1. W.A.P. to display all elements of an array [Traverse Operation].

Ans:-

#include<stdio.h>

int main()

{

int arr[10],i;

printf("Enter array elements\n");

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

{

printf("Element=");

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

}

printf("Array Elements are\n");

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

{

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

}

return 0;

}

1. W.A.P. to insert an element in array at specific index.

Ans:-

#include<stdio.h>

int main()

{

int MAX;

printf("Enter array size=");

scanf("%d",&MAX);

int arr[MAX];

int index,num,i,j;

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

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

{

printf("Elements=");

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

}

printf("Array Elements are:\n");

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

{

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

}

printf("\n");

printf("Enter number to be inserted=");

scanf("%d",&num);

printf("Enter index of inserted number=");

scanf("%d",&index);

j=MAX;

MAX++;

while(j>=index)

{

arr[j+1]=arr[j];

j--;

}

arr[index]=num;

printf("\n");

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

{

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

}

return 0;

}

1. W.A.P. to delete an element in array at specific index.

Ans:-

#include<stdio.h>

int main()

{

int MAX;

printf("Enter array Size=");

scanf("%d",&MAX);

int arr[MAX];

int index,i,j;

printf("Enter Array Elements:-\n");

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

{

printf("Elements=");

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

}

printf("Array Elements before delete element are:-\n");

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

{

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

}

printf("Enter index which is to be deleted=");

scanf("%d",&index);

/\*First Logic\*/

j=index;

while(j<MAX)

{

arr[j-1]=arr[j];

j=j+1;

}

MAX--;

/\*Second Logic\*/

for(i=index;i<MAX-1;i++)

{

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

}

MAX--;

printf("Array after deletion:-\n");

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

{

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

}

return 0;

}

1. W.A.P. to search an element in array at specific index.

Ans:-

#include<stdio.h>

int main()

{

int MAX;

printf("Enter array Size=");

scanf("%d",&MAX);

int arr[MAX];

int num,i;

printf("Enter Array Elements:-\n");

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

{

printf("Elements=");

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

}

printf("Array Elements are:\n");

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

{

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

}

printf("Enter number to be searched=");

scanf("%d",&num);

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

{

if(arr[i]==num)

{

printf("searched element is =%d at index %d",arr[i],i);

}

}

return 0;

}

1. W.A.P. to update an element in array at specific index.

Ans:-

#include<stdio.h>

int main()

{

int MAX;

printf("Enter array Size=");

scanf("%d",&MAX);

int arr[MAX];

int index,num,i;

printf("Enter Array Elements:-\n");

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

{

printf("Elements=");

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

}

printf("Array Elements before upadate are:\n");

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

{

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

}

printf("Enter element to be updated=");

scanf("%d",&num);

printf("Enter index of element to be updated=");

scanf("%d",&index);

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

{

if(i==index)

{

arr[i]=num;

}

}

printf("Array Elements after update element:\n");

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

{

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

}

return 0;

}

1. Find largest & smallest number of an array with index in single dimension.

Ans:- a) Largest Number :-

#include<stdio.h>

int main()

{

int MAX;

printf("Enter array Size=");

scanf("%d",&MAX);

int arr[MAX],i;

int large,index;

printf("Enter Array Elements:-\n");

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

{

printf("Elements=");

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

}

printf("Array Elements are:\n");

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

{

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

}

large=arr[0];

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

{

if(arr[i]>large)

{

large=arr[i];

index=i;

}

}

printf("Largest element in array=%d at index %d\n ",large,index);

return 0;

}

b) Smallest Number:-

#include<stdio.h>

int main()

{

int MAX;

printf("Enter array Size=");

scanf("%d",&MAX);

int arr[MAX],i;

int small,index;

printf("Enter Array Elements:-\n");

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

{

printf("Elements=");

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

}

printf("Array Elements are:\n");

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

{

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

}

small=arr[0];

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

{

if(arr[i]<small)

{

small=arr[i];

index=i;

}

}

printf("Largest element in array=%d at index %d\n ",small,index);

return 0;

}

1. W.A.P. to find Transpose of array.

Ans:-

#include<stdio.h>

int main()

{

int row,col;

printf("Enter row size=");

scanf("%d",&row);

printf("Enter column size=");

scanf("%d",&col);

int arr1[row][col],t\_arr[col][row];

int i,j;

printf("Enter array elements\n");

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

{

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

{

printf("Elements=");

scanf("%d",&arr1[i][j]);

}

}

printf("Matrix elements are\n");

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

{

printf("\n");

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

{

printf("arr1[%d][%d]=%d\t",i,j,arr1[i][j]);

}

}

printf("\n");

printf("Transpose of Matrix\n");

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

{

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

{

t\_arr[j][i]=arr1[i][j];

}

}

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

{

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

{

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

}

printf("\n");

}

return 0;

}

1. W.A.P. to sort an array.

Ans:- a)Ascending Order

#include<stdio.h>

int main()

{

int MAX;

printf("Enter array Size=");

scanf("%d",&MAX);

int arr[MAX],i,j,k,temp;

printf("Enter Array Elements:-\n");

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

{

printf("Elements=");

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

}

printf("Array Elements before sorting in ascending order are:\n");

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

{

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

}

i=0;

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

{

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

{

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

{

temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

}

printf("Array Elements after sorting in ascending order are:\n");

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

{

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

}

return 0;

}

b)Descending Order

#include<stdio.h>

int main()

{

int MAX;

printf("Enter array Size=");

scanf("%d",&MAX);

int arr[MAX],i,j,k,temp;

printf("Enter Array Elements:-\n");

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

{

printf("Elements=");

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

}

printf("Array Elements before sorting in descending order are:\n");

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

{

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

}

i=0;

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

{

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

{

if(arr[j]<arr[j+1])

{

temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

}

printf("Array Elements after sorting in descending order are:\n");

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

{

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

}

return 0;

}

1. W.A.P. to find the addition of elements of subarray

i.e. suppose array[5]={1,5,7,9,10} , then add index from 2 to 4.

Ans:-

#include<stdio.h>

int main()

{

int MAX;

printf("Enter array Size=");

scanf("%d",&MAX);

int arr[MAX],sub\_index1,sub\_index2,sum=0,i;

printf("Enter Array Elements:-\n");

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

{

printf("Elements=");

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

}

printf("Enter 1st index position for addition=");

scanf("%d",&sub\_index1);

printf("Enter last index position for addition=");

scanf("%d",&sub\_index2);

for(i=sub\_index1;i<=sub\_index2;i++)

{

sum+=arr[i];

}

printf("Addition of subarray=%d\n",sum);

return 0;

}

10. W.A.P. to find duplicate element of an array.

Ans:-

#include<stdio.h>

int main()

{

int MAX;

printf("Enter array Size=");

scanf("%d",&MAX);

int arr[MAX],dup[MAX],temp,i,j=0;

printf("Enter Array Elements:-\n");

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

{

printf("Elements=");

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

}

temp=arr[0];

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

{

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

{

if(arr[i]==arr[j])

{

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

}

}

}

return 0;

}

11. W.A.P. to find maximum & minimum number and find factorial of that number.

Ans:-a)Factorial of Largest Number

#include<stdio.h>

int main()

{

int MAX;

printf("Enter array Size=");

scanf("%d",&MAX);

int arr[MAX],fact=1,num,temp,index,i;

printf("Enter Array Elements:-\n");

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

{

printf("Elements=");

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

}

printf("Array Elements are:\n");

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

{

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

}

temp=arr[0];

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

{

if(arr[i]>temp)

{

temp=arr[i];

index=i;

}

}

printf("largest number of an array=%d at index %d\n",temp,index);

num=temp;

while(num!=0)

{

fact\*=num;

num--;

}

printf("Factorial of largest number =%d\n",fact);

return 0;

}

b)Factorial of smallest number

#include<stdio.h>

int main()

{

int MAX;

printf("Enter array Size=");

scanf("%d",&MAX);

int arr[MAX],fact=1,num,temp,index,i;

printf("Enter Array Elements:-\n");

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

{

printf("Elements=");

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

}

printf("Array Elements are:\n");

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

{

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

}

temp=arr[0];

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

{

if(arr[i]<temp)

{

temp=arr[i];

index=i;

}

}

printf("Smallest number of an array=%d at index %d\n",temp,index);

num=temp;

while(num!=0)

{

fact\*=num;

num--;

}

printf("Factorial of smallest number =%d\n",fact);

return 0;

}

12. W.A.P. to merge two array.

Ans:-

#include<stdio.h>

int main()

{

int MAX1,MAX2,MAX3;

printf("Enter 1st array Size=");

scanf("%d",&MAX1);

printf("Enter 2nd array Size=");

scanf("%d",&MAX2);

MAX3=MAX1+MAX2;

int arr1[MAX1],arr2[MAX2],arr3[MAX3],i=0,j=0;

printf("Enter 1st Array Elements:-\n");

for(i=0;i<MAX1;i++)

{

printf("Elements=");

scanf("%d",&arr1[i]);

}

printf("Array Elements are:\n");

for(i=0;i<MAX1;i++)

{

printf("arr1[%d]=%d\n",i,arr1[i]);

}

printf("Enter 2nd Array Elements:-\n");

for(i=0;i<MAX2;i++)

{

printf("Elements=");

scanf("%d",&arr2[i]);

}

printf("Array Elements are:\n");

for(i=0;i<MAX2;i++)

{

printf("arr2[%d]=%d\n",i,arr2[i]);

}

for(i=0;i<MAX1;i++)

{

arr3[j]=arr1[i];

j++;

}

i=0;

for(i=0;i<MAX2;i++)

{

arr3[j]=arr2[i];

j++;

}

i=0;

printf("Merge array is\n");

for(i=0;i<MAX3;i++)

{

printf("arr3[%d]=%d\n",i,arr3[i]);

}

return 0;

}

13.W.A.P. to rotate array.

