Week 1 :

Q1 Given an array of integers, , write an algorithm and a program to left rotate the array by specific number of elements .

**Source Code**

#include<stdio.h>

#include<stdlib.h>

void rotate(int arr[],int n , int k){

  int temp[n];

    k=k%n;

  for(int i=0;i<k;i++){

    temp[i]=arr[i];

  }

  for(int i=0;i<n-k;i++){

    arr[i]=arr[i+k];

  }

  for(int i=0;i<k;i++){

    arr[n-k+i]=temp[i];

  }

}

int main(){

  int t;

  scanf("%d",&t);

 for(int i=t;i>0;i--){

    int n , k ;

    scanf("%d",&n);

    int arr[n];

    for(int i=0;i<n;i++){

      scanf("%d ",&arr[i]);

    }

    scanf("%d",&k);

  rotate(arr,n,k);

  for(int i=0;i<n;i++){

    printf("%d ",arr[i]);

  }

  printf("\n\n");

}

return 0;

}

**Output :**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc Akshat.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

3

5

2 4 -3 2 3

3

2 3 2 4 -3

10

1 6 3 2 9 1 4 2 3 6

4

9 1 4 2 3 6 1 6 3 2

15

-2 8 7 1 2 6 8 9 0 2 -6 2 9 7 1

6

8 9 0 2 -6 2 9 7 1 -2 8 7 1 2 6

Week 1:

Q2-Given an unsorted array of integers and two numbers a and b . Design an algorithm and write a program to find minimum distance between a and b in this array . Minimum distance between any two numbers a and b present in array is the minimum difference between their indices .

**Source Code**

#include<stdio.h>

#include<stdlib.h>

#include<limits.h>

int minimum(int arr[],int a , int b , int n){

  int min\_dist=INT\_MAX;

  int m=-1;

  int s=-1;

  for(int i=0;i<n;i++){

    if(arr[i]==a){

      m=i;

    if(s!=-1){

      int dist=abs(m-s);

      if(min\_dist>dist){

        min\_dist=dist;

      }

      }

    }

    if(arr[i]==b){

      s=i;

      if(m!=-1){

        int dist=abs(s-m);

        if(min\_dist>dist){

          min\_dist=dist;

        }

      }

    }

  }

  return min\_dist;

}

int main(){

  int t;

  scanf("%d",&t);

  for(int i=t;i>0;i++){

    int n;

    scanf("%d",&n);

    int arr[n];

    for(int i=0;i<n;i++){

      scanf("%d",&arr[i]);

    }

    int a , b;

    scanf("%d %d", &a,&b);

    int dis=minimum(arr,a,b,n);

    printf("%d\n\n",dis);

  }

  return 0;

}

**Output:**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc Akshat.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

3

5

2 4 -3 2 3

4 2

1

10

1 6 3 2 9 1 4 3 6 2

3 2

1

15

-2 8 7 1 2 6 8 9 0 2 -6 12 9 7 1

7 2

2

Week 1:

Q3-Given an array of nonnegative integers , where all numbers occur even number of times except one number which occurs odd number of times . Write an algorithm and a program to find this number. (Time complexity =O(n))

**Source Code**

#include<stdio.h>

#include<stdlib.h>

#include<limits.h>

int odd(int arr[],int n){

int res=0;

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

res=res^arr[i];

}

return res;

}

int main(){

int t;

scanf("%d",&t);

for(int i=t;i>0;i--){

int n;

scanf("%d",&n);

int arr[n];

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

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

}

int odd\_num=odd(arr,n);

printf("%d\n\n",odd\_num);

}

return 0;

}

**Output:**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc Akshat.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

3

5

2 4 3 2 3

4

11

1 6 3 2 4 1 4 2 3 6 6

6

15

2 8 7 1 2 6 8 9 0 2 6 2 9 7 1

0

Week 2:

Q1-Given a matrix of size n X n , where every row and every column is sorted in increasing order . Write an algorithm and a program to find whether the given key element is present in the matrix or not . (Linear time complexity)

**Source Code**

#include<stdio.h>

#include<stdlib.h>

int search(int n, int k , int arr[][n]){

int i=0,j=n-1;

while(i<n && j>=0){

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

return 1;

}

else if (arr[i][j]>k){

j--;

}

else{

i++;

}

}

return 0;

}

int main(){

int n;

scanf("%d",&n);

int arr[n][n];

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

for(int j=0;j<n;j++){

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

}

}

int k;

scanf("%d",&k);

int result=search(n,k,arr);

if(result==1){

printf("present \n");

}

else {

printf("not present");

}

return 0;

}

**Output :**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc Akshat.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

4

10 20 30 40

15 25 34 41

27 29 35 45

32 33 38 49

33

present

Week 2:

Q2- Given a boolean matrix (contains only 0 and 1 ) of size m X n where each row is sorted, write an algorithm and a program to find which row has maximum number of 1’s . (Linear time complexity)

**Source Code**

#include<stdio.h>

int row(int m , int n , int arr[][n]){

int ans=-1,i=0,j=n-1;

while(i<m && j>=0){

if(arr[i][j]==0){

i++;

}

else{

ans=i;

j--;

}

}

return ans;

}

int main(){

int m,n;

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

int arr[m][n];

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

for(int j=0;j<n;j++){

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

}

}

int res=row(m,n,arr);

if(res==-1){

printf("Not present \n");

}

else{

printf("row-%d\n",res+1);

}

return 0;

}

**Output :**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc Akshat.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

4 3

0 1 1

0 0 1

1 1 1

0 0 0

row-3

Week 3:

Q1-Design an algorithm and a program to implement stack using array .

**Source Code**

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

#define MAX 10

void push(int stk[] , int\* top,int item){

// check for the overflow condittion

if(\*top==MAX-1){

printf("Stack overflow");

}

else{

\*top+=1;

stk[\*top]=item;

}

}

void pop(int stk[], int\* top){

if(\*top==-1){

printf("stack underflow");

}

else{

printf("the value to be popped is %d",stk[\*top]);

\*top-=1;

}

}

void size (int stk[], int\* top){

printf("Size=%d\n",\*top+1);

}

void traverse(int stk[],int top){

if(top==-1){

printf("stack is empty");

}

else{

printf("Stack-");

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

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

}

printf("\n");

}

}

void peek(int stk[], int\* top){

if(\*top==-1){

printf("stack is empty");

}

else{

printf("%d",stk[\*top]);

}

}

int main(){

int stk[MAX], top=-1 , item , choice ;

printf("Press:\n1 to push\n2 to pop\n3 to calculate size \n4 to traverse \n5 to peek \n6 to exit\n");

do{

scanf("%d",&choice);

switch(choice)

{

case 1:

scanf("%d",&item);

push(stk,&top,item);

traverse(stk,top);

break;

case 2:

pop(stk,&top);

break;

case 3:

size(stk,&top);

break;

case 4:

traverse(stk,top);

break;

case 5:

peek(stk,&top);

break;

case 6:

exit(0);

default:

printf("Invalid choice\n");

}

}while(1);

return 0;

}

**Output:**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc stack.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

Press:

1 to push

2 to pop

3 to calculate size

4 to traverse

5 to peek

6 to exit

1

34

Stack- 34

1

42

Stack- 34 42

1

6

Stack- 34 42 6

3

Size=3

6

Week 3:

Q2-Given an expression string consisting of opening and closing brackets “{“,”}”,”(“,”)”,”[“,”]”, design an algorithm and a program to check whether this expression has balanced paranthesis or not .

**Source Code**

#include <stdio.h>

#include <stdlib.h>

#include<string.h>

#include <limits.h>

#define MAX 100

void push(char stk[] , int\* top,char item){

if(\*top==MAX-1){

printf("Stack overflow");

}

else{

\*top+=1;

stk[\*top]=item;

}

}

char pop(char stk[], int\* top){

if(\*top==-1){

printf("stack underflow");

}

else{

char item=stk[\*top];

\*top-=1;

return item;

}

}

int isopening(char ch){

return (ch=='(' || ch=='{' || ch=='[');

}

int isclosing(char ch){

return (ch==')' || ch=='}' || ch==']');

}

int ismatching( char open , char close){

return ((open=='(' && close==')') || (open=='{' && close=='}') || (open=='[' && close==']' ));

}

int isBalanced(char expr[]){

char stk[MAX];

int top=-1;

for(int i=0;i< strlen(expr);i++){

if(isopening(expr[i])){

push(stk,&top,expr[i]);

}else if(isclosing(expr[i])){

if(top==-1 || !ismatching(pop(stk,&top),expr[i])){

return 0;

}

}

}

return (top==-1);

}

int main(){

char expr[MAX], top=-1 , item ;

int t;

scanf("%d",&t);

for(int i=t;i>0;i--){

scanf("%s",expr);

if (isBalanced(expr)) {

printf("Balanced\n");

} else {

printf("Not Balanced\n");

}

}

return 0;

}

**Output :**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc stack.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

3

{{(()())}}

Balanced

([][])(){(())}

Balanced

{()(()}

Not Balanced

Week 3:

Q3-Given a string of opening and closing paranthesis , find the length of the longest valid paranthesis.

