****DATA STRUCTURE** LAB FILE**



****SUBMITTED BY:****

****NAME: ZISHNENDU SARKER****

****ROLL: 2K19/C0/450****

****ANALOG ELECTRONICS LAB (G3)****

****SUBMITTED TO :****

****Dr. MANOJ KUMAR SIR****

****DEPARTMENT OF COMPUTER SCIENCE &ENGINEERING****

*****VISION*****

Department of Computer Science & Engineering to be a leading world class technology department playing its role as a key node in national and global knowledge network, thus empowering the computer science industry with the wings of knowledge and power of innovation.

***MISSION***

The Mission of the department is as follows:  
  
1. To nurture talent of students for research, innovation and excellence in the field of computer engineering starting from Under graduate level.  
  
2. To develop highly analytical and qualified computer engineers by imparting training on cutting edge technology.  
  
3. To produce socially sensitive computer engineers with professional ethics.  
  
4. To focus on R&D environment in close partnership with industry and foreign universities.  
  
5. To produce well-rounded, up to date, scientifically tempered, design oriented engineer and scientists capable of lifelong learning

***INDEX***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ****S.no**** | ****NAME OF PRACTICAL**** | ****DATE OF SUBMISSION**** | ****DATE OF FINAL SUBMISSION**** | **Page** | ****REMARK**** |
| ****1.**** | Merge Sort- Recursive version. | **18-08-2020** | **25-11-2020** | 5 |  |
| ****2.**** | Merge Sort - Iterative version | **18-08-2020** | **25-11-2020** | 9 |  |
| ****3.**** | Managing unsorted list- menu driven program to perform Insert, Delete and Search operations | **18-08-2020** | **25-11-2020** | 14 |  |
| ****4.**** | Managing sorted list- menu driven program to perform Insert, Delete and Search operations. | **18-08-2020** | **25-11-2020** | 19 |  |
| ****5.**** | Stack implementation using arrays:- menu driven program | **18-08-2020** | **25-11-2020** | 24 |  |
| ****6.**** | Menu driven program for Queue simulation using array | **25-08-2020** | **25-11-2020** | 29 |  |
| ****7.**** | To check if given string contains balanced paranthesis | **25-08-2020** | **25-11-2020** | 34 |  |
| ****8.**** | Represent polynomial using array and write program to add and multiply two polynomials. | **25-08-2020** | **25-11-2020** | 37 |  |
| ****9.**** | Infix to postfix conversion using stack | **01-09-2020** | **25-11-2020** | 42 |  |
| ****10.**** | Sparse Matrix addition. | **01-09-2020** | **25-11-2020** | 46 |  |
| ****11.**** | Sparse Matrix Transpose and Multiplication | **01-09-2020** | **25-11-2020** | 53 |  |
| ****12.**** | Menu driven simple linked list without head node. insert(start, end, after), delete, search,count operations. | **14-09-2020** | **25-11-2020** | 62 |  |
| ****13.**** | Repeat 12 using linked list with head node. | **14-09-2020** | **25-11-2020** | 80 |  |
| ****14.**** | maintain a sorted linked list, implement insert, delete, search, count | **14-09-2020** | **25-11-2020** | 90 |  |
| ****15.**** | Implementation of Insert, delete, search, Inorder, preorder, postorder, height,countNodes, functions for Binary Search Tree. | **06-10-2020** | **25-11-2020** | 104 |  |
| ****16.**** | Implement Priority Queue using MaxHeap: InsertQ(), DeleteQ(), DisplayHeap() are the functions to be implemented | **06-10-2020** | **25-11-2020** | 109 |  |
| ****17.**** | Design HeapSort() to sort array in ascending order. | **02-11-2020** | **25-11-2020** | 115 |  |

***EXPERIMENT – 1***

**Objectives :-** Merge Sort Recursive Version.

**Code: -**

#include <iostream>

#include <stdlib.h>

using namespace std;

void Merge(int arr[], int l, int m, int r)

{

int i, j, k, n1=m-l+1, n2=r - m, L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

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

i=0,j=0,k=l;

while(i<n1 and j<n2){

if(L[i]<=R[j]){

arr[k] = L[i];

i++;

}

else{

arr[k]=R[j];

j++;

}

k++;

}

while(i<n1){

arr[k] = L[i];

i++;

k++;

}

while(j<n2) {

arr[k] = R[j];

j++;

k++;

}

}

void MergeSort(int arr[], int l, int r)

{

if (l < r){

int m=l+(r-l)/2;

MergeSort(arr, l, m);

MergeSort(arr, m+1, r);

Merge(arr, l, m, r);

}

}

void Array(int A[], int size)

{

int i;

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

cout<<A[i]<<" ";

cout<<endl;

}

int main()

{

int n;

cin>>n;

int arr[n];

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

cin>>arr[i];

}

int arr\_size = sizeof(arr) / sizeof(arr[0]);

cout<<"Given array: ";

Array(arr, arr\_size);

MergeSort(arr, 0, arr\_size - 1);