Ans:-

#include<stdio.h>

void array\_rotate(int arr[],int size)

{

int temp,i;

temp=arr[size-1];

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

{

arr[i]=arr[i-1];

}

arr[0]=temp;

}

int main()

{

int MAX;

printf("Enter array size=");

scanf("%d",&MAX);

int arr[MAX],i;

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

{

printf("Elements=");

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

}

printf("Before rotate array elements are:\n");

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

{

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

}

array\_rotate(arr,MAX);

printf("After 1st rotate array elements are:\n");

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

{

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

}

array\_rotate(arr,MAX);

printf("After 2nd rotate array elements are:\n");

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

{

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

}

array\_rotate(arr,MAX);

printf("After 3rd rotate array elements are:\n");

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

{

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

}

array\_rotate(arr,MAX);

printf("After 4th rotate array elements are:\n");

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

{

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

}

array\_rotate(arr,MAX);

printf("After 5th rotate array elements are:\n");

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

{

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

}

array\_rotate(arr,MAX);

printf("After 6th rotate array elements are:\n");

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

{

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

}

array\_rotate(arr,MAX);

printf("After 7th rotate array elements are:\n");

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

{

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

}

return 0;

}

Stack using Array-Static Memory Allocation

1. W.A.P. to perform stack operation.

Ans:-

#include<stdio.h>

#define MAX 5

int stack[MAX],top=-1,element;

void PUSH(int x);

int POP();

void PEEK(void);

void TRAVERSE(void);

int isEpty(void);

int isFull(void);

void PUSH(int x)

{

if(isFull())

{

printf("stack is Full\n");

}

else

{

top++;

stack[top]=x;

}

}

int POP()

{

if(isEmpty())

{

printf("Stack is Empty\n");

}

else

{

return stack[top--];

}

}

void PEEK(void)

{

if(isEmpty())

{

printf("Stack is Empty\n");

}

else

{

printf("Topmost element is=%d",stack[top]);

}

}

void TRAVERSE(void)

{

if(isEmpty())

{

printf("Stack is Empty\n");

}

else

{

printf("Stack Elements are:\n");

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

{

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

}

}

}

int isEmpty()

{

if(top==-1)

{

return 1;

}

else

{

return 0;

}

}

int isFull()

{

if(top==MAX-1)

{

return 1;

}

else

{

return 0;

}

}

int main()

{

int choice;

while(1)

{

printf("Operation on Stack:\n");

printf("1.PUSH\n");

printf("2.POP\n");

printf("3.PEEK\n");

printf("4.TRAVERSE\n");

printf("Enter operation=");

scanf("%d",&choice);

switch(choice)

{

case 1:printf("Enter element=");

scanf("%d",&element);

PUSH(element);

break;

case 2:element=POP();

if(element==0)

{

printf("Stack is Empty");

}

else

{

printf("%d\n",element);

}

break;

case 3:PEEK();

break;

case 4:TRAVERSE();

break;

default:printf("Enter correct choice\n");

break;

}

}

return 0;

}

1. W.A.P. to reverse a string using stack.

Ans:-

#include<stdio.h>

#define MAX 5

int stack[MAX],top=-1,element;

void PUSH(char x);

int POP(void);

void TRAVERSE(void);

int isFull();

int isEmpty();

void PUSH(char x)

{

if(isFull())

{

printf("Stack is Full\n");

}

else

{

top++;

stack[top]=x;

}

}

int POP()

{

if(isEmpty())

{

printf("Stack is Empty");

}

else

{

return stack[top--];

}

}

void TRAVERSE()

{

if(isEmpty())

{

printf("Stack is Empty\n");

}

else

{

printf("Stack elements are:\n");

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

{

printf("%c",stack[i]);

}

}

}

int isFull()

{

if(top==MAX-1)

{

return 1;

}

else

{

return 0;

}

}

int isEmpty()

{

if(top==-1)

{

return 1;

}

else

{

return 0;

}

}

int main()

{

char item,rev[MAX];

PUSH('H');

PUSH('E');

PUSH('L');

PUSH('L');

PUSH('O');

PUSH('\0');

PUSH(35);

printf("Before reversing stack is:\n");

TRAVERSE();

printf("\n");

item=POP();

rev[MAX-5]=item;

item=POP();

rev[MAX-4]=item;

item=POP();

rev[MAX-3]=item;

item=POP();

rev[MAX-2]=item;

item=POP();

rev[MAX-1]=item;

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

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

{

printf("%c",rev[i]);

}

return 0;

}

1. W.A.P. to reverse number using stack

Ans:-

#include<stdio.h>

#define MAX 5

int stack[MAX],top=-1,element;

void PUSH(int x);

int POP(void);

void TRAVERSE(void);

int isFull();

int isEmpty();

void PUSH(int x)

{

if(isFull())

{

printf("Stack is Full\n");

}

else

{

top++;

stack[top]=x;

}

}

int POP()

{

if(isEmpty())

{

printf("Stack is Empty");

}

else

{

return stack[top--];

}

}

void TRAVERSE()

{

if(isEmpty())

{

printf("Stack is Empty\n");

}

else

{

printf("Stack elements are:\n");

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

{

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

}

}

}

int isFull()

{

if(top==MAX-1)

{

return 1;

}

else

{

return 0;

}

}

int isEmpty()

{

if(top==-1)

{

return 1;

}

else

{

return 0;

}

}

int main()

{

int item,rev[MAX];

PUSH(10);

PUSH(12);

PUSH(15);

PUSH(20);

PUSH(25);

PUSH(30);

PUSH(35);

printf("Before reversing stack is:\n");

TRAVERSE();

item=POP();

rev[MAX-5]=item;

item=POP();

rev[MAX-4]=item;

item=POP();

rev[MAX-3]=item;

item=POP();

rev[MAX-2]=item;

item=POP();

rev[MAX-1]=item;

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

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

{

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

}

return 0;

}

1. W.A.P. to sort a stack in ascending order.

Ans:-

#include<stdio.h>

#define MAX 5

int stack[MAX],top=-1,element;

void PUSH(int x);

void TRAVERSE(void);

int isFull();

int isEmpty();

void PUSH(int x)

{

if(isFull())

{

printf("Stack is Full\n");

}

else

{

top++;

stack[top]=x;

}

}

void TRAVERSE()

{

if(isEmpty())

{

printf("Stack is Empty\n");

}

else

{

printf("Stack elements are:\n");

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

{

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

}

}

}

int isFull()

{

if(top==MAX-1)

{

return 1;

}

else

{

return 0;

}

}

int isEmpty()

{

if(top==-1)

{

return 1;

}

else

{

return 0;

}

}

int main()

{

int temp,i,j;

PUSH(10);

PUSH(100);

PUSH(45);

PUSH(20);

PUSH(35);

PUSH(30);

PUSH(35);

printf("Before sorting stack is:\n");

TRAVERSE();

printf("\n");

printf("\n");

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

{

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

{

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

{

temp=stack[j];

stack[j]=stack[j+1];

stack[j+1]=temp;

}

}

}

i=0;

printf("Sorted Stack in ascending order is:\n");

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

{

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

}

return 0;

}

1. W.A.P. to sort stack in descending order.

Ans:-

#include<stdio.h>

#define MAX 5

int stack[MAX],top=-1,element;

void PUSH(int x);

void TRAVERSE(void);

int isFull();

int isEmpty();

void PUSH(int x)

{

if(isFull())

{

printf("Stack is Full\n");

}

else

{

top++;

stack[top]=x;

}

}

void TRAVERSE()

{

if(isEmpty())

{

printf("Stack is Empty\n");

}

else

{

printf("Stack elements are:\n");

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

{

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

}

}

}

int isFull()

{

if(top==MAX-1)

{

return 1;

}

else

{

return 0;

}

}

int isEmpty()

{

if(top==-1)

{

return 1;

}

else

{

return 0;

}

}

int main()

{

int temp,i,j;

PUSH(10);

PUSH(100);

PUSH(45);

PUSH(20);

PUSH(35);

PUSH(30);

PUSH(35);

printf("Before sorting stack is:\n");

TRAVERSE();

printf("\n");

printf("\n");

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

{

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

{

if(stack[j]<stack[j+1])

{

temp=stack[j];

stack[j]=stack[j+1];

stack[j+1]=temp;

}

}

}

i=0;

printf("Sorted Stack in descending order is:\n");

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

{

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

}

return 0;

}

**Stack Using Linked List**

1. W.A.P. to implement Stack operation using linked list.

Ans:-

#include<stdio.h>

void PUSH(int data);

void POP();

void TRAVERSE();

int PEEK();

struct stack

{

int data;

struct stack \*link;

};

struct stack \*top=NULL;

void PUSH(int data)

{

struct stack \*temp;

temp=(int\*)malloc(sizeof(struct stack));

temp->data=data;

temp->link=top;

top=temp;

}

void POP()

{

struct stack \*temp;

if(top==NULL)

{

printf("Stack is Empty\n");

}

else

{

temp=top;

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

top=top->link;

free(temp);

}

}

void TRAVERSE()

{

struct stack \*temp=NULL;

temp=top;

if(temp==NULL)

{

printf("Stack is Empty\n");

}

else

{

temp=top;

while(temp!=NULL)

{

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

temp=temp->link;

}

}

}

int PEEK()

{

struct stack \*temp=NULL;

temp=top;

if(temp==NULL)