**Source Code**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#define MAX 100

int checklength(char expr[]){

    char stk[MAX];

    int top=-1,max=0;

 stk[++top]=-1;

 for(int i=0;i<strlen(expr);i++){

    if(expr[i]=='('){

        stk[++top]=i;

    }

    else if(expr[i]==')'){

        top--;

        if(top==-1){

            stk[++top]=i;

        }

        else {

            int len=i-stk[top];

            if(len>max){

                max=len;

            }

        }

    }

 }

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

}

int main(){

    char expr[MAX];

    int max=0;

    int t;

    scanf("%d",&t);

    for(int i=t;i>0;i--){

        scanf("%s",expr);

        checklength(expr);

    }

    return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc trees.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

3

()()))))

4

((()())(

6

(()()(()))()

12

Week 4:

Q1-Given an empty stack , design an algorithm and a program to reverse a string using this stack .

**Source Code**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MAX 100

void push(char stk[],int \*top, char item){

if(\*top==MAX-1){

printf("Stack overflow \n");

}

else {

\*top+=1;

stk[\*top]=item;

}

}

char pop(char stk[],int\* top){

if(\*top==-1){

printf("Stack underflow");

}

else{

char item=stk[\*top];

\*top-=1;

return item;

}

}

void reverse(char str[]){

int n=strlen(str);

char stk[MAX];

int top=-1;

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

push(stk,&top,str[i]);

}

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

str[i]=pop(stk,&top);

}

}

int main(){

char str[MAX];

int t;

scanf("%d",&t);

for(int i=t;i>0;i--){

scanf("%s",str);

reverse(str);

printf("%s\n", str);

}

return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc stack.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

3

qwerty

ytrewq

computerscience

ecneicsretupmoc

graphicerauniversity

ytisrevinuarecihparg

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX 100

void reverse(char str[], int start, int end) {

if (start >= end) {

return;

}

// Swap the characters at start and end

char temp = str[start];

str[start] = str[end];

str[end] = temp;

// Recur for the remaining substring

reverse(str, start + 1, end - 1);

}

int main() {

char str[MAX];

int t;

scanf("%d", &t);

for (int i = t; i > 0; i--) {

scanf("%s", str);

reverse(str, 0, strlen(str) - 1);

printf("%s\n", str);

}

return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc stack.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

3

qwerty

ytrewq

computerscience

ecneicsretupmoc

graphicerauniversity

ytisrevinuarecihparg

Week 4:

Q2-Design an algorithm and a program to implement two stack using one array

**Source Code**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<limits.h>

#define MAX 100

int arr[MAX];

int top1=-1;

int top2=MAX;

void push1(int item){

    if(top1<top2-1){

        top1+=1;

        arr[top1]=item;

    }

    else{

        printf("stack overflow\n");

    }

}

void push2(int item){

    if(top1<top2-1){

        top2-=1;

        arr[top2]=item;

    }

    else{

        printf("stack overflow\n");

    }

}

void pop1(){

    if(top1>=0){

        top1-=1;

    }

    else{

        printf("stack1 is empty\n");

    }

}

void pop2(){

    if(top2<MAX){

        top2+=1;

    }

    else{

        printf("stack 2 is empty\n");

    }

}

void traverse1(){

   if(top1==-1){

    printf("stack is empty\n");

   }

   else{

    printf("stack1 - ");

    for(int i=0;i<=top1;i++){

        printf(" %d",arr[i]);

    }

    printf("\n");

   }

}

void traverse2(){

   if(top2==MAX){

    printf("stack is emnpty\n");

   }

   else{

    printf("stack2 - ");

     for(int i=MAX-1;i>=top2;i--){

        printf(" %d",arr[i]);

    }

    printf("\n");

   }

}

int main(){

    int choice,item;

    printf("Press: \n1 to push in stack1 \n2 to pop from stack1\n3 to push in stack2\n4 to pop from stack 2\n5 to exit\n");

   do{

     scanf("%d",&choice);

     switch(choice){

        case 1:

        scanf("%d",&item);

        push1(item);

        traverse1();

        break;

        case 2:

        pop1();

        traverse1();

        break;

        case 3:

        scanf("%d",&item);

        push2(item);

        traverse2();

        break;

        case 4:

        pop2();

        traverse2();

        break;

        case 5:

        exit(0);

        default:

         printf("invalid choice\n");

     }

   }while(1);

   return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc practice.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

Press:

1 to push in stack1

2 to pop from stack1

3 to push in stack2

4 to pop from stack 2

5 to exit

1

342

stack1 - 342

1

564

stack1 - 342 564

3

601

stack2 - 601

3

970

stack2 - 601 970

3

123

stack2 - 601 970 123

5

Week 4:

Q3-Given an expression in the form of postfix representation , design an algorithm and a program to find the result of this expression .

**Source Code**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MAX 100

int eval(int a,int b ,char ch){

    switch(ch){

        case '+' : return a+b;

        case '-' : return a-b;

        case '\*' : return a\*b;

        case '/' : return a/b;

    }

    return 0;

}

void push(int stk[],int\* top,int item){

    if(\*top==MAX-1){

        printf("stack overflow\n");

    }

    else{

        \*top+=1;

        stk[\*top]=item;

    }

}

int pop(int stk[],int\* top){

    if(\*top==-1){

        printf("Stack is empty\n");

    }

    else{

        int item=stk[\*top];

        \*top-=1;

        return item;

    }

}

int main(){

    int t;

    scanf("%d",&t);

    getchar();

    int stk[MAX];

    char s[MAX];

    int top=-1;

    for(int i=t;i>0;i--){

    gets(s);

    for(int i=0;i<strlen(s);i++){

        if(s[i]>='0'&&s[i]<='9'){

            push(stk,&top,s[i]-'0');

        }

        else{

            int b=pop(stk,&top);

            int a=pop(stk,&top);

            int result=eval(a,b,s[i]);

            push(stk,&top,result);\

        }

    }

    printf("The result is %d\n",pop(stk,&top)); }

    return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc trees.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

3

231\*+4-

The result is 1

231\*+4+

The result is 9

231\*+

The result is 5

Week 5:

Q1-Design an algorithm and a program to implement queue using array

**Source Code**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#define MAX 100

void enque(int queue[] , int\* front , int\*rear , int item){

if(\*rear==MAX-1){

printf("queue is full i.e. overflow condition\n");

}

else if(\*rear==-1){

\*front=\*rear=0;

queue[\*rear]=item;

}

else{

\*rear+=1;

queue[\*rear]=item;

}

}

void dequeue(int queue[] , int\* front , int\* rear ){

if(\*front==-1){

printf("Queue is empty i.e. underflow condition\n");

}

else if(\*front==\*rear){

printf("Element to be dequeued is %d\n",queue[\*front]);

\*front=-1;

\*rear=-1;

return;

}

else{

printf("Element to be dequeued is %d\n",queue[\*front]);

\*front+=1;

}

}

void traverse(int queue[], int\* front , int\* rear){

if(\*front==-1){

printf("Queue underflow\n");

}

else{ printf("Queue - ");

for(int i=\*front;i<=\*rear;i++){

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

}

printf("\n");

}

}

int isempty(int queue[] , int\* front , int\*rear){

if(\*front==-1 && \*rear==-1){

printf("queue is empty\n");

}

else{

printf("queue is not empty\n");

}

}

void size(int queue[], int\* front, int\* rear) {

if (\*front == -1) {

printf("Size = 0\n");

} else {

printf("Size = %d\n", \*rear - \*front + 1);

}

}

int main(){

int queue[MAX];

int front=-1,rear=-1,item;

int choice;

printf("Press: \n1 to enqueue\n2 to dequeue\n3 to calculate size\n4 to exit\n");

do{

scanf("%d",&choice);

switch(choice){

case 1:

scanf("%d",&item);

enque(queue,&front,&rear,item);

traverse(queue,&front,&rear);

break;

case 2:

dequeue(queue,&front,&rear);

break;

case 3:

traverse(queue,&front,&rear);

break;

case 4:

size(queue,&front,&rear);

break;

case 5:

exit(0);

default:

printf("invalid choice\n");