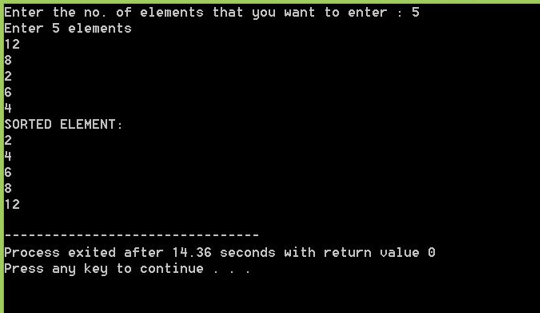
cout<<"Sorted array: ";

Array(arr, arr\_size);

return 0;

}

**OUTPUT**



***EXPERIMENT – 2***

**Objectives :-** Merge Sort Iterative Version

CODE: -

#include<stdlib.h>

#include<stdio.h>

void merge(int arr[], int l, int m, int r);

int min(int x, int y) { return (x<y)? x :y; }

void mergeSort(int arr[], int n)

{

int curr\_size;

int left\_start;

for (curr\_size=1; curr\_size<=n-1; curr\_size = 2\*curr\_size)

{

for (left\_start=0; left\_start<n-1; left\_start += 2\*curr\_size)

{

int mid = min(left\_start + curr\_size - 1, n-1);

int right\_end = min(left\_start + 2\*curr\_size - 1, n-1);

merge(arr, left\_start, mid, right\_end);

}

}

}

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

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

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

void printArray(int A[], int size)

{

int i;

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

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

printf("\n");

}

int main()

{

int arr[] = {12, 11, 13, 5, 6, 7};

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

printf("Given array is \n");

printArray(arr, n);

mergeSort(arr, n);

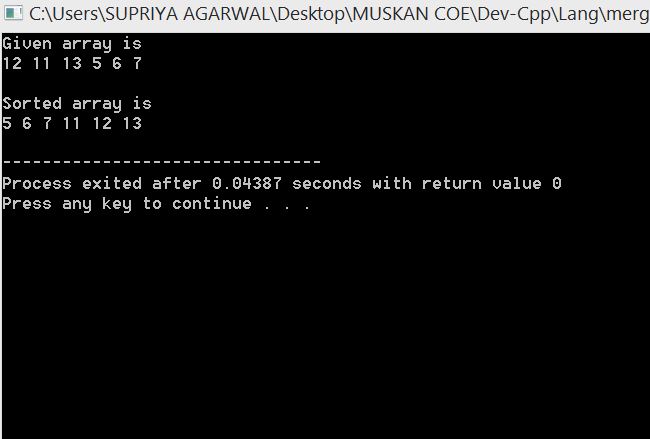
printf("\nSorted array is \n");

printArray(arr, n);

return 0;

}

OUTPUT: -



***EXPERIMENT – 3***

**Objectives :-** Managing Unsorted List- Menu Driven Program To Perform Insert, Delete And Search Operations.

**CODE: -**

#include <iostream>

using namespace std;

int main()

{

int n;

cin>>n;

int arr[100]={0};

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

{

cin>>arr[i]; //unsorted

}

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

{

cout<<arr[i]<<" ";

}cout<<endl;

cout<<"1.Insert 2.Delete 3.Find Minimum Element 4.Find Maximum Element 5.Display List 6.Exit"<<endl;

char ch='y';

while(ch=='y')

{

cout<<"Enter your choice:";

int key;

cin>>key;

if(key==1){

cout<<"Enter the no. you want to insert :";

int num1;

cin>>num1;

arr[n]=num1;

n++;

ch ='y';cout<<endl;

}

else if(key==2){

cout<<"Enter the index at which u want to delete the element :";

int idx;

cin>>idx;

for(int j=idx;j<n;j++)

{

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

}n--;ch ='y';cout<<endl;

}

else if(key==3){

int min=arr[0];

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

{

if(arr[i]<min){

min=arr[i];

}

}cout<<"Minimum element is "<<min;

ch ='y';cout<<endl;

}

else if(key==4){

int max=arr[0];

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

{

if(arr[i]>max){

max=arr[i];

}

}cout<<"Maximum element is "<<max;

ch ='y';cout<<endl;

}

else if(key==5){

cout<<"Currently the list is:"<<endl;

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

{

cout<<arr[i]<<" ";

}ch ='y';cout<<endl;

}

else if(key==6){

cout<<"Exit"<<endl;break;

}

else{

cout<<"Invalid";ch ='y';cout<<endl;

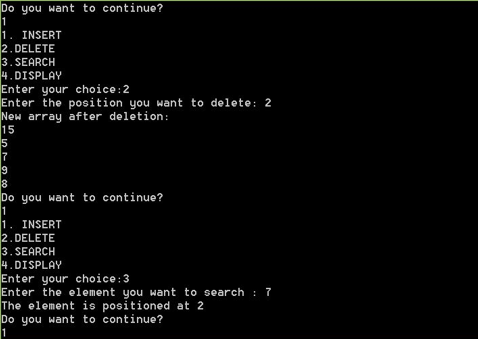
}

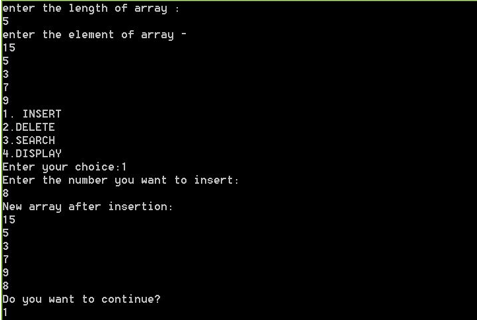
}

return 0;