{

printf("Stack is Empty\n");

}

else

{

return temp->data;

}

}

int main()

{

int val,val1;

PUSH(10);

PUSH(20);

PUSH(30);

PUSH(40);

PUSH(50);

TRAVERSE();

val=PEEK();

printf("Top most element of stack=%d\n",val);

val1=PEEK();

printf("Top most element of stack=%d\n",val1);

POP();

POP();

POP();

POP();

POP();

POP();

POP();

return 0;

}

1. W.A.P. to implement operations on Queue.

Ans:-

#include<stdio.h>

#define MAX 5

int rear=0;

int front=0;

int queue[MAX];

void enqueue(int item);

void dequeue();

void traverse();

void peek();

void enqueue(int item)

{

if(rear==MAX)

{

printf("Queue is Full\n");

}

else

{

queue[rear]=item;

rear++;

}

}

void dequeue()

{

if(front==rear)

{

printf("Queue is Empty\n");

}

else

{

printf("Deleted item=%d\n",queue[front]);

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

{

queue[i]=queue[i+1];

}

rear--;

}

}

void traverse()

{

if(front==rear)

{

printf("Queue is Empty\n");

}

else

{

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

{

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

}

}

}

void peek()

{

if(front==rear)

{

printf("Queue is Empty\n");

}

else

{

printf("Peek element=%d\n",queue[front]);

}

}

int main()

{

enqueue(10);

enqueue(20);

enqueue(30);

enqueue(40);

enqueue(50);

enqueue(50);

traverse();

peek();

dequeue();

dequeue();

dequeue();

dequeue();

dequeue();

dequeue();

return 0;

}

1. Reverse a Queue

Ans:-

#include<stdio.h>

#define MAX 5

int rear=0;

int front=0;

int queue[MAX];

int arr[MAX];

void enqueue(int item);

int dequeue();

void traverse();

void reverse\_queue();

void enqueue(int item)

{

if(rear==MAX)

{

printf("Queue is Full\n");

}

else

{

queue[rear]=item;

rear++;

}

}

int dequeue()

{

if(front==rear)

{

printf("Queue is Empty\n");

}

else

{

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

{

queue[i]=queue[i+1];

}

rear--;

}

return queue[front];

}

void traverse()

{

if(front==rear)

{

printf("Queue is Empty\n");

}

else

{

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

{

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

}

}

}

void reverse\_queue()

{

int i,j,temp;

for(i=front,j=rear;i<=j;i++,j--)

{

temp=queue[i];

queue[i]=queue[j-1];

queue[j-1]=temp;

}

}

int main()

{

int i;

enqueue(10);

enqueue(20);

enqueue(30);

enqueue(40);

enqueue(50);

enqueue(50);

traverse();

reverse\_queue();

printf("After reversing a queue:\n");

traverse();

return 0;

}

1. W.A.P. to sort a queue.

Ans:-

#include<stdio.h>

#define MAX 5

int rear=0;

int front=0;

int queue[MAX];

int arr[MAX];

void enqueue(int item);

int dequeue();

void traverse();

void enqueue(int item)

{

if(rear==MAX)

{

printf("Queue is Full\n");

}

else

{

queue[rear]=item;

rear++;

}

}

int dequeue()

{

if(front==rear)

{

printf("Queue is Empty\n");

}

else

{

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

{

queue[i]=queue[i+1];

}

rear--;

}

return queue[front];

}

void traverse()

{

if(front==rear)

{

printf("Queue is Empty\n");

}

else

{

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

{

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

}

}

}

int main()

{

int i,j,temp,arr[MAX];

enqueue(40);

enqueue(30);

enqueue(10);

enqueue(50);

enqueue(20);

enqueue(50);

traverse();

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

{

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

{

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

{

temp=queue[j];

queue[j]=queue[j+1];

queue[j+1]=temp;

}

}

}

printf("Queue after Sorting in Ascending Order:\n");

traverse();

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

{

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

{

if(queue[j] < queue[j+1])

{

temp=queue[j];

queue[j]=queue[j+1];

queue[j+1]=temp;

}

}

}

printf("Queue after Sorting in Descending Order:\n");

traverse();

return 0;

}

Queue Using Linked List

1. W.A.P. to implement Queue Operation using Linked List.

#include<stdio.h>

void enqueue(int data);

void dequeue();

void TRAVERSE();

int PEEK();

struct queue

{

int data;

struct queue \*link;

};

struct queue \*front=NULL;

struct queue \*rear=NULL;

void enqueue(int data)

{

struct queue \*temp;

temp=(int\*)malloc(sizeof(struct queue));

temp->data=data;

temp->link=NULL;

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

{

printf("Queue is Empty\n");

front=rear=temp;

}

else

{

rear->link=temp;

rear=temp;

}

}

void dequeue()

{

struct queue \*temp;

if(front==NULL)

{

printf("Queue is Empty\n");

}

else

{

temp=front;

printf("Dequeue element=%d\n",front->data);

front=front->link;

free(temp);

}

}

void TRAVERSE()

{

struct queue \*temp;

if(front==NULL)

{

printf("Queue is empty\n");

front=rear=temp;

}

else

{

temp=front;

while(temp!=NULL)

{

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

temp=temp->link;

}

}

}

int PEEK()

{

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

{

printf("Queue is empty\n");

}

else

{

return front->data;

}

}

int main()

{

int val,val1;

enqueue(10);

enqueue(20);

enqueue(30);

enqueue(40);

enqueue(50);

TRAVERSE();

val=PEEK();

printf("Front element of Queue=%d\n",val);

val1=PEEK();

printf("Front element of Queue=%d\n",val1);

dequeue();

dequeue();

dequeue();

dequeue();

dequeue();

dequeue();

dequeue();

}

1. W.A.P. for Linear Search

Ans:-

/\*Linear Search\*/

#include<stdio.h>

int main()

{

int arr[10],i,num,flag=0;

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

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

{

printf("Elements:-");

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

}

printf("Array elements are:-\n");

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

{

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

}

printf("Enter element to be searched=");

scanf("%d",&num);

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

{

if(arr[i]==num)

{

flag=1;

printf("Searched element=%d\n",num);

}

}

if(flag==0)

{

printf("%d is not in an array",num);

}

return 0;

}

1. W.A.P. for binary search

Ans:-

/\*Binary Search\*/

#include<stdio.h>

void sort(int \*arr,int size);

void sort(int \*arr,int size)

{

int temp;

int i,j;

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

{

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

{

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

{

temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

}

}

int main()

{

int size;

printf("Enter size of array=");

scanf("%d",&size);

int arr[size],i,num,flag=0,mid,start,end;

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

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

{

printf("Elements:-");

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

}

printf("Array elements are:-\n");

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

{

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

}

sort(arr,size);

printf("Sorted array is:\n");

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

{

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

}

printf("\n");

printf("\n");

printf("Enter element to be searched=");

scanf("%d",&num);

start=0;

end=size-1;

while(start<=end)

{

mid=(start+end)/2;

if(arr[mid]==num)

{

flag=1;

printf("searched element=%d\n",num);

printf("Index postion of searched element=%d",mid);

break;

}

else if(arr[mid]>num)

{

end=mid-1;

}

else

{

start=mid+1;

}

}

if(start>end && flag==0)

{

printf("%d is not in an array",num);

}

return 0;

}

1. W.A.P. for interpolation search.

Ans:-

/\*Interpolation Search

Step 1:- Sort array

Step 2:- use formula i.e. pos=low + (high-low)\*[(num-arr[low])/arr[high]-arr[low]]\*/

#include<stdio.h>

#include<math.h>

void sort(int \*arr,int size);

int interpolation\_sort(int \*arr,int size,int num);

void sort(int \*arr,int size)

{

int temp;

int i,j;

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

{

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

{

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

{

temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

}

}

int interpolation\_sort(int \*arr,int size,int num)

{

int mid,high,low;

low=0;

high=size-1;

while(low<=high)

{

mid = low + (high+low)\*((num-arr[low])/(arr[high]-arr[low]));

if(num==arr[mid])

{

return mid;

}

else if(num<arr[mid])

{

high=mid-1;

}

else

{

low=mid+1;

}

}

return -1;

}

int main()

{

int size;

printf("Enter size of array=");

scanf("%d",&size);

int arr[size],i,num,pos;

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

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

{

printf("Elements:-");

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

}

printf("Array elements are:-\n");

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

{

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

}

sort(arr,size);

printf("Sorted array is:\n");

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

{

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

}

printf("\n");

printf("\n");

printf("Enter element to be searched=");

scanf("%d",&num);

pos=interpolation\_sort(arr,size,num);

if(pos==-1)

{

printf("%d is not in an array",num);

}

else

{

printf("Searched element=%d\n",num);

printf("Index of searched elemet=%d\n",pos);

}

return 0;

}

23.W.A.P. for jump search.