}

}while(1);

return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc practice.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

Press:

1 to enqueue

2 to dequeue

3 to calculate size

4 to exit

1

34

Queue - 34

1

42

Queue - 34 42

1

6

Queue - 34 42 6

2

Element to be dequeued is 34

3

Queue - 42 6

4

Size = 2

5

Week 5:

Q2- Design an algorithm and write a program to reverse a queue

**Source Code**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

void enqueue(int queue[],int\* front,int\* rear,int item){

if(\*rear==MAX-1){

printf("Queue is full\n");

return;

}

else if(\*rear==-1){

\*front=\*rear=0;

queue[\*rear]=item;

}

else{

\*rear+=1;

queue[\*rear]=item;

}

}

void dequeue(int queue[],int\* front,int\*rear){

if(\*front==-1){

printf("Queue is empty\n");

return;

}

else if(\*front==\*rear){

\*front=\*rear=-1;

}

else{

\*front+=1;

}

}

void traverse(int queue[],int\* front,int\* rear){

if(\*front==-1){

printf("Queue is empty\n");

return;

}

else{

printf("Queue- ");

for(int i=\*front;i<=\*rear;i++){

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

}

printf("\n");

}

}

void reverse(int queue[],int\* front, int\* rear){

if(\*front==-1){

return;

}

int data=queue[\*front];

dequeue(queue,front,rear);

reverse(queue,front,rear);

enqueue(queue,front,rear,data);

}

int main(){

int queue[MAX];

int front=-1,rear=-1;

int choice,item;

printf("1-insert\n2-to exit\n");

do{

scanf("%d",&choice);

switch(choice){

case 1:

scanf("%d",&item);

enqueue(queue,&front,&rear,item);

traverse(queue,&front,&rear);

break;

case 2:

reverse(queue,&front,&rear);

traverse(queue,&front,&rear);

exit(0);

default:

printf("Invalid choice\n");

}

}while(1);

return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc practice.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

1-insert

2-to exit

1

924

Queue- 924

1

380

Queue- 924 380

1

206

Queue- 924 380 206

2

Queue- 206 380 924

Week 5:

Q3- Design an algorithm and write a program to implement Deque i.e. double ended queue.

**Source Code**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

void insertionatfront(int queue[],int\* front, int\* rear,int item){

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

        printf("queue is full\n");

    }

    if(\*front==-1){

        \*front=\*rear=0;

    }

    else if(\*front==0){

        \*front=MAX-1;

    }

    else{

        \*front=\*front-1;

    }

      queue[\*front]=item;

}

void insertatrear(int queue[],int\* front,int\* rear,int item){

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

        printf("Queue is full\n");

        return;

    }

    if(\*rear==-1){

        \*front=\*rear=0;

    }

    else if(\*rear==MAX-1){

        \*rear=0;

    }

    else{

        \*rear=\*rear+1;

    }

    queue[\*rear]=item;

}

void deleteatfront(int queue[],int\* front,int\* rear){

    if(\*front==-1){

        printf("Queue is empty\n");

        return;

    }

    printf("Deleted %d from the front\n",queue[\*front]);

    if(\*front==\*rear){

        \*front=\*rear=-1;

    }

    else if(\*front==MAX-1){

        \*front=0;

    }

    else{

        \*front+=1;

    }

}

void deleteatrear(int queue[],int\* front,int\* rear){

    if(\*front==-1){

        printf("Queue is empty\n");

        return;

    }

    printf("Deleted %d from rear\n",queue[\*rear]);

    if(\*front==\*rear){

        \*front=\*rear=-1;

    }

    else if(\*rear==0){

        \*rear=MAX-1;

    }

    else{

        \*rear=\*rear-1;

    }

}

void traverse(int queue[],int\* front,int\* rear){

    if(\*front==-1){

        printf("Queue is empty\n");

    }

    printf("Queue elements are - ");

    int i=\*front;

    do{

        printf("%d ",queue[i]);

        if(i==\*rear)

            break;

        i=(i+1)%MAX;

    }while(1);

    printf("\n");

}

int main(){

    int queue[MAX];

    int front=-1,rear=-1;

    int item,choice;

    printf("1: insert at front\n2: insert at rear\n3: delete from front\n4: delete from rear\n5: traverse\n6: exit\n");

    do{

        scanf("%d",&choice);

        switch(choice){

            case 1:

            scanf("%d",&item);

            insertionatfront(queue,&front,&rear,item);

            traverse(queue,&front,&rear);

            break;

            case 2:

            scanf("%d",&item);

            insertatrear(queue,&front,&rear,item);

            traverse(queue,&front,&rear);

            break;

            case 3:

            deleteatfront(queue,&front,&rear);

             traverse(queue,&front,&rear);

            break;

            case 4:

            deleteatrear(queue,&front,&rear);

            traverse(queue,&front,&rear);

            break;

            case 5:

            traverse(queue,&front,&rear);

            break;

            case 6:

            exit(0);

            default:

            printf("Invalid choice\n");

        }

    }while(1);

    return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc practice.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

1: insert at front

2: insert at rear

3: delete from front

4: delete from rear

5: traverse

6: exit

1

382

Queue elements are - 382

1

299

Queue elements are - 299 382

2

453

Queue elements are - 299 382 453

3

Deleted 299 from the front

Queue elements are - 382 453

4

Deleted 453 from rear

Queue elements are - 382

5

Queue elements are - 382

6

Week 6:

Q1-Design an algorithm and write a program to implement stack using queues .

**Source Code**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

void enqueue(int queue[],int\* front, int\* rear,int item){

if(\*rear==MAX-1){

printf("Queue is full\n");

return;

}

else if(\*rear==-1){

\*front=\*rear=0;

queue[\*rear]=item;

}else{

\*rear+=1;

queue[\*rear]=item;}

}

void dequeue(int queue[],int\* front,int\* rear){

if(\*front==-1){

printf("Queue is empty\n");

}

else if(\*front==\*rear){

\*front=\*rear=-1;

}

else{

\*front+=1;

}

}

void pushintostack(int queue1[],int queue2[],int\* front1,int\* rear1 , int\* front2,int\* rear2,int item){

if(\*rear1==MAX-1){

printf("Stack is full\n");

return;

}

while(\*front1!=-1){

int x=queue1[\*front1];

enqueue(queue2,front2,rear2,x);

dequeue(queue1,front1,rear1);

}

enqueue(queue1,front1,rear1,item);

while(\*front2!=-1){

int x=queue2[\*front2];

enqueue(queue1,front1,rear1,x);

dequeue(queue2,front2,rear2);

}

}

void popfromstack(int queue[],int\* front,int\* rear){

if(\*front==-1){

printf("Queue is empty\n");

return;

}

printf("Element popped from stack is - %d\n",queue[\*front]);

dequeue(queue,front,rear);

}

void traverse(int queue[],int\* front,int\* rear){

if(\*front==-1){

printf("Stack is empty\n");

return;

}

else{

for(int i=\*rear;i>=\*front;i--){

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

}

printf("\n");

}

}

int top(int queue[],int\* front,int\* rear){

if(\*front==-1){

printf("Queue is empty\n");

return -1;

}

return queue[\*front];

}

int main(){

int queue1[MAX],queue2[MAX];

int front1=-1,rear1=-1,front2=-1,rear2=-1;

int item,choice,x;

printf("1: to push\n2: to pop\n3 top\n4: exit\n");

do{

scanf("%d",&choice);

switch(choice){

case 1:

scanf("%d",&item);

pushintostack(queue1,queue2,&front1,&rear1,&front2,&rear2,item);

printf("stack - ");

traverse(queue1,&front1,&rear1);

break;

case 2:

popfromstack(queue1,&front1,&rear1);

printf("Stack after pop operation - ");

traverse(queue1,&front1,&rear1);

break;

case 3:

x=top(queue1,&front1,&rear1);

if(x!=-1){

printf("Top - %d\n",x);

}

break;

case 4:

exit(0);

default:

printf("Invalid choice\n");

}

}while(1);

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc practice.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

1: to push

2: to pop

3 top

4: exit

1

349

stack - 349

1

427

stack - 349 427

1

610

stack - 349 427 610

2

Element popped from stack is - 610

Stack after pop operation - 349 427

3

Top - 427

4

Week 6:

Q2-Design an algorithm and write a program to implement queue using stacks .