}

**OUTPUT: -**





***EXPERIMENT – 4***

**Objectives :-** Managing Sorted List - Menu Driven Program To Perform Insert, Delete and Search Operations.

**Code:-**

#include <iostream>

using namespace std;

int main()

{

int n;

cin>>n;

int arr[100]={0};

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

{

cin>>arr[i]; //sorted

}

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

{

cout<<arr[i]<<" ";

}cout<<endl;

cout<<"1.Insert 2.Delete 3.Find Minimum Element 4.Find Maximum Element 5.Display List 6.Exit"<<endl;

char ch='y';

while(ch=='y')

{

cout<<"Enter your choice:";

int key;

cin>>key;

if(key==1){

cout<<"Enter the no. you want to insert :";

int num1;

cin>>num1;int t=0;

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

{

if(num1>arr[i]){

t++;

}

else break;

}

for(int j=n-1;j>=t;j--)

{

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

}

arr[t]=num1;n++;

ch=='y';cout<<endl;

}

else if(key==2){

cout<<"Enter the index at which u want to delete the element :";

int idx;

cin>>idx;

for(int j=idx;j<n;j++)

{

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

}n--;ch=='y';cout<<endl;

}

else if(key==3){

cout<<"Minimum element is "<<arr[0];

ch=='y';cout<<endl;

}

else if(key==4){

cout<<"Maximum element is "<<arr[n-1];

ch=='y';cout<<endl;

}

else if(key==5){

cout<<"Currently the list is:"<<endl;

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

{

cout<<arr[i]<<" ";

}ch=='y';cout<<endl;

}

else if(key==6){

cout<<"Exit"<<endl;break;

}

else{

cout<<"Invalid";ch=='y';cout<<endl;

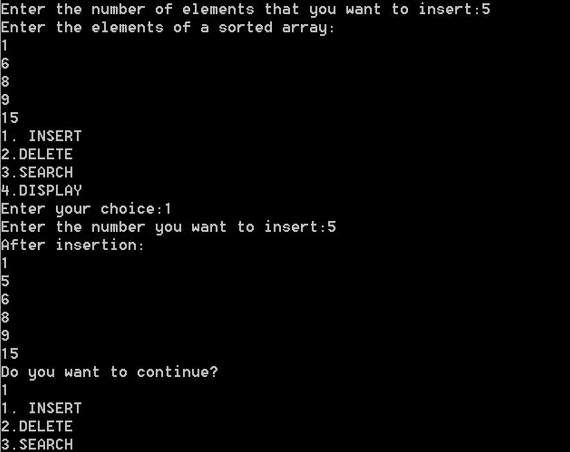
}

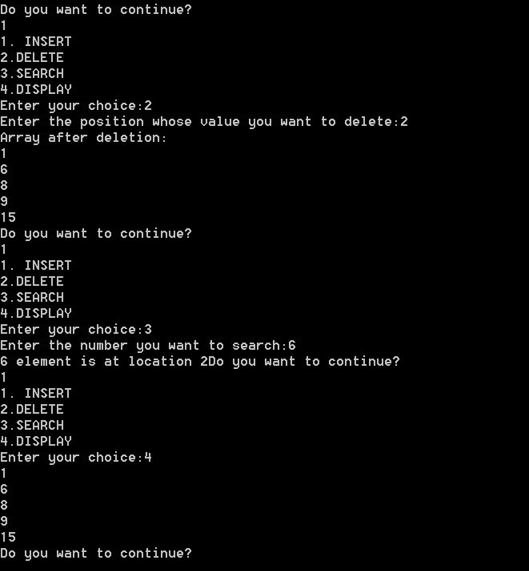
}

return 0;

}

**OUTPUT: -**





***EXPERIMENT – 5***

**Objectives :-** Stack implementation Using Arrays , Menu Driven Program.

**CODE : -**

#include<stdio.h>

#define MAXSIZE 5

struct stack

{

int a[MAXSIZE];

int top;

};

void push(struct stack\*s,int x)

{

if(s->top==MAXSIZE-1)

{

printf("STACK IS FULL.\n");

return;

}

s->top=s->top+1;

s->a[s->top]=x;

}

int pop(struct stack\*s)

{

int y;

if(s->top==-1)

{

printf("STACK IS EMPTY\n");

return -1;

}

y=s->a[s->top];

s->top=s->top-1;

return y;

}

void display(struct stack s)

{

printf("STACK CONTENTS:\n");

int i;

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

{

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

}

}

int main()