Ans:-

/\*Jump Search

Step 1:- Sort array

Step 2:- use formula i.e. step\_size=square\_root(Array\_size)\*/

#include<stdio.h>

#include<math.h>

void sort(int \*arr,int size);

int jump\_search(int \*arr,int size,int num);

void sort(int \*arr,int size)

{

int temp;

int i,j;

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

{

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

{

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

{

temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

}

}

int jump\_search(int \*arr,int size,int num)

{

int step,high,low,i;

low=0;

high=size-1;

step=sqrt(size);

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

{

if(num<arr[step])

{

high=step-1;

}

else

{

low=step+1;

}

}

for(i=low;i<=high;i++)

{

if(arr[i]==num)

{

return i;

}

}

return -1;

}

int main()

{

int size;

printf("Enter size of array=");

scanf("%d",&size);

int arr[size],i,num,pos;

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

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

{

printf("Elements:-");

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

}

printf("Array elements are:-\n");

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

{

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

}

sort(arr,size);

printf("Sorted array is:\n");

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

{

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

}

printf("\n");

printf("\n");

printf("Enter element to be searched=");

scanf("%d",&num);

pos=jump\_search(arr,size,num);

if(pos==-1)

{

printf("%d is not in an array",num);

}

else

{

printf("Searched element=%d\n",num);

printf("Index of searched elemet=%d\n",pos);

}

return 0;

}

1. W.AP. to implement all operations of Single Linked List

Ans:-

#include<stdio.h>

void create\_node(int data);

void insert\_node\_at\_beginning(int data);

void insert\_node\_at\_end(int data);

void insert\_node\_at\_specific(int data);

void delete\_begin\_node(void);

void delete\_end\_node(void);

void delete\_specific\_node(void);

void delete\_list(void);

int node\_length(void);

void display(void);

struct node

{

int data;

struct node \*link;

};

struct node \*root=NULL;

void create\_node(int data)

{

struct node \*temp;

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

temp -> data=data;

temp ->link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

struct node \*ptr;

ptr=root;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

}

void insert\_node\_at\_beginning(int data)

{

struct node \*temp;

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

temp->link=root;

temp->data=data;

root=temp;

}

void insert\_node\_at\_end(int data)

{

struct node \*temp;

struct node \*ptr;

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

temp->data=data;

temp->link=NULL;

ptr=root;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

void insert\_node\_at\_specific(int data)

{

struct node \*temp;

struct node \*ptr;

int node\_pos,length,i=1;

printf("Enter node position for which insert a new node=");

scanf("%d",&node\_pos);

length=node\_length();

printf("Length of list=%d\n",length);

temp->data=data;

ptr=root;

if(node\_pos==1)

{

temp->link=temp;

root=temp;

}

else

{

for(i=1;i<node\_pos-1;i++)

{

ptr=ptr->link;

}

temp->link=ptr->link;

ptr->link=temp;

}

}

void delete\_begin\_node()

{

struct node \*ptr;

ptr=root;

root=root->link;

free(ptr);

}

void delete\_end\_node(void)

{

struct node \*temp=NULL;

struct node \*ptr=NULL;

temp=root;

while(temp->link!=NULL)

{

ptr=temp;

temp=temp->link;

}

ptr->link=NULL;

free(temp);

}

void delete\_specific\_node()

{

struct node \*temp=NULL;

struct node \*ptr=NULL;

struct node \*ptr1=NULL;

int node\_loc,length,i=1;

printf("Enter node location which is to be deleted=");

scanf("%d",&node\_loc);

length=node\_length();

printf("Length of node is=%d\n",length);

if(node\_loc>length)

{

printf("location is not present in list\n");

}

else if(node\_loc==1)

{

temp=root;

root=temp->link;

temp->link=NULL;

free(temp);

}

else

{

ptr=root;

while(i<node\_loc-1)

{

ptr=ptr->link;

i++;

}

ptr1=ptr->link;

ptr->link=ptr1->link;

ptr1->link=NULL;

free(ptr1);

}

}

void delete\_list()

{

struct node \*ptr=NULL;

if(root!=NULL)

{

ptr=root;

while(ptr!=NULL)

{

root=root->link;

ptr=root;

}

}

}

int node\_length()

{

struct node \*temp;

int count=0;

temp=root;

while(temp!=NULL)

{

count++;

temp=temp->link;

}

return count;

}

void display()

{

struct node \*temp;

temp=root;

while(temp!=NULL)

{

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

temp=temp->link;

}

}

int main()

{

int length;

create\_node(20);

create\_node(50);

create\_node(70);

create\_node(90);

create\_node(100);

display();

length=node\_length();

printf("Length of Single List is=%d\n",length);

insert\_node\_at\_specific(65);

display();

length=node\_length();

printf("Length of Single List is=%d\n",length);

insert\_node\_at\_beginning(10);

display();

length=node\_length();

printf("Length of Single List is=%d\n",length);

insert\_node\_at\_end(120);

display();

length=node\_length();

printf("Length of Single List is=%d\n",length);

delete\_begin\_node();

display();

length=node\_length();

printf("Length of Single List is=%d\n",length);

delete\_end\_node();

display();

length=node\_length();

printf("Length of Single List is=%d\n",length);

delete\_specific\_node();

display();

length=node\_length();

printf("Length of Single List is=%d\n",length);

delete\_list();

printf("Delete all list\n");

length=node\_length();

printf("Length of Single List is=%d\n",length);

}

1. W.A.P. to reverse a single linked list.

Ans:-

#include<stdio.h>

void create\_node(int data);

void reverse\_list(void);

int node\_length(void);

void display(void);

struct node

{

int data;

struct node \*link;

};

struct node \*root=NULL;

void create\_node(int data)

{

struct node \*temp;

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

temp -> data=data;

temp ->link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

struct node \*ptr;

ptr=root;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

}

void reverse\_list()

{

struct node \*prev=NULL;

struct node \*next=NULL;

struct node \*temp;

temp=root;

while(temp!=NULL)

{

next=temp->link;

temp->link=prev;

prev=temp;

temp=next;

}

root=prev;

}

int node\_length()

{

struct node \*temp;

int count=0;

temp=root;

while(temp!=NULL)

{

count++;

temp=temp->link;

}

return count;

}

void display()

{

struct node \*temp;

temp=root;

while(temp!=NULL)

{

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

temp=temp->link;

}

}

int main()

{

int length;

create\_node(20);

create\_node(50);

create\_node(70);

create\_node(90);

create\_node(100);

display();

length=node\_length();

printf("Length of Single List is=%d\n",length);

reverse\_list();

display();

length=node\_length();

printf("Length of Single List is=%d\n",length);

return 0;

}

1. W.A.P. to sort a single linked list.

Ans:-

#include<stdio.h>

void create\_node(int data);

void sort\_list\_ascending(void);

void sort\_list\_descending(void);

int node\_length(void);

void display(void);

struct node

{

int data;

struct node \*link;

};

struct node \*root=NULL;

void create\_node(int data)

{

struct node \*temp;

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

temp -> data=data;

temp ->link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

struct node \*ptr;

ptr=root;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

}

void sort\_list\_ascending()

{

struct node \*ptr1;

struct node \*ptr2;

int temp;

ptr1=root;

while(ptr1->link!=NULL)

{

ptr2=ptr1->link;

while(ptr2!=NULL)

{

if(ptr1->data >ptr2->data)

{

temp=ptr1->data;

ptr1->data = ptr2->data;

ptr2->data=temp;

}

ptr2=ptr2->link;

}

ptr1=ptr1->link;

}

}

void sort\_list\_descending()

{

struct node \*ptr1;

struct node \*ptr2;

int temp;

ptr1=root;

while(ptr1->link!=NULL)

{

ptr2=ptr1->link;

while(ptr2!=NULL)

{

if(ptr1->data < ptr2->data)

{

temp=ptr1->data;

ptr1->data = ptr2->data;

ptr2->data=temp;

}

ptr2=ptr2->link;

}

ptr1=ptr1->link;

}

}

int node\_length()

{

struct node \*temp;

int count=0;

temp=root;

while(temp!=NULL)

{

count++;

temp=temp->link;

}

return count;

}

void display()

{

struct node \*temp;

temp=root;

while(temp!=NULL)

{

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

temp=temp->link;

}

}

int main()

{

int length;

create\_node(20);

create\_node(0);

create\_node(200);

create\_node(15);

create\_node(1);

display();

length=node\_length();

printf("Length of Single List is=%d\n",length);

printf("List after sorting in ascending order\n");

sort\_list\_ascending();

display();

length=node\_length();

printf("Length of Single List is=%d\n",length);

printf("List after sorting in descending order\n");

sort\_list\_descending();

display();

length=node\_length();

printf("Length of Single List is=%d\n",length);

return 0;

}

1. W.A.P. to find largest and smallest number of linked list.

Ans:-

#include<stdio.h>

void create\_node(int data);

int list\_smallest\_num(void);

int list\_largest\_num(void);

int node\_length(void);

void display(void);

struct node

{

int data;

struct node \*link;

};

struct node \*root=NULL;

void create\_node(int data)

{

struct node \*temp;

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

temp -> data=data;

temp ->link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

struct node \*ptr;

ptr=root;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

}

int list\_smallest\_num()

{

struct node \*ptr;

int min;

ptr=root;

min=ptr->data;

while(ptr !=NULL)

{

if(min > ptr->data)

{

min=ptr->data;

}

ptr=ptr->link;

}

return min;

}

int list\_largest\_num()

{

struct node \*ptr;

int max;

ptr=root;

max=ptr->data;

while(ptr !=NULL)

{

if(max < ptr->data)

{

max=ptr->data;

}

ptr=ptr->link;

}

return max;

}

int node\_length()

{

struct node \*temp;

int count=0;

temp=root;

while(temp!=NULL)

{

count++;

temp=temp->link;

}

return count;

}

void display()

{

struct node \*temp;

temp=root;

while(temp!=NULL)

{

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

temp=temp->link;

}

}

int main()

{

int length,small\_num,large\_num;

create\_node(1);

create\_node(1000);

create\_node(20);

create\_node(65);

create\_node(0);

display();

length=node\_length();

printf(“Length of List=%d\n”,length);

small\_num=list\_smallest\_num();

printf("Smallest number in Single List is=%d\n",small\_num);

large\_num=list\_largest\_num();

printf("Largest number in Single List is=%d\n",large\_num);

return 0;