**Source Code**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

void push(int stk[],int\* top,int item){

if(\*top==MAX-1){

printf("Queue is empty\n");

return;

}

else{

\*top+=1;

stk[\*top]=item;}

}

void pop(int stk[],int\* top){

if(\*top==-1){

printf("Queue is empty\n");

return;

}

else{

\*top-=1;

}

}

int peek(int stk[],int top){

if(top==-1){

printf("Queue is empty\n");

return -1;

}

return stk[top];

}

void enqueuestack(int stk1[],int stk2[],int\* top1,int\* top2,int item){

if(\*top1==MAX-1){

printf("Queue is full\n");

}

while(\*top1!=-1){

int x=peek(stk1,\*top1);

push(stk2,top2,x);

pop(stk1,top1);

}

push(stk1,top1,item);

while(\*top2!=-1){

int x=peek(stk2,\*top2);

push(stk1,top1,x);

pop(stk2,top2);

}

}

void traverse(int stk[],int\* top){

if(\*top==-1){

printf("Queue is empty\n");

}

else{

printf("Queue - ");

for(int i=\*top;i>=0;i--){

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

}

printf("\n");

}

}

void dequeuestack(int stk1[],int\* top1){

if(\*top1==-1){

printf("Queue is empty\n");

return;

}

printf("Element deleted from queue - %d\n", peek(stk1, \*top1));

pop(stk1, top1);

}

int main(){

int stk1[MAX],stk2[MAX];

int top1=-1,top2=-1;

int item,choice;

printf("1. Enqueue\n2. Dequeue\n3. Peek\n4. Exit\n");

do {

scanf("%d", &choice);

switch (choice) {

case 1:

scanf("%d", &item);

enqueuestack(stk1, stk2, &top1, &top2, item);

traverse(stk1, &top1);

break;

case 2:

dequeuestack(stk1, &top1);

traverse(stk1, &top1);

break;

case 3:

item = peek(stk1, top1);

if (item != -1) {

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

}

break;

case 4:

exit(0);

default:

printf("Invalid choice\n");

}

} while (1);

return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc practice.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

1. Enqueue

2. Dequeue

3. Peek

4. Exit

1

816

Queue - 816

1

398

Queue - 816 398

1

961

Queue - 816 398 961

2

Element deleted from queue - 816

Queue - 398 961

3

Front element is 398

Week 6:

Q3- Design an algorithm and write a program to implement circular queue .

**Source Code**

#include<stdio.h>

#include<stdlib.h>

void enqueue(int queue[],int\* front,int\* rear,int item,int n){

if(\*front==(\*rear+1)%n){

printf("Queue is full \n");

return;

}

else if(\*rear==-1){

\*front=\*rear=0;

queue[\*rear]=item;

}

else{

\*rear=(\*rear+1)%n;

queue[\*rear]=item;

}

}

void dequeue(int queue[],int\* front,int\* rear,int n){

if(\*front==-1){

printf("Queue is empty\n");

return;

}

else if(\*front==\*rear){

printf("Element to be deleted is %d\n",queue[\*front]);

\*front=\*rear=-1;

}

else{

printf("Element to be deleted is %d\n",queue[\*front]);

\*front=(\*front+1)%n;

}

}

void traverse(int queue[],int\* front,int\* rear,int n){

if(\*front==-1){

printf("Queue is empty\n");

return;

}

int i=\*front;

printf("Queue - ");

while(i!=\*rear){

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

i=(i+1)%n;

}

printf("%d\n", queue[\*rear]);

}

int frontreturn(int queue[],int\*front){

if(\*front==-1){

printf("Queue is empty\n");

return -1;

}

return queue[\*front];

}

int rearreturn(int queue[],int\* rear){

if(\*rear==-1){

printf("Queue is empty\n");

return -1;

}

return queue[\*rear];

}

int main(){

int n;

printf("Enter the size of circular queue : ");

scanf("%d",&n);

int queue[n];

int front=-1,rear=-1;

int item,choice;

printf("1: to enqueue\n2: to dequeue\n3: to find front\n4: to find rear\n5: traverse\n6: exit\n");

do{

scanf("%d",&choice);

switch(choice){

case 1:

scanf("%d",&item);

enqueue(queue,&front,&rear,item,n);

traverse(queue,&front,&rear,n);

break;

case 2:

dequeue(queue,&front,&rear,n);

traverse(queue,&front,&rear,n);

break;

case 3:

item=frontreturn(queue,&front);

if(item!=-1){

printf("Front - %d\n",item);

}

break;

case 4:

item=rearreturn(queue,&rear);

if(item!=-1){

printf("Rear - %d\n",item);

}

break;

case 5:

traverse(queue,&front,&rear,n);

break;

case 6:

exit(0);

default:

printf("Invalid choice\n");

}

}while(1);

return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc lab.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

Enter the size of circular queue : 4

1: to enqueue

2: to dequeue

3: to find front

4: to find rear

5: traverse

6: exit

1

816

Queue - 816

1

398

Queue - 816 398

1

961

Queue - 816 398 961

2

Element to be deleted is 816

Queue - 398 961

3

Front - 398

4

Rear - 961

5

Queue - 398 961

6

Week 7:

Q1-Write a algorithm and a program to implement singly linked list .

**Source Code**

#include <stdio.h>

#include <stdlib.h>

struct node {

int info;

struct node\* next;

};

typedef struct node node;

node\* insertatbegin(node\* head) {

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

if (temp == NULL) {

printf("Memory not allocated\n");

} else {

printf("Enter info: ");

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

temp->next = head;

head = temp;

}

return head;

}

node\* insertionatend(node\* head) {

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

if (temp == NULL) {

printf("Memory not allocated\n");

return head;

}

printf("Enter info: ");

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

if (head == NULL) {

head = temp;

temp->next = NULL;

} else {

node\* p = head;

while (p->next != NULL) {

p = p->next;

}

p->next = temp;

temp->next = NULL;

}

return head;

}

int countNodes(node\* head) {

int count = 0;

node\* temp = head;

while (temp != NULL) {

count++;

temp = temp->next;

}

return count;

}

node\* insertAtPosition(node\* head, int position) {

int c = countNodes(head);

if (position > c + 1) {

printf("Invalid position\n");

return head;

}

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

if (temp == NULL) {

printf("Memory not allocated\n");

return head;

}

printf("Enter info: ");

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

if (position == 1) {

temp->next = head;

head = temp;

return head;

}

node\* p = head;

int currentPos = 1;

while (currentPos < position - 1 && p != NULL) {

p = p->next;

currentPos++;

}

if (p == NULL) {

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

free(temp);

} else {

temp->next = p->next;

p->next = temp;

}

return head;

}

node\* deleteathead(node\* head){

node\* temp;

if(head==NULL){

printf("linked list is empty\n");

}

else{

temp=head;

printf("the node to be deleted is %d\n",temp->info);

head=head->next;

free(temp);}

return head;

}

node\* deleteattail(node\* head){

if(head==NULL){

printf("list is empty\n");

}

if(head->next==NULL){

node\* temp=head;

free(temp);

return head;

}

node\* temp=head;

node\* p=NULL;

while(temp->next!=NULL){

p=temp;

temp=temp->next;

}

p->next=NULL;

printf("the node to be deleted is :%d\n",temp->info);

free(temp);

}

node\* deleteatk(node\* head,int position){

int c =countNodes(head);

if(position>c+1 || position<1){

printf("invalid postion\n");

}

if(position==1){

node\* temp=head;

head=temp->next;

temp->next=NULL;

free(temp);

return head;

}

node\*temp=head;

node\*p=NULL;

for(int i=1;i<position && temp!=NULL ;i++){

p=temp;

temp=temp->next;

}

p->next=temp->next;

temp->next=NULL;

free(temp);

}  
node\* middlelement(node\* head) {

node\* slow = head;

node\* fast = head;

while (fast != NULL) {

fast = fast->next;

if (fast != NULL) {

fast = fast->next;

slow = slow->next;

}

}

return slow;

}

void print(node\* head) {

node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

node\* head = NULL;

node\* middle;

int choice, position,x;

printf("1: Insert at begin\n2: Insert at end\n3: Insert at a given position\n4 delete at head \n5:delete at tail\n6:delete at position\n7: size\n8: middle element of the list\n9: print\n10: exit\n");

do {

scanf("%d", &choice);

switch (choice) {

case 1:

head = insertatbegin(head);

break;

case 2:

head = insertionatend(head);

break;

case 3:

printf("Enter the position: ");

scanf("%d", &position);

head = insertAtPosition(head, position);

break;

case 4:

head=deleteathead(head);

break;

case 5:

deleteattail(head);

break;

case 6:

printf("enter the position:");

scanf("%d",&position);

deleteatk(head,position);

break;

case 7:

x=countNodes(head);

printf("Size - %d\n",x);

break;

case 8:

middle=middlelement(head);

if(middle!=NULL){

printf("Middle element - %d\n",middle->info);