{

struct stack s;

int choice,x;

s.top=-1;

while(1)

{

printf("1.Push\n");

printf("2.Pop\n");

printf("3.Display\n");

printf("4.Exit\n");

printf("ENTER YOUR CHOICE\n");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter a number:");

scanf("%d",&x);

push(&s,x);

display(s);

break;

case 2:

x=pop(&s);

if(x!=-1)

{

printf("Popped data is =%d\n",x);

display(s);

}

break;

case 3:

display(s);

break;

}

if(choice==4)

{

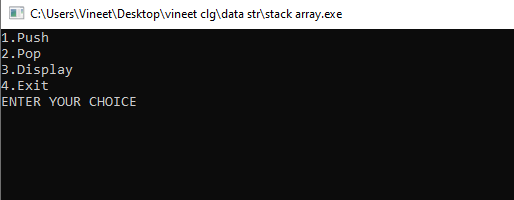
return 0;

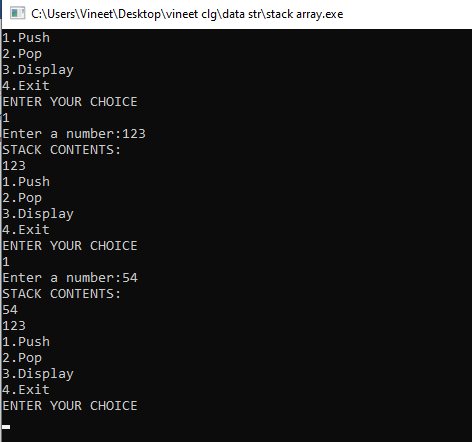
}

}

}

**OUTPUT : -**





***EXPERIMENT – 6***

**Objectives :-** Menu Driven Program For Queue Simulation Using Array .

**CODE : -**

#include<stdio.h>

#define N 5

int queue[N];

int front=-1,rear=-1;

void enqueue(int x)

{

if((rear+1)%N==front)

{

printf("Queue is full");

}

else if(front==-1 && rear==-1)

{

front=rear=0;

queue[rear]=x;

}

else

{

rear=(rear+1)%N;

queue[rear]=x;

}

}

void dequeue()

{

if(front==-1&&rear==-1)

{

printf("Queue is empty");

}

else if(front==rear)

{

printf("dequeued element is %d",queue[front]);

front=rear=-1;

}

else

{

printf("Dequeued element is %d", queue[front]);

front=(front+1)%N;

}

}

void display()

{

int i=front;

if(front==-1 && rear==-1)

{

printf("Queue is empty");

}

else

{

printf("queue is");

while(i!=rear)

{

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

i=(i+1)%N;

}

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

}

}

int main()

{

int ch,a,x;

do

{

printf("ENTER YOUR CHOICE:\n 1.Enqueue \n 2.Dequeue \n 3.Display \n");

scanf("%d", &ch);

switch(ch)

{

case 1:

printf("Enter the value you want to enqueue");

scanf("%d", &x);

enqueue(x);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

default:

printf("Invalid Choice");

}

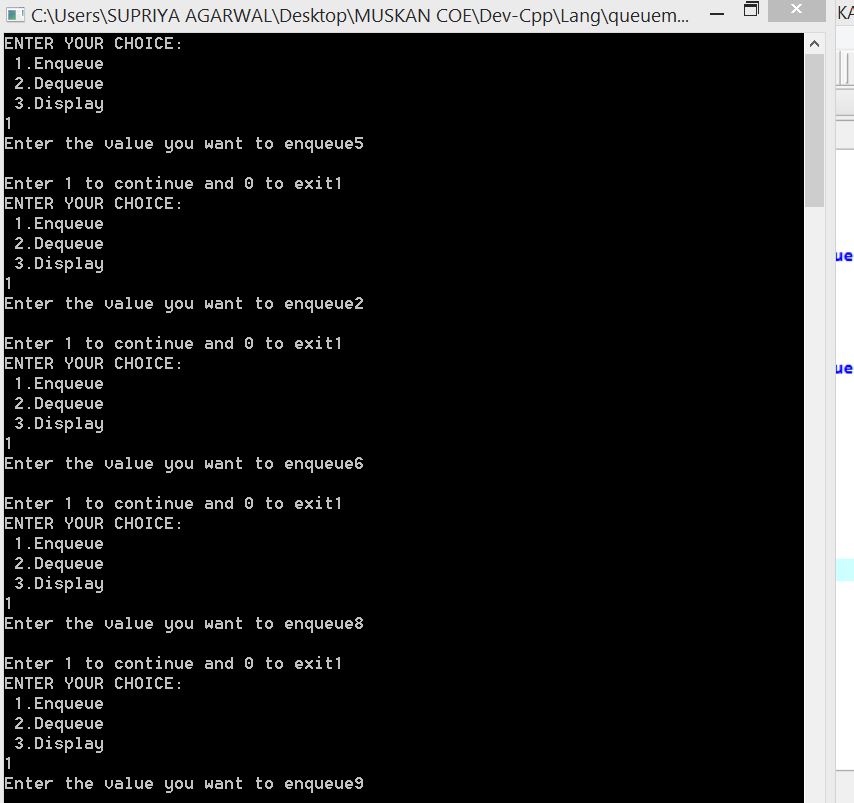
printf("\nEnter 1 to continue and 0 to exit");