}

1. W.A.P. to merge two linked list.

Ans:-

#include<stdio.h>

void create\_node\_one(int data);

void create\_node\_sec(int data);

struct node merge\_list(struct node \*head1 ,struct node \*head2);

struct node node\_length(struct node \*parent);

struct node display(struct node \*head);

struct node

{

int data;

struct node \*link;

};

struct node \*root1=NULL;

struct node \*root2=NULL;

void create\_node\_one(int data)

{

struct node \*temp;

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

temp -> data=data;

temp ->link=NULL;

if(root1==NULL)

{

root1=temp;

}

else

{

struct node \*ptr;

ptr=root1;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

}

void create\_node\_sec(int data)

{

struct node \*temp;

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

temp -> data=data;

temp ->link=NULL;

if(root2==NULL)

{

root2=temp;

}

else

{

struct node \*ptr;

ptr=root2;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

}

struct node merge\_list(struct node \*head1 ,struct node \*head2)

{

struct node \*temp;

temp=head1;

while(temp->link!=NULL)

{

temp=temp->link;

}

temp->link=head2;

}

struct node node\_length(struct node \*parent)

{

struct node \*temp;

int count=0;

temp=parent;

while(temp!=NULL)

{

count++;

temp=temp->link;

}

printf("Length of list=%d\n",count);

}

struct node display(struct node \*head)

{

struct node \*temp;

temp=head;

while(temp!=NULL)

{

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

temp=temp->link;

}

}

int main()

{

int length;

create\_node\_one(1000);

create\_node\_one(20);

create\_node\_one(65);

create\_node\_one(0);

display(root1);

node\_length(root1);

create\_node\_sec(45);

create\_node\_sec(55);

create\_node\_sec(65);

create\_node\_sec(75);

create\_node\_sec(85);

create\_node\_sec(95);

create\_node\_sec(25);

display(root2);

node\_length(root2);

merge\_list(root1,root2);

display(root1);

node\_length(root1);

return 0;

}

1. W.A.P. to merge two ascending sorted list

Ans:-

#include<stdio.h>

void create\_node\_one(int data);

void create\_node\_sec(int data);

struct node sort\_list\_ascending(struct node \*head);

struct node merge\_list(struct node \*head1 ,struct node \*head2);

struct node node\_length(struct node \*parent);

struct node display(struct node \*head);

struct node

{

int data;

struct node \*link;

};

struct node \*root1=NULL;

struct node \*root2=NULL;

void create\_node\_one(int data)

{

struct node \*temp;

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

temp -> data=data;

temp ->link=NULL;

if(root1==NULL)

{

root1=temp;

}

else

{

struct node \*ptr;

ptr=root1;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

}

void create\_node\_sec(int data)

{

struct node \*temp;

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

temp -> data=data;

temp ->link=NULL;

if(root2==NULL)

{

root2=temp;

}

else

{

struct node \*ptr;

ptr=root2;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

}

struct node sort\_list\_ascending(struct node \*head)

{

struct node \*ptr1;

struct node \*ptr2;

int temp;

ptr1=head;

while(ptr1->link!=NULL)

{

ptr2=ptr1->link;

while(ptr2!=NULL)

{

if(ptr1->data >ptr2->data)

{

temp=ptr1->data;

ptr1->data = ptr2->data;

ptr2->data=temp;

}

ptr2=ptr2->link;

}

ptr1=ptr1->link;

}

}

struct node merge\_list(struct node \*head1 ,struct node \*head2)

{

struct node \*temp;

temp=head1;

while(temp->link!=NULL)

{

temp=temp->link;

}

temp->link=head2;

}

struct node node\_length(struct node \*parent)

{

struct node \*temp;

int count=0;

temp=parent;

while(temp!=NULL)

{

count++;

temp=temp->link;

}

printf("Length of list=%d\n",count);

}

struct node display(struct node \*head)

{

struct node \*temp;

temp=head;

while(temp!=NULL)

{

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

temp=temp->link;

}

}

int main()

{

int length;

create\_node\_one(1);

create\_node\_one(1000);

create\_node\_one(20);

create\_node\_one(65);

create\_node\_one(0);

display(root1);

node\_length(root1);

create\_node\_sec(45);

create\_node\_sec(55);

create\_node\_sec(65);

create\_node\_sec(75);

create\_node\_sec(85);

create\_node\_sec(95);

create\_node\_sec(25);

display(root2);

node\_length(root2);

printf("\n");

printf("\n");

printf("Merge list is=\n");

merge\_list(root1,root2);

display(root1);

node\_length(root1);

printf("\n");

printf("\n");

printf("Ascending Sorted Merge list is=\n");

sort\_list\_ascending(root1);

display(root1);

node\_length(root1);

return 0;

}

1. W.A.P. to merge two descending order list.

Ans:-

#include<stdio.h>

void create\_node\_one(int data);

void create\_node\_sec(int data);

struct node sort\_list\_descending(struct node \*head);

struct node merge\_list(struct node \*head1 ,struct node \*head2);

struct node node\_length(struct node \*parent);

struct node display(struct node \*head);

struct node

{

int data;

struct node \*link;

};

struct node \*root1=NULL;

struct node \*root2=NULL;

void create\_node\_one(int data)

{

struct node \*temp;

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

temp -> data=data;

temp ->link=NULL;

if(root1==NULL)

{

root1=temp;

}

else

{

struct node \*ptr;

ptr=root1;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

}

void create\_node\_sec(int data)

{

struct node \*temp;

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

temp -> data=data;

temp ->link=NULL;

if(root2==NULL)

{

root2=temp;

}

else

{

struct node \*ptr;

ptr=root2;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

}

struct node sort\_list\_descending(struct node \*head)

{

struct node \*ptr1;

struct node \*ptr2;

int temp;

ptr1=head;

while(ptr1->link!=NULL)

{

ptr2=ptr1->link;

while(ptr2!=NULL)

{

if(ptr1->data < ptr2->data)

{

temp=ptr1->data;

ptr1->data = ptr2->data;

ptr2->data=temp;

}

ptr2=ptr2->link;

}

ptr1=ptr1->link;

}

}

struct node merge\_list(struct node \*head1 ,struct node \*head2)

{

struct node \*temp;

temp=head1;

while(temp->link!=NULL)

{

temp=temp->link;

}

temp->link=head2;

}

struct node node\_length(struct node \*parent)

{

struct node \*temp;

int count=0;

temp=parent;

while(temp!=NULL)

{

count++;

temp=temp->link;

}

printf("Length of list=%d\n",count);

}

struct node display(struct node \*head)

{

struct node \*temp;

temp=head;

while(temp!=NULL)

{

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

temp=temp->link;

}

}

int main()

{

int length;

create\_node\_one(1);

create\_node\_one(1000);

create\_node\_one(20);

create\_node\_one(65);

create\_node\_one(0);

display(root1);

node\_length(root1);

create\_node\_sec(45);

create\_node\_sec(55);

create\_node\_sec(65);

create\_node\_sec(75);

create\_node\_sec(85);

create\_node\_sec(95);

create\_node\_sec(25);

display(root2);

node\_length(root2);

printf("\n");

printf("\n");

printf("Merge list is=\n");

merge\_list(root1,root2);

display(root1);

node\_length(root1);

printf("\n");

printf("\n");

printf("Descending Sorted Merge list is=\n");

sort\_list\_descending(root1);

display(root1);

node\_length(root1);

return 0;

}

1. W.A.P. to swap two nodes in single linked list.

Ans:-

#include<stdio.h>

void create\_node(int data);

void swap\_node(int data1,int data2);

int node\_length(void);

void display(void);

struct node

{

int data;

struct node \*link;

};

struct node \*root=NULL;

void create\_node(int data)

{

struct node \*temp;

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

temp -> data=data;

temp ->link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

struct node \*ptr;

ptr=root;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

}

void swap\_node(int data1,int data2)

{

struct node \*ptr1=NULL;

struct node \*temp1=NULL;

struct node \*ptr2=NULL;

struct node \*temp2=NULL;

struct node \*temp3=NULL;

if(data1==data2)

{

printf("Both node values are same:\n");

}

temp1=root;

while(temp1 && temp1->data!=data1)

{

ptr1=temp1;

temp1=temp1->link;

}

temp2=root;

while(temp2 && temp2->data!=data2)

{

ptr2=temp2;

temp2=temp2->link;

}

if(temp1==NULL || temp2==NULL)

{

printf("Data is not present in list\n");

}

if(ptr1!=NULL)

{

ptr1->link=temp2;

}

else

{

root=temp2;

}

if(ptr2!=NULL)

{

ptr2->link=temp1;

}

else

{

root=temp1;

}

temp3=temp2->link;

temp2->link=temp1->link;

temp1->link=temp3;

}

int node\_length()

{

struct node \*temp;

int count=0;

temp=root;

while(temp!=NULL)

{

count++;

temp=temp->link;

}

return count;

}

void display()

{

struct node \*temp;

temp=root;

while(temp!=NULL)

{

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

temp=temp->link;

}

}

int main()

{

int length,x,y;

create\_node(20);

create\_node(50);

create\_node(70);

create\_node(90);

create\_node(100);

display();

length=node\_length();

printf("Length of Single List is=%d\n",length);

printf("Enter x=");

scanf("%d",&x);

printf("Enter y=");

scanf("%d",&y);

swap\_node(x,y);

display();

length=node\_length();

printf("Length of Single List is=%d\n",length);

return 0;

}

1. W.A.P. to find duplicates of number and its count in linked list.

Ans:-

#include<stdio.h>

void create\_list(int data);

int list\_length(void);

void display(void);

void reverse\_list(void);

struct node

{

int data;

struct node \*link;