}

else{

printf("List is empty\n");

}

break;

case 9:

print(head);

break;

case 10:

exit(0);

default:

printf("Invalid choice\n");

}

} while (1);

return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc practice.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

1: Insert at begin

2: Insert at end

3: Insert at a given position

4 delete at head

5:delete at tail

6:delete at position

7: size

8: middle element of the list

9: print

10: exit

1

Enter info: 5

1

Enter info: 4

1

Enter info: 3

1

Enter info: 2

1

Enter info: 1

9

1->2->3->4->5->NULL

2

Enter info: 6

2

Enter info: 7

9

1->2->3->4->5->6->7->NULL

3

Enter the position: 3

Enter info: 0

9

1->2->0->3->4->5->6->7->NULL

8

Middle element - 4

4

the node to be deleted is 1

9

2->0->3->4->5->6->7->NULL

5

the node to be deleted is :7

9

2->0->3->4->5->6->NULL

5

the node to be deleted is :6

9

2->0->3->4->5->NULL

6

enter the position:2

9

2->3->4->5->NULL

7

Size - 4

8

Middle element - 4

Week 7:

Q2- Write an algorithm and a program to implement queue using linked list .

**Source Code**

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node\* next;

};

typedef struct node node;

node\* enqueue(node\* head){

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

if(temp==NULL){

printf("Memory not allocated\n");

return head;

}

printf("Enter info : ");

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

temp->next=head;

head=temp;

return head;

}

node\* dequeue(node\* head){

node\* temp;

if(head==NULL){

printf("Queue is empty\n");

return head;

}

temp=head;

node\* p=NULL;

while(temp->next!=NULL){

p=temp;

temp=temp->next;

}

p->next=NULL;

printf("Dequeued element is %d\n",temp->info);

free(temp);

return head;

}

int calculatesize(node\* head){

int count=0;

node\* temp=head;

while(temp!=NULL){

count++;

temp=temp->next;

}

return count;

}

void printreverse(node\* head){

if(head==NULL){

return;

}

printreverse(head->next);

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

}

void print(node\* head){

node\* temp=head;

if(head==NULL){

printf("Queue is empty\n");

}

else{

printf("Queue - ");

printreverse(head);

printf("\n");

}

}

int main(){

node\* head=NULL;

int choice,x;

printf("1: enqueue\n2: dequeue\n3: size\n4: traverse\n5: exit\n");

do{

scanf("%d",&choice);

switch(choice){

case 1:

head=enqueue(head);

print(head);

break;

case 2:

head=dequeue(head);

print(head);

break;

case 3:

x=calculatesize(head);

printf("Size - %d\n",x);

break;

case 4:

print(head);

break;

case 5:

exit(0);

default:

printf("Invalid choice\n");

}

}while(1);

return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc practice.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

1: enqueue

2: dequeue

3: size

4: traverse

5: exit

1

Enter info : 34

Queue - 34

1

Enter info : 42

Queue - 34 42

1

Enter info : 6

Queue - 34 42 6

2

Dequeued element is 34

Queue - 42 6

3

Size - 2

5

Week 7:

Q3- Write an algorithm and a program to implement stack using linked list .

**Source Code**

#include<stdio.h>

#include<stdlib.h>

struct node{

    int info;

    struct node\* next;

};

typedef struct node node;

node\* push(node\* top){

    node\* temp;

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

    if(temp==NULL){

        printf("Memeory not allocated\n");

        return top;

    }

    printf("Enter info: ");

    scanf("%d",&temp->info);

    temp->next=top;

    top=temp;

    return top;

}

node\* pop(node\* top){

    node\* temp;

    if(top==NULL){

        printf("Stack is empty\n");

        return top;

    }

    temp=top;

    top=temp->next;

    temp->next=NULL;

    printf("Popped element: %d\n", temp->info);

    free(temp);

    return top;

}

int countsize(node\* top){

    int count=0;

    node\* temp=top;

    while(temp!=NULL){

        count++;

        temp=temp->next;

    }

    return count;

}

void printReverse(node\* top) {

if (top == NULL) {

return;

}

printReverse(top->next);

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

}

void print(node\* top) {

if (top == NULL) {

printf("Stack is empty\n");

} else {

printf("Stack - ");

printReverse(top);

printf("\n");

}

}

int peek(node\* top){

if(top==0){

printf("Stack is empty\n");

return -1;

}

return top->info;

}

int main(){

node\* top=NULL;

int choice,x;

printf("1: to push\n2: to pop\n3: size\n4: peek\n5: print\n6: exit\n");

do{

scanf("%d",&choice);

switch(choice){

case 1:

top=push(top);

print(top);

break;

case 2:

top=pop(top);

print(top);

break;

case 3:

x=countsize(top);

printf("Size - %d\n",x);

break;

case 4:

x=peek(top);

printf("Peek - %d\n",x);

break;

case 5:

print(top);

break;

case 6:

exit(0);

default:

printf("Invalid choice\n");

}

}while(1);

return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc practice.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

1: to push

2: to pop

3: size

4: peek

5: print

6: exit

1

Enter info: 34

Stack - 34

1

Enter info: 42

Stack - 34 42

1

Enter info: 6

Stack - 34 42 6

2

Popped element: 6

Stack - 34 42

3

Size - 2

4

Peek - 42

6

Week 8:

Q1- Write an algorithm and a program to implement doubly linked list.

**Source Code**

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

    struct node\* prev;

    int info;

    struct node\* next;

}node;

node\* insertatbeg(node\* head){

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

    if(temp==NULL){

        printf("Memory not allocated \n");

    }

    printf("enter info :");

    scanf("%d",&temp->info);

    temp->next=NULL;

    temp->prev=NULL;

    if(head==NULL){

        head=temp;

        return head;

    }

    temp->next=head;

    if(head!=NULL){

    head->prev=temp;}

    head=temp;

    return head;

}

node\* insertatend(node\* head){

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

    if(temp==NULL){

        printf("Memory not allocated\n");

    }

    printf("enter info : ");

    scanf("%d",&temp->info);

    temp->next=NULL;

    temp->prev=NULL;

    if(head==NULL){

        head=temp;

        return head;

    }

    if(head->next==NULL){

        head->next=temp;

        temp->prev=head;

        return head;

    }

    node\* p=head;

    while(p->next!=NULL){

        p=p->next;

    }

    p->next=temp;

    temp->prev=p;

    return head;

}

int countnodes(node\* head){

    node\* temp=head;

    int count=0;

    while(temp!=NULL){

        count++;

        temp=temp->next;

    }

    return count;

}

node\* insertatanywhere(node\* head,int position){

    int c=countnodes(head);

    if(position>c+1 || position<1){

        printf("invalid position \n");

    }

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

    if(temp==NULL){

        printf("Memory not allocated\n");

    }

    printf("enter info : ");

    scanf("%d",&temp->info);

    temp->prev=NULL;

    temp->next=NULL;

    if(position==1){

        temp->next=head;

        head->prev=temp;

        head=temp;

        return head;

    }

    node\* p=head;

    int currentnode=1;

    while(currentnode<position-1 && p!=NULL){

        p=p->next;

        currentnode++;

    }

    temp->next=p->next;

    p->next=temp;

    p->next->prev=temp;

    temp->prev=p;

    return head;

}

node\* deleteathead(node\* head){

    if(head==NULL){

        printf("Linked list is empty\n");

    }

    if(head->next==NULL){

        node\* temp=head;

        printf("the node to be deleted is %d\n",temp->info);

        free(temp);

        return NULL;

    }

    node\* temp=head;

    head=head->next;

    temp->next=NULL;

    head->prev=NULL;

    printf("the node to be deleted is %d\n",temp->info);

    free(temp);

    return head;

}

node\* deleteatend(node\* head){

    if(head==NULL){

        printf("Linked list is empty\n");

        return NULL;

    }

    if(head->next==NULL){

        node\* temp=head;

        printf("the node to be deleted is %d\n",temp->info);

        free(temp);

        return NULL;

    }

    node\* temp=head;

    node\* p=NULL;

    while(temp->next!=NULL){

        p=temp;

        temp=temp->next;

    }

    p->next=temp->next;

    if(temp->next!=NULL){

    temp->next->prev=p;}

    temp->prev=NULL;

    temp->next=NULL;

    printf("the node to be deleted is %d\n",temp->info);

    free(temp);

    return head;

