node \*p = head;

while(p)

{

printf(" %f",p->data);

p = p->link;

}

printf(" \n\n");

}

**63.WAP to store n floats in linked list and sort them using bubble sort.**

#include<stdio.h>

#include<stdlib.h>

struct Node

{

float data;

struct Node \*next;

};

void insertAtTheBegin(struct Node \*\*start\_ref, float data);

void bubbleSort(struct Node \*start);

void swap(struct Node \*a, struct Node \*b);

void printList(struct Node \*start);

int main()

{

float arr[] = {1.1, 1.2, 2.3, 1.4, 1.5, 9};

float list\_size;

int i;

struct Node \*start = NULL;

for (i = 0; i< 6; i++)

insertAtTheBegin(&start, arr[i]);

printf("\nLinked list before sorting : \n");

printList(start);

bubbleSort(start);

printf("\n\nLinked list after sorting : \n");

printList(start);

getchar();

return 0;

}

void insertAtTheBegin(struct Node \*\*start\_ref, float data)

{

struct Node\* ptr1 = (struct Node\*)malloc(sizeof(struct Node));

ptr1->data = data;

ptr1->next = \*start\_ref;

\*start\_ref = ptr1;

}

void printList(struct Node \*start)

{

struct Node \*temp = start;

printf("\n");

while (temp!=NULL)

{

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

temp = temp->next;

}

}

void bubbleSort(struct Node \*start)

{

float swapped, i;

struct Node \*ptr1;

struct Node \*lptr = NULL;

if (start == NULL)

return;

do

{

swapped = 0;

ptr1 = start;

while (ptr1->next != lptr)

{

if (ptr1->data > ptr1->next->data)

{

swap(ptr1, ptr1->next);

swapped = 1;

}

ptr1 = ptr1->next;

}

lptr = ptr1;

}

while (swapped);

}

void swap(struct Node \*a, struct Node \*b)

{

float temp = a->data;

a->data = b->data;

b->data = temp;

}

**64.Write a program for deleting all the nodes from the single linked list which are divisible by a given input number k > 1.**

#include <stdio.h>

#include <stdlib.h>

struct Node

{

int data;

struct Node \*next;

};

struct Node \*getNode(int data)

{

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

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void deleteDivisibleNodes(struct Node \*\*head\_ref, int K)

{

struct Node \*temp = \*head\_ref, \*prev;

while (temp != NULL && temp->data % K == 0)

{

\*head\_ref = temp->next; / Changed head

free(temp);

temp = \*head\_ref;

}

while (temp != NULL)

{

while (temp != NULL && temp->data % K != 0)

{

prev = temp;

temp = temp->next;

}

if (temp == NULL)

return;

prev->next = temp->next;

free(temp);

temp = prev->next;

}

}

// function to a print a linked list

void printList(struct Node \*head)

{

while (head)

{

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

head = head->next;

}

}

int main()

{

struct Node \*head = getNode(16);

head->next = getNode(15);

head->next->next = getNode(9);

head->next->next->next = getNode(12);

head->next->next->next->next = getNode(5);

head->next->next->next->next->next = getNode(6);

head->next->next->next->next->next->next = getNode(4);

int K = 4;

printf("Initial List is : ");

printList(head);

deleteDivisibleNodes(&head, K);

printf("\nFinal List is : ");

printList(head);

return 0;

}

**65.Write a function void quickSort(int \*A, int n) to sort an array of numbers.The function is supposed to sort randomly generated ‘n’ integer (including negative and positive) numbers and int \*A holds the address of dynamic array from the main function. Compute and display the number of comparisons required for sorting ‘n’ numbers. Finally, print both unsorted and sorted data**

#include <stdio.h>

#include <stdlib.h>

void quickSort(int n, int\* A)

{

qsrt(A,0,n-1);

}

void qsrt(int array[], int low, int high) {

if (low < high) {

int pi = partition(array, low, high);

qsrt(array, low, pi - 1);

qsrt(array, pi + 1, high);

}

}

int partition(int array[], int low, int high) {

int pivot = array[high];

int i = (low - 1);

int j;

for (j = low; j < high; j++) {

if (array[j] <= pivot) {

i++;

swap(&array[i], &array[j]);

}

}

swap(&array[i + 1], &array[high]);

return (i + 1);

}

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

int t = \*a;

\*a = \*b;

\*b = t;

}

void quickSort\_Comparisom(int n, int\* A)

{

int i, j, t,count=0;

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

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

if (\*(A + j) < \*(A + i)) {

t = \*(A + i);

\*(A + i) = \*(A + j);

\*(A + j) = t;

count=count+1;

}

}

}

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

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

printf("\nWas compared %d times",count);

}

int main()

{

int \*A;

int n = 8;

int randomnumber;

int i;

randomnumber = rand() % 10;

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

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

A[i] = rand() % 10;

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

}

printf("\n");

quickSort(n, A);

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

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

}

return 0;

}

|  |  |
| --- | --- |
| DECLARATION  I hereby declare that,  ☑ I have written the assignment in my own handwritting as mentioned in Handwritten  Code Section.  ☑ I have typed my source code in code editor and taken my own test case output after  running of code .  Full Signature of the Student   |  | | --- | |  | |