};

struct node \*root=NULL;

void create\_list(int data)

{

struct node \*temp;

struct node \*ptr;

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

temp->data=data;

temp->link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

ptr=root;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

}

int list\_length(void)

{

int count=0;

struct node \*temp;

temp=root;

while(temp!=NULL)

{

count++;

temp=temp->link;

}

return count;

}

void display()

{

struct node \*temp=NULL;

temp=root;

while(temp!=NULL)

{

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

temp=temp->link;

}

}

void duplicates()

{

struct node \*ptr=NULL;

int temp;

int count=0;

while(root!=NULL)

{

ptr=root->link;

while(ptr->link!=NULL)

{

if(root->data == ptr->data)

{

temp=ptr->data;

printf("%d\n",temp);

count++;

}

ptr=ptr->link;

}

root=root->link;

}

printf("Total duplicates =%d \n",count);

}

int main()

{

int length;

create\_list(1);

create\_list(2);

create\_list(3);

create\_list(1);

create\_list(1);

create\_list(5);

create\_list(4);

create\_list(8);

create\_list(1);

create\_list(1);

display();

length=list\_length();

printf("Length of list=%d\n",length);

printf("Duplicates from list are:\n");

duplicates();

return 0;

}

1. W.A.P. to priintf specific node from linked list.

Ans:-

#include<stdio.h>

void create\_node(int data);

int node\_length(void);

void display\_list(void);

void display\_specific\_endside(int node\_num);

void display\_specific\_frontside(int node\_num);

struct node

{

int data;

struct node \*link;

};

struct node \*root=NULL;

void create\_node(int data)

{

struct node \*temp;

struct node \*ptr;

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

temp->data=data;

temp->link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

ptr=root;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

}

int node\_length(void)

{

struct node \*temp=NULL;

int count=0;

temp=root;

while(temp!=NULL)

{

count++;

temp=temp->link;

}

return count;

}

void display(void)

{

struct node \*temp=NULL;

temp=root;

while(temp!=NULL)

{

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

temp=temp->link;

}

}

void display\_specific\_endside(int node\_num)

{

struct node \*ptr=NULL;

ptr=root;

int len,temp,i;

len=node\_length();

temp=len - node\_num;

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

{

ptr=ptr->link;

}

printf("%d\n",ptr->data);

}

void display\_specific\_frontside(int node\_num)

{

struct node \*ptr=NULL;

int len,i;

len=node\_length();

if(node\_num==1)

{

printf("%d\n",root->data);

}

else

{

ptr=root;

for(i=2;i<=node\_num;i++)

{

ptr=ptr->link;

}

printf("%d\n",ptr->data);

}

}

int main()

{

printf("Size of struct node=%d\n\n",sizeof(struct node));

int length,node;

create\_node(20);

create\_node(30);

create\_node(40);

create\_node(50);

create\_node(60);

display();

length=node\_length();

printf("Length of list=%d\n",length);

POS1:printf("Enter node which is to be display from endside=");

scanf("%d",&node);

if(node > length)

{

printf("please enter node number berween 1 to %d\n",length);

goto POS1;

}

printf("Node at %d position from endside=",node);

display\_specific\_endside(node);

POS2:printf("Enter node which is to be display from frontside=");

scanf("%d",&node);

if(node > length)

{

printf("please enter node number berween 1 to %d\n",length);

goto POS2;

}

printf("Node at %d position from frontside=",node);

display\_specific\_frontside(node);

return 0;

}

1. W.A.P. to create string using linked list and reverse it.

Ans:-

#include<stdio.h>

void create\_list(char ch);

int list\_length(void);

void display(void);

void reverse\_list(void);

struct node

{

char ch;

struct node \*link;

};

struct node \*root=NULL;

void create\_list(char ch)

{

struct node \*temp;

struct node \*ptr;

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

temp->ch=ch;

temp->link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

ptr=root;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

}

int list\_length()

{

int count=0;

struct node \*temp;

temp=root;

while(temp!=NULL)

{

count++;

temp=temp->link;

}

return count;

}

void display()

{

struct node \*temp;

temp=root;

while(temp!=NULL)

{

printf("%c",temp->ch);

temp=temp->link;

}

}

void reverse\_list()

{

struct node \*next=NULL;

struct node \*prev=NULL;

struct node \*ptr=NULL;

ptr=root;

while(ptr->link!=NULL)

{

next=ptr->link;

ptr->link=prev;

prev=ptr;

ptr=next;

}

root=prev;

}

int main()

{

int length;

create\_list('P');

create\_list('R');

create\_list('A');

create\_list('S');

create\_list('A');

create\_list('D');

create\_list('\0');

display();

printf("\n");

length=list\_length();

printf("Length of String List=%d",length);

printf("\n");

reverse\_list();

display();

printf("\n");

length=list\_length();

printf("Length of String List=%d",length);

return 0;

}

1. W.A.P. to find middle of linked list.

Ans:-

#include<stdio.h>

void create\_node(int data);

void middle\_node(void);

int node\_length(void);

void display(void);

struct node

{

int data;

struct node \*link;

};

struct node \*root=NULL;

void create\_node(int data)

{

struct node \*temp;

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

temp -> data=data;

temp ->link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

struct node \*ptr;

ptr=root;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

}

void middle\_node(void)

{

struct node \*temp;

temp=root;

int length,middle,i=1;

length=node\_length();

middle=length/2;

while(i<middle)

{

temp=temp->link;

i++;

}

printf("Middle of list=%d\n",temp->data);

}

int node\_length()

{

struct node \*temp;

int count=0;

temp=root;

while(temp!=NULL)

{

count++;

temp=temp->link;

}

return count;

}

void display()

{

struct node \*temp;

temp=root;

while(temp!=NULL)

{

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

temp=temp->link;

}

}

int main()

{

int length;

create\_node(20);

create\_node(50);

create\_node(70);

create\_node(90);

create\_node(100);

create\_node(150);

display();

length=node\_length();

printf("Length of Single List is=%d\n",length);

middle\_node();

return 0;

}

1. W.A.P. to implement all operations of Double Linked List

Ans:-

#include<stdio.h>

void create\_list(int data);

void insert\_at\_begin(int data);

void insert\_at\_end(int data);

void insert\_at\_specific(int data,int location);

void delete\_begin(void);

void delete\_end(void);

void delete\_specific(void);

void delete\_list(void);

int list\_length(void);

void list\_display(void);

struct node

{

int data;

struct node \*l\_link;

struct node \*r\_link;

};

struct node \*root=NULL;

void create\_list(int data)

{

struct node \*temp;

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

temp->data=data;

temp->l\_link=NULL;

temp->r\_link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

struct node \*ptr;

ptr=root;

while(ptr->r\_link!=NULL)

{

ptr=ptr->r\_link;

}

ptr->r\_link=temp;

temp->l\_link=ptr;

}

}

void insert\_at\_begin(int data)

{

struct node \*temp;

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

temp->data=data;

temp->l\_link=NULL;

temp->r\_link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

temp->r\_link=root;

root->l\_link=temp;

root=temp;

}

}

void insert\_at\_end(int data)

{

struct node \*temp;

struct node \*ptr;

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

temp->data=data;

ptr=root;

while(ptr->r\_link!=NULL)

{

ptr=ptr->r\_link;

}

ptr->r\_link=temp;

temp->l\_link=ptr;

temp->r\_link=NULL;

}

void insert\_at\_specific(int data,int location)

{

struct node \*temp;

struct node \*ptr;

int len,i=1;

len=list\_length();

if(location>len)

{

printf("Location is not present in list\n");

}

else

{

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

temp->data=data;

temp->l\_link=NULL;

temp->r\_link=NULL;

ptr=root;

while(i<location-1)

{

ptr=ptr->r\_link;

i++;

}

temp->r\_link=ptr->r\_link;

ptr->r\_link->l\_link=temp;

temp->l\_link=ptr;

ptr->r\_link=temp;

}

}

void delete\_begin(void)

{

struct node \*ptr;

ptr=root;

root=root->r\_link;

root->l\_link=NULL;

free(ptr);

}

void delete\_end(void)

{

struct node \*ptr;

ptr = root;

while(ptr->r\_link!=NULL)

{

ptr = ptr ->r\_link;

}

ptr->l\_link->r\_link=NULL;

free(ptr);

}

void delete\_specific()

{

struct node \*ptr;

struct node \*temp = NULL;

struct node \*temp1 = NULL;

int data;

printf("Enter data to be deleted=");

scanf("%d",&data);

ptr=root;

if(root == NULL)

{

return NULL;

}

while(ptr->data != data)

{

if(ptr->r\_link == NULL)

{

return NULL;

}

else

{

temp=ptr;

ptr=ptr->r\_link;

}

}

if(ptr==root)

{

root=root->r\_link;

}

else

{

ptr->l\_link->r\_link = ptr->r\_link;

}

if(ptr==temp1)

{

temp1=ptr->l\_link;

}

else

{

ptr->r\_link->l\_link=ptr->l\_link;

}

}

void delete\_list()

{

struct node \*ptr;

if(root==NULL)

{

return NULL;

}

else

{

ptr=root;

while(ptr->r\_link!=NULL)

{

ptr=ptr->r\_link;

}

ptr->l\_link->r\_link=NULL;

root=ptr->l\_link->r\_link;

free(ptr);

}

}

int list\_length(void)

{

struct node \*temp;

int count=0;

temp=root;

while(temp!=NULL)

{

count++;

temp=temp->r\_link;

}

return count;

}

void list\_display(void)

{

struct node \*temp;

temp=root;

if(temp==NULL)