}

node\* deleteatanywhere(node\* head,int position){

    int c=countnodes(head);

    if(position>c || position<1){

        printf("invalid position\n");

        return head;

    }

    if(position==1){

        node\* temp=head;

        head=head->next;

        temp->next=NULL;

        printf("the node to be deleted is %d\n",temp->info);

        free(temp);

    }

    node\* p=head;

    node\* previ=NULL;

    int currentpos=1;

    while(currentpos<position && p!=NULL){

        previ=p;

        p=p->next;

        currentpos++;

    }

    previ->next=p->next;

    if(p->next!=NULL){

    p->next->prev=previ;}

    p->next=NULL;

    printf("the node to be deleted is %d\n",p->info);

    free(p);

    return head;

}

node\* middle(node\* head){

    node\* slow=head;

    node\* fast=head;

    while(fast!=NULL && fast->next!=NULL && fast->next->next!=NULL){

        slow=slow->next;

        fast=fast->next->next;

    }

    return slow;

}

int size(node\* head){

    return countnodes(head);

}

int isempty(node\* head){

    return head==NULL;

}

void traverse(node\* head){

    if(head==NULL){

        printf("Linked list is empty \n");

    }

    node\* temp=head;

    while(temp!=NULL){

        printf("%d->",temp->info);

        temp=temp->next;

    }

    printf("NULL\n");

}

int main(){

    node\* head=NULL;

    node\* mid;

    int choice,position,x,z;

    printf("Press : \n1: Insert at head\n2: Insert at tail\n3: Insert at anywhere\n4: Delete at head\n5: Delete at tail\n6: Delete at anywhere\n7: Traverse\n8: Middle\n9: Size\n10: Empty \n11: Exit\n");

    do{

        scanf("%d",&choice);

        switch(choice){

            case 1:

            head=insertatbeg(head);

            traverse(head);

            break;

            case 2:

            head=insertatend(head);

            traverse(head);

            break;

            case 3:

            printf("enter position : ");

            scanf("%d",&position);

            head=insertatanywhere(head,position);

            traverse(head);

            break;

            case 4:

            head=deleteathead(head);

            traverse(head);

            break;

            case 5:

            head=deleteatend(head);

            traverse(head);

            break;

            case 6:

            printf("enter position : ");

            scanf("%d",&position);

            head=deleteatanywhere(head,position);

            traverse(head);

            break;

            case 7:

            traverse(head);

            break;

            case 8:

            mid=middle(head);

            if(mid!=NULL){

                printf("Middle - %d\n",mid->info);

            }

            else{

                printf("List is empty\n");

            }

            break;

            case 9:

            x=size(head);

            printf("Size - %d\n",x);

            break;

            case 10:

            if(isempty(head)){

                printf("Linked list is empty\n");

            }

            else{

                printf("Linked list is not empty\n");

            }

            break;

            case 11:

            exit(0);

            default:

            printf("invalid choice\n");

        }

    }while(1);

    return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc trees.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

Press :

1: Insert at head

2: Insert at tail

3: Insert at anywhere

4: Delete at head

5: Delete at tail

6: Delete at anywhere

7: Traverse

8: Middle

9: Size

10: Empty

11: Exit

2

enter info : 568

568->NULL

1

enter info :894

894->568->NULL

4

the node to be deleted is 894

568->NULL

11

Week 8:

Q2- Given a doubly linked list, design an algorithm and write a program to reverse this list without using any extra space

**Source Code**

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

    struct node\* prev;

    int info;

    struct node\* next;

}node;

node\*insertatend(node\* head){

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

    if(temp==NULL){

        printf("Memory not allocated\n");

        return head;

    }

    printf("enter info : ");

    scanf("%d",&temp->info);

    temp->prev=NULL;

    temp->next=NULL;

    if(head==NULL){

        head=temp;

        return head;

    }

    if(head->next==NULL){

        head->next=temp;

        temp->prev=head;

        return head;

    }

    node\* p=head;

    while(p->next!=NULL){

        p=p->next;

    }

    p->next=temp;

    temp->prev=p;

    return head;

}

node\* reverse(node\* head){

    node\* prev=NULL;

    node\* curr=head;

    node\* forward=NULL;

    while(curr!=NULL){

        forward=curr->next;

        curr->next=prev;

        curr->prev=forward;

        prev=curr;

        curr=forward;

    }

    return prev;

}

void traverse(node\* head){

    if(head==NULL){

        printf("Linked list is empty\n");

    }

    node\* temp=head;

    while(temp!=NULL){

        printf("%d<->",temp->info);

        temp=temp->next;

    }

    printf("NULL\n");

}

int main(){

    node\* head=NULL;

    int choice;

    printf("Press : \n1: Insert \n2: Exit\n");

    do{

        scanf("%d",&choice);

        switch(choice){

            case 1:

            head=insertatend(head);

            traverse(head);

            break;

            case 2:

            head=reverse(head);

            traverse(head);

            exit(0);

            default:

            printf("Invalid choice\n");

        }

    }while(1);

    return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc trees.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

Press :

1: Insert

2: Exit

1

enter info : 689

689<->NULL

1

enter info : 961

689<->961<->NULL

1

enter info : 731

689<->961<->731<->NULL

2

731<->961<->689<->NULL

Week 8:

Q3- . Given a sorted doubly linked list, design an algorithm and write a program to eliminate duplicates from this list

**Source Code**

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

    struct node\* prev;

    int info;

    struct node\* next;

}node;

node\*insertatend(node\* head){

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

    if(temp==NULL){

        printf("Memory not allocated\n");

        return head;

    }

    printf("enter info : ");

    scanf("%d",&temp->info);

    temp->prev=NULL;

    temp->next=NULL;

    if(head==NULL){

        head=temp;

        return head;

    }

    if(head->next==NULL){

        head->next=temp;

        temp->prev=head;

        return head;

    }

    node\* p=head;

    while(p->next!=NULL){

        p=p->next;

    }

    p->next=temp;

    temp->prev=p;

    return head;

}

node\* removedupsorted(node\* head) {

    if (head == NULL) {

        return NULL;

    }

    node\* curr = head;

    while (curr != NULL && curr->next != NULL) {

        if (curr->info == curr->next->info) {

            node\* dup = curr->next;

            curr->next = dup->next;

            if (dup->next != NULL) {

                dup->next->prev = curr;

            }

            printf("Duplicate removed: %d\n", dup->info);

            free(dup);

        } else {

            curr = curr->next;

        }

    }

    return head;

}

void traverse(node\* head){

    if(head==NULL){

        printf("Linked list is empty\n");

        return ;

    }

    node\* temp=head;

    while(temp!=NULL){

        printf("%d<->",temp->info);

        temp=temp->next;

    }

    printf("NULL\n");

}

int main(){

    node\* head=NULL;

    int choice;

    printf("Press :\n1: Insert \n2: Remove duplicated from sorted List\n3: Exit\n");

    do{

        scanf("%d",&choice);

        switch(choice){

            case 1:

            head=insertatend(head);

            traverse(head);

            break;

            case 2:

            head=removedupsorted(head);

            traverse(head);

            break;

            case 3:

            exit(0);

            default:

            printf("Invalid choice\n");

        }

    }while(1);

    return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc trees.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

Press :

1: Insert

2: Remove duplicated from sorted List

3: Exit

1

enter info : 689

689<->NULL

1

enter info : 731

689<->731<->NULL

1

enter info : 961

689<->731<->961<->NULL

1

enter info : 961

689<->731<->961<->961<->NULL

2

Duplicate removed: 961

689<->731<->961<->NULL

3

Week 9:

Q1- Design an algorithm and write a program to implement circular linked list

**Source Code**

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

    int info;

    struct node\* next;

}node;

node\* insertatfront(node\* head){

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

    if(temp==NULL){

        printf("Memory not allocated\n");

        return head;

    }

    printf("enter info :");

    scanf("%d",&temp->info);

    temp->next=NULL;

    if(head==NULL){

        head=temp;

        temp->next=head;

        return head;

    }

    if(head->next==head){

        temp->next=head;

        head->next=temp;

        head=temp;

        return head;

    }

    node\* p=head;

    while(p->next!=head){

        p=p->next;

    }

    temp->next=head;

    p->next=temp;

    head=temp;

    return head;

}

node\* insertatrear(node\* head){

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

    if(temp==NULL){

        printf("Memory not allocated\n");

        return head;