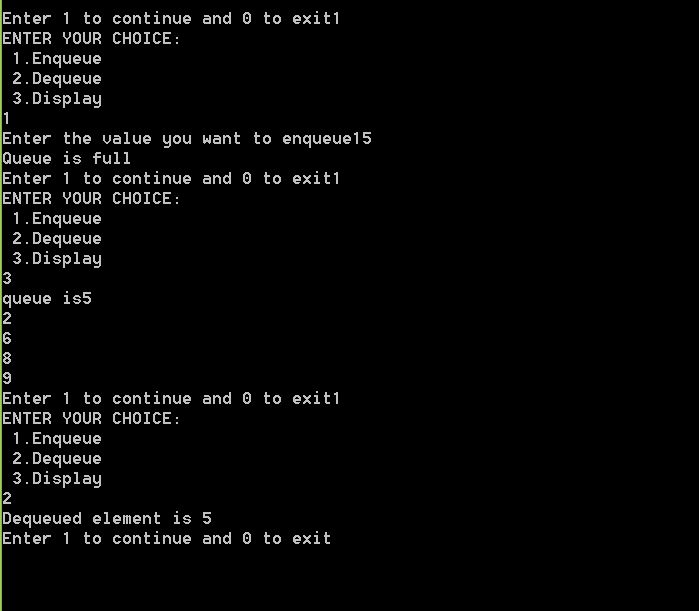
scanf("%d", &a);

}while(a==1);

return 0;

}





***EXPERIMENT – 7***

**Objectives :-** To check if given string contains balanced parenthesis e.g. (()()) is balanced.

**CODE : -**

#include <stdio.h>

#include <string.h>

int top;

void check(char str[], int n, char stack[])

{

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

{

if (str[i] == '(') // push

{

top = top + 1;

stack[top] = '(';

}

if (str[i] == ')' && stack[top] == '(') //pop

{

top = top - 1;

}

}

if (top == -1) // stack is empty

{

printf("string is balances\n");

}

else

{

printf("string is unbalanced:\n");

}

}

int main()

{

// balanced parenthesis string

char str[] = "(a+(b-c))";

//unbalanced parenthesis string

char str1[] = "((a+b)";

char stack[15];

top = -1;

check(str, 9, stack); //passing balance to string

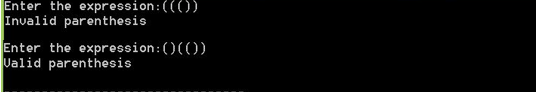
top = -1;

check(str1, 6, stack); // passing unbalanced to string

return 0;

}

**OUTPUT : -**



***EXPERIMENT – 8***

**Objectives :-** USING ARRAY AND WRITE PROGRAM TO ADD AND MULTIPLY TWO POLYNOMIALS.

**CODE: -**

#include<stdio.h>

#include<stdlib.h>

struct node{

int coef;

int expr ;

struct node\*next;

};

struct node\*Getnode(){

struct node \*p;

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

return p;

}

void Insbeg(struct node \*\*list,int c, int e){

struct node \*temp;

temp=Getnode();

temp->coef=c;

temp->expr=e;

temp->next=\*list;

\*list=temp;

}

void Inend(struct node \*\*list,int c,int e){

struct node \*temp,\*p;

temp= \*list;

if(\*list==NULL)

Insbeg(&(\*list),c,e);

else

{

while(temp->next!=NULL)

temp=temp->next;

p=Getnode();

p->coef=c;

p->expr=e;

p->next=NULL;

temp->next=p;

}

}

void Traverse(struct node \*list)

{

struct node \*t;

t=list;

while(t!=NULL){

printf("\t %dX%d +",t->coef,t->expr);

t=t->next;

}

}

struct node \* addPolynomial(struct node \*poly1,struct node \*poly2){

struct node \*poly3=NULL;

struct node \*p,\*q;

p=poly1;

q=poly2;

while(p!=NULL && q!=NULL){

if(p->expr==q->expr){

Inend(&poly3,p->coef+q->coef,p->expr);

p=p->next;

q=q->next;

}

else{

if(p->expr>q->expr)

{

Inend(&poly3,p->coef,p->expr);

p=p->next;

}

else

{

Inend(&poly3,q->coef,q->expr);

q=q->next;

}

}

}

while(p!=NULL){

Inend(&poly3,p->coef,p->expr);

p=p->next;

}

while(q!=NULL){

Inend(&poly3,q->coef,q->expr);

q=q->next;

}

return poly3;

}

int main() {

struct node \*start1,\*start2,\*start3;

start1=NULL;

start2=NULL;

start3=NULL;

int x;

Inend(&start1,3,8);

Inend(&start1,5,7);

Inend(&start1,2,6);

Inend(&start1,8,4);

printf("\n First Polynomial is:= ");