{

printf("List is Empty\n");

}

else

{

while(temp!=NULL)

{

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

temp=temp->r\_link;

}

}

}

int main()

{

int length,position;

create\_list(20);

create\_list(40);

create\_list(60);

create\_list(80);

create\_list(100);

printf("List Elements are:\n");

list\_display();

length=list\_length();

printf("Length of list=%d\n",length);

insert\_at\_begin(10);

printf("List Elements after add at beginning are:\n");

list\_display();

length=list\_length();

printf("Length of list=%d\n",length);

insert\_at\_end(120);

printf("List Elements after add at end are:\n");

list\_display();

length=list\_length();

printf("Length of list=%d\n",length);

printf("Enter location which is node to be inserted=");

scanf("%d",&position);

insert\_at\_specific(250,position);

printf("List After Inserting node at %d location\n",position);

list\_display();

length=list\_length();

printf("Length of list=%d\n",length);

delete\_begin();

printf("List Elements after deleting beginning node\n");

list\_display();

length=list\_length();

printf("Length of list=%d\n",length);

delete\_end();

printf("List Elements after deleting end node\n");

list\_display();

length=list\_length();

printf("Length of list=%d\n",length);

delete\_specific();

printf("List Elements after deleting specific node\n");

list\_display();

length=list\_length();

printf("Length of list=%d\n",length);

delete\_list();

printf("List Elements after deleting all nodes\n");

list\_display();

length=list\_length();

printf("Length of list=%d\n",length);

return 0;

}

1. W.A.P. to reverse double linked list.

Ans:-

#include<stdio.h>

void create\_list(int data);

void reverse\_list(void);

int list\_length(void);

void list\_display(void);

struct node

{

int data;

struct node \*l\_link;

struct node \*r\_link;

};

struct node \*root=NULL;

void create\_list(int data)

{

struct node \*temp;

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

temp->data=data;

temp->l\_link=NULL;

temp->r\_link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

struct node \*ptr;

ptr=root;

while(ptr->r\_link!=NULL)

{

ptr=ptr->r\_link;

}

ptr->r\_link=temp;

temp->l\_link=ptr;

}

}

void reverse\_list()

{

struct node \*ptr=NULL;

struct node \*temp=NULL;

temp=root;

while(temp!=NULL)

{

ptr=temp->l\_link;

temp->l\_link=temp->r\_link;

temp->r\_link=ptr;

temp=temp->l\_link;

}

if(ptr!=NULL)

{

root=ptr->l\_link;

}

}

int list\_length(void)

{

struct node \*temp;

int count=0;

temp=root;

while(temp!=NULL)

{

count++;

temp=temp->r\_link;

}

return count;

}

void list\_display(void)

{

struct node \*temp;

temp=root;

if(temp==NULL)

{

printf("List is Empty\n");

}

else

{

while(temp!=NULL)

{

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

temp=temp->r\_link;

}

}

}

int main()

{

int length;

create\_list(20);

create\_list(40);

create\_list(60);

create\_list(80);

create\_list(100);

printf("List Elements are:\n");

list\_display();

length=list\_length();

printf("Length of list=%d\n",length);

reverse\_list();

printf("List Elements after reversing:\n");

list\_display();

length=list\_length();

printf("Length of list=%d\n",length);

return 0;

}

1. W.A.P. to sort double linked list.

Ans:-

#include<stdio.h>

void create\_list(int data);

void sort\_list\_ascending(void);

void sort\_list\_descending(void);

int list\_length(void);

void list\_display(void);

struct node

{

int data;

struct node \*l\_link;

struct node \*r\_link;

};

struct node \*root=NULL;

void create\_list(int data)

{

struct node \*temp;

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

temp->data=data;

temp->l\_link=NULL;

temp->r\_link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

struct node \*ptr;

ptr=root;

while(ptr->r\_link!=NULL)

{

ptr=ptr->r\_link;

}

ptr->r\_link=temp;

temp->l\_link=ptr;

}

}

void sort\_list\_ascending(void)

{

struct node \*ptr1=NULL;

struct node \*ptr2=NULL;

int temp;

ptr1=root;

while(ptr1->r\_link!=NULL)

{

ptr2=ptr1->r\_link;

while(ptr2!=NULL)

{

if(ptr1->data > ptr2->data)

{

temp=ptr1->data;

ptr1->data=ptr2->data;

ptr2->data=temp;

}

ptr2=ptr2->r\_link;

}

ptr1=ptr1->r\_link;

}

}

void sort\_list\_descending(void)

{

struct node \*ptr1=NULL;

struct node \*ptr2=NULL;

int temp;

ptr1=root;

while(ptr1->r\_link!=NULL)

{

ptr2=ptr1->r\_link;

while(ptr2!=NULL)

{

if(ptr1->data < ptr2->data)

{

temp=ptr1->data;

ptr1->data=ptr2->data;

ptr2->data=temp;

}

ptr2=ptr2->r\_link;

}

ptr1=ptr1->r\_link;

}

}

int list\_length(void)

{

struct node \*temp;

int count=0;

temp=root;

while(temp!=NULL)

{

count++;

temp=temp->r\_link;

}

return count;

}

void list\_display(void)

{

struct node \*temp;

temp=root;

if(temp==NULL)

{

printf("List is Empty\n");

}

else

{

while(temp!=NULL)

{

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

temp=temp->r\_link;

}

}

}

int main()

{

int length;

create\_list(500);

create\_list(100);

create\_list(0);

create\_list(01);

create\_list(200);

printf("List Elements are:\n");

list\_display();

length=list\_length();

printf("Length of list=%d\n",length);

sort\_list\_ascending();

printf("List Elements after ascending sort are:\n");

list\_display();

sort\_list\_descending();

printf("List Elements after descending sort are:\n");

list\_display();

return 0;

}

1. W.A.P. to find largest & smallest number in Double Linked List.

Ans:-

#include<stdio.h>

void create\_list(int data);

int largest\_num();

int smallest\_num();

int list\_length(void);

void list\_display(void);

struct node

{

int data;

struct node \*l\_link;

struct node \*r\_link;

};

struct node \*root=NULL;

void create\_list(int data)

{

struct node \*temp;

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

temp->data=data;

temp->l\_link=NULL;

temp->r\_link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

struct node \*ptr;

ptr=root;

while(ptr->r\_link!=NULL)

{

ptr=ptr->r\_link;

}

ptr->r\_link=temp;

temp->l\_link=ptr;

}

}

int largest\_num()

{

struct node \*ptr;

int max;

ptr=root;

max=ptr->data;

while(ptr->r\_link!=NULL)

{

if(max<ptr->data)

{

max=ptr->data;

}

ptr=ptr->r\_link;

}

return max;

}

int smallest\_num()

{

struct node \*ptr;

int min;

ptr=root;

min=ptr->data;

while(ptr->r\_link!=NULL)

{

if(min>ptr->data)

{

min=ptr->data;

}

ptr=ptr->r\_link;

}

return min;

}

int list\_length(void)

{

struct node \*temp;

int count=0;

temp=root;

while(temp!=NULL)

{

count++;

temp=temp->r\_link;

}

return count;

}

void list\_display(void)

{

struct node \*temp;

temp=root;

if(temp==NULL)

{

printf("List is Empty\n");

}

else

{

while(temp!=NULL)

{

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

temp=temp->r\_link;

}

}

}

int main()

{

int length,small\_num,large\_num;

create\_list(500);

create\_list(100);

create\_list(0);

create\_list(01);

create\_list(200);

printf("List Elements are:\n");

list\_display();

length=list\_length();

printf("Length of list=%d\n",length);

small\_num=smallest\_num();

printf("Smallest number in Double List is=%d\n",small\_num);

large\_num=largest\_num();

printf("Largest number in Double List is=%d\n",large\_num);

return 0;

}

1. W.A.P. to merge tow unsorted Double Linked List

Ans:-

#include<stdio.h>

void create\_list1(int data);

void create\_list2(int data);

struct node merge\_list(struct node\*head1 , struct node \*head2);

struct node list\_length(struct node\*head);

struct node list\_display(struct node\*head);

struct node

{

int data;

struct node \*l\_link;

struct node \*r\_link;

};

struct node \*root1=NULL;

struct node \*root2=NULL;

void create\_list1(int data)

{

struct node \*temp;

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

temp->data=data;

temp->l\_link=NULL;

temp->r\_link=NULL;

if(root1==NULL)

{

root1=temp;

}

else

{

struct node \*ptr;

ptr=root1;

while(ptr->r\_link!=NULL)

{

ptr=ptr->r\_link;

}

ptr->r\_link=temp;

temp->l\_link=ptr;

}

}

void create\_list2(int data)

{

struct node \*temp;

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

temp->data=data;

temp->l\_link=NULL;

temp->r\_link=NULL;

if(root2==NULL)

{

root2=temp;

}

else

{

struct node \*ptr;

ptr=root2;

while(ptr->r\_link!=NULL)

{

ptr=ptr->r\_link;

}

ptr->r\_link=temp;

temp->l\_link=ptr;

}

}

struct node merge\_list(struct node\*head1 , struct node \*head2)

{

struct node \*ptr;

ptr=head1;

while(ptr->r\_link!=NULL)

{

ptr=ptr->r\_link;

}

ptr->r\_link=head2;

}

struct node list\_length(struct node\*head)

{

struct node \*temp;

int count=0;

temp=head;

while(temp!=NULL)

{

count++;

temp=temp->r\_link;

}

printf("Length of list=%d",count);

}

struct node list\_display(struct node\*head)

{

struct node \*temp;

temp=head;

if(temp==NULL)