    }

    printf("enter info : ");

    scanf("%d",&temp->info);

    temp->next=NULL;

    if(head==NULL){

        head=temp;

        temp->next=head;

        return head;

    }

    if(head->next==head){

        head->next=temp;

        temp->next=head;

        head=temp;

        return head;

    }

    node\* p=head;

    while(p->next!=head){

        p=p->next;

    }

    p->next=temp;

    temp->next=head;

    return head;

}

int countnodes(node\* head){

    if(head==NULL){

        return 0;

    }

    node\* temp=head;

    int count=0;

    do{

        count++;

        temp=temp->next;

    }while(temp!=head);

    return count;

}

node\* insertanywhere(node\* head,int position){

    int c=countnodes(head);

    if(position>c+1 || position<1){

        printf("Invalid position\n");

        return head;

    }

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

    if(temp==NULL){

        printf("Memory not allocted\n");

        return head;

    }

    printf("enter info : ");

    scanf("%d",&temp->info);

    if(position==1){

        if(head==NULL){

            head=temp;

            temp->next=head;

            return head;

        }

        node\* p=head;

        while(p->next!=head){

            p=p->next;

        }

        temp->next=head;

        p->next=temp;

        head=temp;

        return head;

    }

    node\* p=head;

    int currentpos=1;

    while(currentpos<position-1 && p!=NULL){

        p=p->next;

        currentpos++;

    }

    if(position==c+1){

        p->next=temp;

        temp->next=head;

    }else{

    temp->next=p->next;

    p->next=temp;}

    return head;

}

node\* deleteatfront(node\* head){

    if(head==NULL){

        printf("Linked list is empty\n");

        return NULL;

    }

    if(head->next==head){

        node\* temp=head;

        printf("The node to be deleted is %d\n",temp->info);

        free(temp);

        return NULL;

    }

    node\* p=head;

    while(p->next!=head){

        p=p->next;

    }

    node\* temp=head;

    head=temp->next;

    p->next=head;

    temp->next=NULL;

    printf("the node to be deleted is %d\n",temp->info);

    free(temp);

    return head;

}

node\* deleteatrear(node\* head){

    if(head==NULL){

        printf("Linked list is empty\n");

        return NULL;

    }

    if(head->next==head){

        node\* temp=head;

        printf("the node to be deleted is %d\n",temp->info);

        free(temp);

        return NULL;

    }

    node\* temp=head;

    node\* p=NULL;

    while(temp->next!=head){

        p=temp;

        temp=temp->next;

    }

    p->next=head;

    temp->next=NULL;

    printf("the node to be deleted is %d\n",temp->info);

    free(temp);

    return head;

}

node\* deleteatanywhere(node\* head,int position){

    int c=countnodes(head);

    if(position>c || position<1){

        printf("Invalid position\n");

        return head;

    }

    if(position==1){

        node\*p =head;

        while(p->next!=head){

            p=p->next;

        }

        node\* temp=head;

        if(head->next==head){

            head=NULL;

        }else{

        head=head->next;

        p->next=head;

        temp->next=NULL;}

        printf("the node to be deleted is %d\n",temp->info);

        free(temp);

        return head;

    }

    node\* temp=head;

    node\* p=NULL;

    int currentpos=1;

    while(currentpos<position && temp!=NULL){

        p=temp;

        temp=temp->next;

        currentpos++;

    }

    if(position==c){

        p->next=head;

        temp->next=NULL;

        printf("the node to be deleted is %d\n",temp->info);

        free(temp);

    }else{

        p->next=temp->next;

        temp->next=NULL;

        printf("the node to be deleted is %d\n",temp->info);

        free(temp);

    }

    return head;

}

int size(node\* head){

    return countnodes(head);

}

void traverse(node\* head){

    if(head==NULL){

        printf("Linked list is empty\n");

    }

    node\* temp=head;

   do{

    printf("%d->",temp->info);

    temp=temp->next;

   }while(temp!=head);

   printf("Back to head\n");

}

int isempty(node\* head){

    return head==NULL;

}

int main(){

    node\* head=NULL;

    int choice,position,x;

    printf("Press :\n1: Insert at front\n2: Insert at rear\n3: Insert at anywhere\n4: Delete at front\n5: Delete at rear\n6: Delete at anywhere\n7: Traverse\n8: Size\n9: Empty\n10: Exit\n");

    do{

        scanf("%d",&choice);

        switch(choice){

            case 1:

            head=insertatfront(head);

            traverse(head);

            break;

            case 2:

            head=insertatrear(head);

            traverse(head);

            break;

            case 3:

            printf("enter position :");

            scanf("%d",&position);

            head=insertanywhere(head,position);

            traverse(head);

            break;

            case 4:

            head=deleteatfront(head);

            traverse(head);

            break;

            case 5:

            head=deleteatrear(head);

            traverse(head);

            break;

            case 6:

            printf("enter position : ");

            scanf("%d",&position);

            head=deleteatanywhere(head,position);

            traverse(head);

            break;

            case 7:

            traverse(head);

            break;

            case 8:

            x=size(head);

            printf("Size - %d\n",x);

            break;

            case 9:

            if(isempty(head)){

                printf("Linked list is empty\n");

            }

            else{

                printf("Linked list is not empty\n");

            }

            break;

            case 10:

            exit(0);

            default:

            printf("Invalid choice\n");

        }

    }while(1);

    return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc trees.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

Press :

1: Insert at front

2: Insert at rear

3: Insert at anywhere

4: Delete at front

5: Delete at rear

6: Delete at anywhere

7: Traverse

8: Size

9: Empty

10: Exit

2

enter info : 568

568->Back to head

1

enter info :894

894->568->Back to head

1

enter info :157

157->894->568->Back to head

4

the node to be deleted is 157

894->568->Back to head

10

Week 9:

Q2- Design an algorithm and write a program to concatenate two circularly linked lists

**Source Code**

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

    int info;

    struct node\* next;

}node;

node\* insertatrear(node\* head){

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

    if(temp==NULL){

        printf("Memory not allocated\n");

        return head;

    }

    printf("enter info :");

    scanf("%d",&temp->info);

    temp->next=NULL;

    if(head==NULL){

        head=temp;

        temp->next=head;

        return head;

    }

    if(head->next==head){

        head->next=temp;

        temp->next=head;

        return head;

    }

    node\* p=head;

    while(p->next!=head){

        p=p->next;

    }

    p->next=temp;

    temp->next=head;

    return head;

}

void traverse(node\* head){

    if(head==NULL){

        printf("Linked list is empty\n");

    }

    node\* temp=head;

    do{

        printf("%d->",temp->info);

        temp=temp->next;

    }while(temp!=head);

    printf("Back to head\n");

}

node\* concatenate(node\* head1,node\* head2){

    if(head1==NULL) return head2;

    if(head2==NULL) return head1;

    node\* temp1=head1;

    node\* temp2=head2;

    while(temp1->next!=head1){

        temp1=temp1->next;

    }

    while(temp2->next!=head2){

        temp2=temp2->next;

    }

    temp1->next=head2;

    temp2->next=head1;

    return head1;

}

int main(){

    node\* head1=NULL;

    node\* head2=NULL;

    int choice;

    printf("Press :\n1: Insert at list 1\n2: Insert at list 2\n3: Concatenate List\n4: Exit\n");

    do{

        scanf("%d",&choice);

        switch(choice){

            case 1:

            head1=insertatrear(head1);

            traverse(head1);

            break;

            case 2:

            head2=insertatrear(head2);

            traverse(head2);

            break;

            case 3:

            head1=concatenate(head1,head2);

            traverse(head1);

            break;

            case 4:

            exit(0);

            default:

            printf("Invalid choice\n");

        }

    }while(1);

    return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc trees.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

Press :

1: Insert at list 1

2: Insert at list 2

3: Concatenate List

4: Exit

1

enter info :864

864->Back to head

1

enter info :628

864->628->Back to head

1

enter info :193

864->628->193->Back to head

2

enter info :197

197->Back to head

2

enter info :496

197->496->Back to head

3

864->628->193->197->496->Back to head

4

Week 9:

Q3- Write an algorithm and a program that will split a circularly linked list into two circularly linked list provided position from where circular linked list has to be splitted.