Traverse(start1);

Inend(&start2,4,8);

Inend(&start2,5,6);

Inend(&start2,7,2);

Inend(&start2,3,0);

printf("\n Second Polynomial is:= ");

Traverse(start2);

start3=addPolynomial(start1,start2);

printf("\n\n");

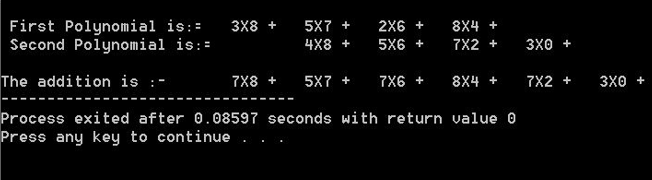
printf("The addition is :- ");

Traverse(start3);

return 0;

}

**OUTPUT: -**



***EXPERIMENT – 9***

**Objectives:-** Infix To Postfix Conversion Using Stack.

**CODE: -**

#include<stdio.h>

#include<ctype.h>

char stack[100];

int top = -1;

void push(char x)

{

stack[++top] = x;

}

char pop()

{

if(top == -1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x == '(')

return 0;

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

return 1;

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

return 2;

return 0;

}

int main()

{

char exp[100];

char \*e, x;

printf("Enter the expression : ");

scanf("%s",exp);

printf("\n");

e = exp;

while(\*e != '\0')

{

if(isalnum(\*e))

printf("%c ",\*e);

else if(\*e == '(')

push(\*e);

else if(\*e == ')')

{

while((x = pop()) != '(')

printf("%c ", x);

}

else

{

while(priority(stack[top]) >= priority(\*e))

printf("%c ",pop());

push(\*e);

}

e++;

}

while(top != -1)

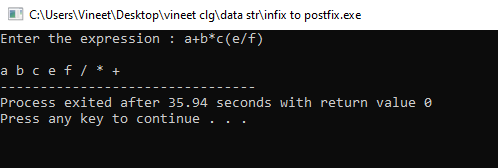
{

printf("%c ",pop());

}return 0;

}

**OUTPUT: -**



EXPERIMENT – 10

**Objectives :-**  SPARSE MATRIX ADDITION

**CODE : -**

#include<stdio.h>

#define size 20

struct sparse

{

int rows , columns , d ;

int row[size] , col[size], value[size];

};

void readMatrix(struct sparse \*P)

{

int i,a,b,c;

printf("Enter number of rows: ");

scanf("%d",&P->rows);

printf("Enter number of columns: ");

scanf("%d",&P->columns);

printf("Enter number of nonzero elements: ");

scanf("%d",&P->d);

for(i=0;i<P->d;i++)

{

printf("Enter row index, column index and value : ");

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

P->row[i]=a;

P->col[i]=b;

P->value[i]=c;

}

}

void printMatrix(struct sparse Q)

{

int i,j,k=0;

for(i=0;i<Q.rows;i++)

{

for(j=0;j<Q.columns;j++)

{

if((i==Q.row[k])&&(j==Q.col[k]))

{

printf("%6d",Q.value[k]);

k++;

}

else

printf("%6d",0);

}

printf("\n");

}

}

struct sparse addMatrix(struct sparse A, struct sparse B)

{

struct sparse C;

int i,j,k;

i=0;

j=0;

k=0;

while(i<A.d && j<B.d)

{

if(A.row[i]==B.row[j]&&A.col[i]==B.col[j])

{

if(A.value[i]+B.value[j]!=0)

{

C.row[k]=A.row[i];

C.col[k]=A.col[i];

C.value[k]=A.value[i]+B.value[j];

k++;

}

i++;

j++;

}

if((A.row[i]<B.row[j])||(A.row[i]==B.row[j]&&A.col[i]<B.col[j]))

{

C.row[k]=A.row[i];

C.col[k]=A.col[i];

C.value[k]=A.value[i];

i++;

k++;

}

else

{

C.row[k]=B.row[j];

C.col[k]=B.col[j];

C.value[k]=B.value[j];

j++;

k++;

}

}

while(i<A.d)

{

C.row[k]=A.row[i];

C.col[k]=A.col[i];

C.value[k]=A.value[i];

i++;

k++;

}

while(j<B.d)

{

C.row[k]=B.row[j];

C.col[k]=B.col[j];

C.value[k]=B.value[j];

j++;

k++;

}

C.rows=A.rows;

C.columns=A.columns;

C.d=k;

return C;

}

int main()

{

struct sparse A,B;

printf("For First matrix:\n\n");

readMatrix(&A);

printf("\n");

printf("For Second matrix:\n\n");

readMatrix(&B);

printf("\n");

printf("\n Elements Of Matrix A:\n");

printMatrix(A);

printf("\n Elements Of Matrix B:\n");

printMatrix(B);