{

printf("List is Empty\n");

}

else

{

while(temp!=NULL)

{

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

temp=temp->r\_link;

}

}

}

int main()

{

create\_list1(50);

create\_list1(45);

create\_list1(40);

create\_list1(10);

create\_list1(5);

printf("List Elements are:\n");

list\_display(root1);

list\_length(root1);

create\_list2(65);

create\_list2(80);

create\_list2(78);

create\_list2(91);

create\_list2(56);

printf("List Elements are:\n");

list\_display(root2);

list\_length(root2);

merge\_list(root1,root2);

printf("List After Merging:\n");

list\_display(root1);

list\_length(root1);

return 0;

}

1. W.A.P. to merge two ascending sorted double linkedlist.

Ans:-

#include<stdio.h>

void create\_list1(int data);

void create\_list2(int data);

struct node sort\_list\_ascending(struct node \*head);

struct node merge\_list(struct node\*head1 , struct node \*head2);

struct node list\_length(struct node\*head);

struct node list\_display(struct node\*head);

struct node

{

int data;

struct node \*l\_link;

struct node \*r\_link;

};

struct node \*root1=NULL;

struct node \*root2=NULL;

void create\_list1(int data)

{

struct node \*temp;

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

temp->data=data;

temp->l\_link=NULL;

temp->r\_link=NULL;

if(root1==NULL)

{

root1=temp;

}

else

{

struct node \*ptr;

ptr=root1;

while(ptr->r\_link!=NULL)

{

ptr=ptr->r\_link;

}

ptr->r\_link=temp;

temp->l\_link=ptr;

}

}

void create\_list2(int data)

{

struct node \*temp;

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

temp->data=data;

temp->l\_link=NULL;

temp->r\_link=NULL;

if(root2==NULL)

{

root2=temp;

}

else

{

struct node \*ptr;

ptr=root2;

while(ptr->r\_link!=NULL)

{

ptr=ptr->r\_link;

}

ptr->r\_link=temp;

temp->l\_link=ptr;

}

}

struct node sort\_list\_ascending(struct node \*head)

{

struct node \*ptr1=NULL;

struct node \*ptr2=NULL;

int temp;

ptr1=head;

while(ptr1->r\_link!=NULL)

{

ptr2=ptr1->r\_link;

while(ptr2!=NULL)

{

if(ptr1->data > ptr2->data)

{

temp=ptr1->data;

ptr1->data=ptr2->data;

ptr2->data=temp;

}

ptr2=ptr2->r\_link;

}

ptr1=ptr1->r\_link;

}

}

struct node merge\_list(struct node\*head1 , struct node \*head2)

{

struct node \*ptr;

ptr=head1;

while(ptr->r\_link!=NULL)

{

ptr=ptr->r\_link;

}

ptr->r\_link=head2;

}

struct node list\_length(struct node\*head)

{

struct node \*temp;

int count=0;

temp=head;

while(temp!=NULL)

{

count++;

temp=temp->r\_link;

}

printf("Length of list=%d\n",count);

}

struct node list\_display(struct node\*head)

{

struct node \*temp;

temp=head;

if(temp==NULL)

{

printf("List is Empty\n");

}

else

{

while(temp!=NULL)

{

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

temp=temp->r\_link;

}

}

}

int main()

{

create\_list1(50);

create\_list1(45);

create\_list1(40);

create\_list1(10);

create\_list1(5);

printf("1st List Elements are:\n");

list\_display(root1);

list\_length(root1);

create\_list2(65);

create\_list2(80);

create\_list2(78);

create\_list2(91);

create\_list2(56);

printf("2nd List Elements are:\n");

list\_display(root2);

list\_length(root2);

merge\_list(root1,root2);

printf("Merge Two List");

list\_display(root1);

list\_length(root1);

sort\_list\_ascending(root1);

printf("List After Merging two sorted list in ascending order:\n");

list\_display(root1);

list\_length(root1);

return 0;

}

1. W.A.P. to merge two descending sorted double linked list.

Ans:-

#include<stdio.h>

void create\_list1(int data);

void create\_list2(int data);

struct node sort\_list\_descending(struct node \*head);

struct node merge\_list(struct node\*head1 , struct node \*head2);

struct node list\_length(struct node\*head);

struct node list\_display(struct node\*head);

struct node

{

int data;

struct node \*l\_link;

struct node \*r\_link;

};

struct node \*root1=NULL;

struct node \*root2=NULL;

void create\_list1(int data)

{

struct node \*temp;

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

temp->data=data;

temp->l\_link=NULL;

temp->r\_link=NULL;

if(root1==NULL)

{

root1=temp;

}

else

{

struct node \*ptr;

ptr=root1;

while(ptr->r\_link!=NULL)

{

ptr=ptr->r\_link;

}

ptr->r\_link=temp;

temp->l\_link=ptr;

}

}

void create\_list2(int data)

{

struct node \*temp;

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

temp->data=data;

temp->l\_link=NULL;

temp->r\_link=NULL;

if(root2==NULL)

{

root2=temp;

}

else

{

struct node \*ptr;

ptr=root2;

while(ptr->r\_link!=NULL)

{

ptr=ptr->r\_link;

}

ptr->r\_link=temp;

temp->l\_link=ptr;

}

}

struct node sort\_list\_descending(struct node \*head)

{

struct node \*ptr1=NULL;

struct node \*ptr2=NULL;

int temp;

ptr1=head;

while(ptr1->r\_link!=NULL)

{

ptr2=ptr1->r\_link;

while(ptr2!=NULL)

{

if(ptr1->data < ptr2->data)

{

temp=ptr1->data;

ptr1->data=ptr2->data;

ptr2->data=temp;

}

ptr2=ptr2->r\_link;

}

ptr1=ptr1->r\_link;

}

}

struct node merge\_list(struct node\*head1 , struct node \*head2)

{

struct node \*ptr;

ptr=head1;

while(ptr->r\_link!=NULL)

{

ptr=ptr->r\_link;

}

ptr->r\_link=head2;

}

struct node list\_length(struct node\*head)

{

struct node \*temp;

int count=0;

temp=head;

while(temp!=NULL)

{

count++;

temp=temp->r\_link;

}

printf("Length of list=%d\n",count);

}

struct node list\_display(struct node\*head)

{

struct node \*temp;

temp=head;

if(temp==NULL)

{

printf("List is Empty\n");

}

else

{

while(temp!=NULL)

{

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

temp=temp->r\_link;

}

}

}

int main()

{

create\_list1(50);

create\_list1(45);

create\_list1(40);

create\_list1(10);

create\_list1(5);

printf("1st List Elements are:\n");

list\_display(root1);

list\_length(root1);

create\_list2(65);

create\_list2(80);

create\_list2(78);

create\_list2(91);

create\_list2(56);

printf("2nd List Elements are:\n");

list\_display(root2);

list\_length(root2);

merge\_list(root1,root2);

printf("Merge List:\n");

list\_display(root1);

list\_length(root1);

sort\_list\_descending(root1);

printf("List After Merging two sorted list in descending order:\n");

list\_display(root1);

list\_length(root1);

return 0;

}

1. W.A.P. to swap data of two node in double linked list.

Ans:-

#include<stdio.h>

void create\_list(int data);

void swap\_node(int data1,int data2);

int list\_length(void);

void list\_display(void);

struct node

{

int data;

struct node \*l\_link;

struct node \*r\_link;

};

struct node \*root=NULL;

void create\_list(int data)

{

struct node \*temp;

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

temp->data=data;

temp->l\_link=NULL;

temp->r\_link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

struct node \*ptr;

ptr=root;

while(ptr->r\_link!=NULL)

{

ptr=ptr->r\_link;

}

ptr->r\_link=temp;

temp->l\_link=ptr;

}

}

void swap\_node(int data1,int data2)

{

struct node \*ptr1=NULL;

struct node \*temp1=NULL;

struct node \*ptr2=NULL;

struct node \*temp2=NULL;

struct node \*temp3=NULL;

if(data1==data2)

{

printf("Both node values are same:\n");

}

temp1=root;

while(temp1 && temp1->data!=data1)

{

ptr1=temp1;

temp1=temp1->r\_link;

}

temp2=root;

while(temp2 && temp2->data!=data2)

{

ptr2=temp2;

temp2=temp2->r\_link;

}

if(temp1==NULL || temp2==NULL)

{

printf("Data is not present in list\n");

}

if(ptr1!=NULL)

{

ptr1->r\_link=temp2;

}

else

{

root=temp2;

}

if(ptr2!=NULL)

{

ptr2->r\_link=temp1;

}

else

{

root=temp1;

}

temp3=temp2->r\_link;

temp2->r\_link=temp1->r\_link;

temp1->r\_link=temp3;

}

int list\_length(void)

{

struct node \*temp;

int count=0;

temp=root;

while(temp!=NULL)

{

count++;

temp=temp->r\_link;

}

return count;

}

void list\_display(void)

{

struct node \*temp;

temp=root;

if(temp==NULL)

{

printf("List is Empty\n");

}

else

{

while(temp!=NULL)

{

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

temp=temp->r\_link;

}

}

}

int main()

{

int length,val1,val2;

create\_list(20);

create\_list(40);

create\_list(60);

create\_list(80);

create\_list(100);

printf("List Elements are:\n");

list\_display();

length=list\_length();

printf("Length of list=%d\n",length);

printf("Enter val1=");

scanf("%d",&val1);

printf("Enter val2=");

scanf("%d",&val2);

swap\_node(val1,val2);

printf("List Elements after swapping data:\n");

list\_display();

length=list\_length();

printf("Length of list=%d\n",length);

return 0;

}