**Source Code**

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

    int info;

    struct node\* next;

}node;

node\* insertatrear(node\* head){

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

    if(temp==NULL){

        printf("Memory not allocated\n");

        return head;

    }

    printf("enter info : ");

    scanf("%d",&temp->info);

    temp->next=NULL;

    if(head==NULL){

        head=temp;

        temp->next=head;

        return head;

    }

    if(head->next==head){

        head->next=temp;

        temp->next=head;

        return head;

    }

    node\* p=head;

    while(p->next!=head){

        p=p->next;

    }

    p->next=temp;

    temp->next=head;

    return head;

}

int countnodes(node\* head){

    if(head==NULL){

        return 0;

    }

    node\* temp=head;

    int count=1;

    while(temp->next!=head){

        count++;

        temp=temp->next;

    }

    return count;

}

void traverse(node\* head){

    if(head==NULL){

        printf("Linked list is empty\n");

    }

    node\* temp=head;

    do{

        printf("%d->",temp->info);

        temp=temp->next;

    }while(temp!=head);

    printf("Back to head\n");

}

void split(node\* head, int position) {

    int c = countnodes(head);

    if (position > c || position < 1) {

        printf("Invalid position\n");

        return;

    }

    node\* temp = head;

    int currentPos = 1;

    while (currentPos < position && temp->next != head) {

        temp = temp->next;

        currentPos++;

    }

node\* head2 = temp->next;

    temp->next = head;

    node\* temp2 = head2;

    while (temp2->next != head) {

        temp2 = temp2->next;

    }

    temp2->next = head2;

    printf("List 1: ");

    traverse(head);

    printf("List 2: ");

    traverse(head2);

}

int main(){

    node\* head=NULL;

    int choice,position;

    printf("Press : \n1: Insert \n2: Split linked list\n3: Exit\n");

    do{

        scanf("%d",&choice);

        switch(choice){

            case 1:

            head=insertatrear(head);

            traverse(head);

            break;

            case 2:

            printf("enter position : ");

            scanf("%d",&position);

            split(head,position);

            break;

            case 3:

            exit(0);

            default:

            printf("Invalid choice\n");

        }

    }while(1);

    return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc trees.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

Press :

1: Insert

2: Split linked list

3: Exit

1

enter info : 864

864->Back to head

1

enter info : 628

864->628->Back to head

1

enter info : 193

864->628->193->Back to head

1

enter info : 496

864->628->193->496->Back to head

2

enter position : 3

List 1: 864->628->193->Back to head

List 2: 496->Back to head

3

Week 10:

Q1- Write an algorithm that will split a linked list into two linked lists, so that successive nodes go to different lists. Odd numbered nodes to the first list while even numbered nodes to the second list.

**Source Code**

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

    int info;

    struct node\* next;

}node;

node\* insertatrear(node\* head){

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

    if(temp==NULL){

        printf("Memory not allocated\n");

        return head;

    }

    printf("enter info :");

    scanf("%d",&temp->info);

    temp->next=NULL;

    if(head==NULL){

        head=temp;

        return head;

    }

    if(head->next==NULL){

        head->next=temp;

        return head;

    }

    node\* p=head;

    while(p->next!=NULL){

        p=p->next;

    }

    p->next=temp;

    return head;

}

void traverse(node\* head){

    if(head==NULL){

        printf("Linked list is empty\n");

    }

    node\* temp=head;

    while(temp!=NULL){

        printf("%d->",temp->info);

        temp=temp->next;

    }

    printf("NULL\n");

}

void seggreate(node\* head){

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

        return ;

    }

    node\* oddnode=head;

    node\* evenode=head->next;

    node\* evenhead=head->next;

    while(evenode!=NULL && evenode->next!=NULL){

        oddnode->next=oddnode->next->next;

        oddnode=oddnode->next;

        evenode->next=evenode->next->next;

        evenode=evenode->next;

    }

    printf("Odd numbered Linked list : ");

    traverse(head);

    printf("Even numbered Linked list : ");

    traverse(evenhead);

}

int main(){

    node\* head=NULL;

    int choice;

    printf("Press : \n1: Insert \n2: Split\n3: Exit\n");

    do{

        scanf("%d",&choice);

        switch(choice){

            case 1:

            head=insertatrear(head);

            traverse(head);

            break;

            case 2:

            seggreate(head);

            break;

            case 3:

            exit(0);

            default:

            printf("Invalid choice\n");

        }

    }while(1);

    return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc trees.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

Press :

1: Insert

2: Split

3: Exit

1

enter info :380

380->NULL

1

enter info :234

380->234->NULL

1

enter info :963

380->234->963->NULL

1

enter info :125

380->234->963->125->NULL

1

enter info :154

380->234->963->125->154->NULL

2

Odd numbered Linked list : 380->963->154->NULL

Even numbered Linked list : 234->125->NULL

3

Week 10:

Q2- Given a linked list and a number n, design an algorithm and write a program to find the value at the n'th node from end of this linked list

**Source Code**

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

    int info;

    struct node\* next;

}node;

node\* insertatrear(node\* head){

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

    if(temp==NULL){

        printf("Memory not allocated\n");

        return head;

    }

    printf("enter info :");

    scanf("%d",&temp->info);

    temp->next=NULL;

    if(head==NULL){

        head=temp;

        return head;

    }

    if(head->next==NULL){

        head->next=temp;

        return head;

    }

    node\* p=head;

    while(p->next!=NULL){

        p=p->next;

    }

    p->next=temp;

    return head;

}

int findnthnode(node\* head,int n){

    node\* fast=head;

    for(int i=0;i<n;i++){

        fast=fast->next;

    }

    if(fast==NULL){

        printf("List has less than %d nodes\n",n);

        return -1;

    }

    node\* slow=head;

    while(fast!=NULL){

        slow=slow->next;

        fast=fast->next;

    }

    return slow->info;

}

void traverse(node\* head){

    if(head==NULL){

        printf("Linked list is empty\n");

    }

    node\* temp=head;

    while(temp!=NULL){

        printf("%d->",temp->info);

        temp=temp->next;

    }

    printf("NULL\n");

}

int main(){

    node\* head=NULL;

    int choice,n,x;

    printf("Press : \n1: Insert\n2: Value from end\n3: Exit\n");

    do{

        scanf("%d",&choice);

        switch(choice){

            case 1:

            head=insertatrear(head);

            traverse(head);

            break;

            case 2:

            printf("Enter value n : ");

            scanf("%d",&n);

            x=findnthnode(head,n);

            if(x!=-1){

                printf("The %d node from the end is %d\n",n,x);

            }

            break;

            case 3:

            exit(0);

            default:

            printf("Invalid choice\n");

        }

    }while(1);

    return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc trees.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

Press :

1: Insert

2: Value from end

3: Exit

1

enter info :380

380->NULL

1

enter info :963

380->963->NULL

1

enter info :125

380->963->125->NULL

1

enter info :154

380->963->125->154->NULL

2

Enter value n : 3

The 3 node from the end is 963

3

Week 10:

Q3- Write a program that will reverse a linked list only by traversing it once and without using extra space.

**Source Code**

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

    int info;

    struct node\* next;

}node;

node\* insertatrear(node\* head){

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

    if(temp==NULL){

        printf("Memory not allocated\n");

        return head;

    }

    printf("enter info :");

    scanf("%d",&temp->info);

    temp->next=NULL;

    if(head==NULL){

        head=temp;

        return head;

    }

    if(head->next==NULL){

        head->next=temp;

        return head;

    }

    node\* p=head;

    while(p->next!=NULL){

        p=p->next;

    }

    p->next=temp;

    return head;

}

node\* reverse(node\* head){

   node\* curr=head;

   node\* prev=NULL;

   node\* forward=NULL;

   while(curr!=NULL){

    forward=curr->next;

    curr->next=prev;

    prev=curr;

    curr=forward;

   }

   return prev;

}

void traverse(node\* head){

    if(head==NULL){

        printf("Linked list is empty\n");

    }

    node\* temp=head;

    while(temp!=NULL){

        printf("%d->",temp->info);

        temp=temp->next;

    }

    printf("NULL\n");

}

int main(){

    node\* head=NULL;

    int choice;

    printf("Press : \n1: Insert\n2: Reverse\n3: Exit\n");

    do{

        scanf("%d",&choice);

        switch(choice){

            case 1:

            head=insertatrear(head);

            traverse(head);

            break;

            case 2:

            head=reverse(head);

            traverse(head);

            case 3:

            exit(0);

            default:

            printf("Invalid choice\n");

        }

    }while(1);

    return 0;

}

**Output**

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> gcc trees.c

PS C:\Users\A\OneDrive\Desktop\AkshatVerma.cpp\23011483> ./a.exe

Press :

1: Insert

2: Reverse

3: Exit

1

enter info :780

780->NULL

1

enter info :563

780->563->NULL

1

enter info :125

780->563->125->NULL

1

enter info :154

780->563->125->154->NULL

2

154->125->563->780->NULL