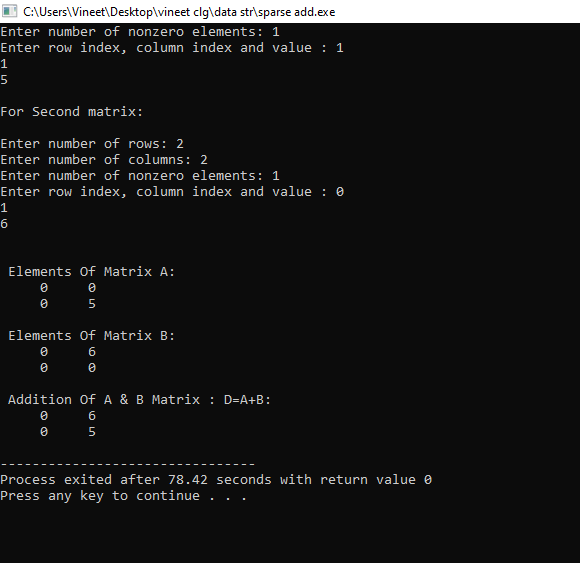
printf("\n Addition Of A & B Matrix : D=A+B:\n");

printMatrix(addMatrix(A,B));

return 0;

}

**OUTPUT : -**



***EXPERIMENT – 11***

**Objectives :-** Sparse Matric Transpose And Multiplication

**CODE : -**

#include<stdio.h>

#include<stdlib.h>

#define size 100

struct sparse

{

int nrows,ncols,nz;

int row[size],col[size],val[size];

};

void readMatrix(struct sparse \*s)

{

int i;

printf("Enter the dimensions of the matrix (row\*col)\n");

scanf("%d %d",&s->nrows,&s->ncols);

printf("Enter the no. of non-zero elements\n");

scanf("%d",&s->nz);

printf("Enter the non-zero elements(row col val)\n");

for(i=0;i<s->nz;i++)

{

scanf("%d %d %d",&s->row[i],&s->col[i],&s->val[i]);

}

}

void printMatrix(struct sparse s)

{

int i,j,k=0;

for(i=0;i<s.nrows;i++)

{

for(j=0;j<s.ncols;j++)

{

if(k<s.nz && i==s.row[k] && j==s.col[k])

printf("%d ",s.val[k++]);

else

{

printf("%d ",0);

}

}

printf("\n");

}

}

struct sparse Transpose(struct sparse A)

{

struct sparse B;

int C[A.ncols],t[A.ncols];

int i;

B.nrows=A.ncols;

B.ncols=A.nrows;

B.nz=A.nz;

for(i=0;i<A.ncols;i++)

{

C[i]=0;

}

for(i=0;i<A.nz;i++)

{

C[A.col[i]]++;

}

t[0]=0;

for(i=1;i<A.ncols;i++)

{

t[i]=t[i-1]+C[i-1];

}

for(i=0;i<A.nz;i++)

{

B.row[t[A.col[i]]]=A.col[i];

B.col[t[A.col[i]]]=A.row[i];

B.val[t[A.col[i]]]=A.val[i];

t[A.col[i]]++;

}

return B;

}

struct sparse addMatrix(struct sparse A,struct sparse B)

{

struct sparse C;

int i,j,k;

i=j=k=0;

while(i<A.nz && j<B.nz)

{

if(A.row[i]==B.row[j] && A.col[i]==B.col[j])

{

if(A.val[i]+B.val[j]!=0)

{

C.row[k]=A.row[i];

C.col[k]=A.col[i];

C.val[k]=A.val[i]+B.val[j];

k++;

}

i++;

j++;

}

if((A.row[i]<B.row[j])|| (A.row[i]==B.row[j] && A.col[i]<B.col[j]))

{

C.row[k]=A.row[i];

C.col[k]=A.col[i];

C.val[k]=A.val[i];

i++;

k++;

}

else

{

C.row[k]=B.row[j];

C.col[k]=B.col[j];

C.val[k]=B.val[j];

j++;

k++;

}

}

while(i<A.nz)

{

C.row[k]=A.row[i];

C.col[k]=A.col[i];

C.val[k]=A.val[i];

i++;

k++;

}

while(j<B.nz)

{

C.row[k]=B.row[j];

C.col[k]=B.col[j];

C.val[k]=B.val[j];

j++;

k++;

}

C.nrows=A.nrows;

C.ncols=A.ncols;

C.nz=k;

return C;

}

void mulMatrix(struct sparse a,struct sparse m)

{

if(a.ncols!=m.nrows)

{

printf("Matrices can't be multiplied");

return;

}

struct sparse b=Transpose(m);

int res[size][size]={};

int i=0,j;

while(i<a.nz)

{

j=0;

while(j<b.nz)

{

if(a.col[i]==b.col[j])

res[a.row[i]][b.row[j]]+=a.val[i]\*b.val[j];

j++;

}

i++;

}

i=0;

while(i<a.nrows)

{

j=0;

while(j<b.nrows)

{

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

j++;

}

printf("\n");

i++;

}

}

int main()

{

struct sparse A,B,C;

readMatrix(&A);

readMatrix(&B);

printf("MATRIX A\n");

printMatrix(A);

printf("MATRIX B\n");

printMatrix(B);

printf("The Addition of Matrix A aNd B\n");

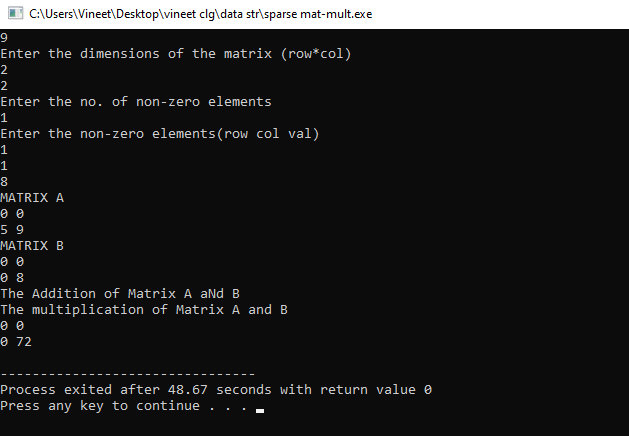
addMatrix(A,B);

printf("The multiplication of Matrix A and B\n");

mulMatrix(A,B);

}

**OUTPUT : -**



***EXPERIMENT – 12***

**Objectives :-** Menu Driven Simple Linked List Without Head Node . Insert (at Start , End , After ), Delete , Search , Count Operations.

**CODE : -**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int info;

struct node \*link;

};

struct node \*createlist(struct node \*start);

void displaylist(struct node \*start);

void countnodes(struct node \*start);

void search(struct node \*start,int x);

struct node \*insertinbeginning(struct node \*start,int data);

void insertatend(struct node \*start,int data);

void insertafter(struct node \*start,int data,int x);

struct node \*insertbefore(struct node \*start,int data,int x);

struct node \*insertatposition(struct node \*start,int data,int k);

struct node \*deletenode(struct node \*start,int data);

struct node \*reverselist(struct node \*start);

main()

{

struct node \*start=NULL;

int choice,data,x,k;

start=createlist(start);

while(1)

{

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

printf("\t\t\t 1.Display list \n");

printf("\t\t\t 2.Count no of nodes \n");

printf("\t\t\t 3.Search for an element \n");

printf("\t\t\t 4.Add to empty list/add at the beginning \n");

printf("\t\t\t 5.Add a node at the end of the list \n");

printf("\t\t\t 6.Add a after a specefied node \n");

printf("\t\t\t 7.Add a before a specefied node \n");

printf("\t\t\t 8.Add a node at a given position \n");

printf("\t\t\t 9.Delete a node \n");

printf("\t\t\t 10.Reverse the list \n");

printf("\t\t\t 11.Quit \n");

printf("ENTER YOUR CHOICE: ");

scanf("%d",&choice);

if(choice==11)

break;

switch(choice)

{

case 1:

displaylist(start);

break;

case 2:

countnodes(start);

break;

case 3:

printf("Enter element you want to search");

scanf("%d",&data );

search(start,data);

break;

case 4:

printf("Enter an element you want to insert: ");

scanf("%d",&data);

start=insertinbeginning(start,data);

break;

case 5:

printf("Enter element you want to enter: ");

scanf("%d",&data);

insertatend(start,data);

break;

case 6:

printf("Enter element you want to enter: ");

scanf("%d",&data);

printf("enter the element after which you want to insert: ");

scanf("%d",&x);

insertafter(start,data,x);

break;

case 7:

printf("Enter element you want to enter: ");

scanf("%d",&data);

printf("enter the element before which you want to insert: ");

scanf("%d",&x);

start=insertbefore(start,data,x);

break;

case 8:

printf("Enter element you want to enter: ");

scanf("%d",&data);

printf("enter the position at which to insert: ");

scanf("%d",&k);

start=insertatposition(start,data,k);

break;

case 9:

printf("Enter the element you want to delete: ");

scanf("%d", &x);

start=deletenode(start,x);

break;

case 10:

start=reverselist(start);

break;

default:

printf(" INVALID CHOICE ");

}

}

}

struct node \*createlist(struct node \*start)

{

int i,n,data;

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

scanf("%d",&n);

if(n==0)

return start;

printf("\nEnter the first element to be inserted \n");

scanf("%d",&data);

start=insertinbeginning(start,data);

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

{

printf("Enter the next elements to be inserted \n");

scanf("%d",&data);

insertatend(start,data);

}

return start;

}

struct node \*insertinbeginning(struct node \*start,int data)

{

struct node \*temp;

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

temp->info=data;

temp->link=start;

start=temp;

return start;

}

void insertatend(struct node \*start,int data)

{

struct node \*p,\*temp;

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

temp->info=data;

p=start;

while(p->link!=NULL)

p=p->link;

p->link=temp;

temp->link=NULL;

}

void displaylist(struct node \*start)

{

struct node \*p;

if(start==NULL)

{

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

return;

}

printf("List is: ");

p=start;

while(p!=NULL)

{

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

p=p->link;

}

printf(" \n ");

}

void countnodes(struct node \*start)

{

int n=0;

struct node \*p;

p=start;

while(p!=NULL)

{

n++;

p=p->link;

}

printf("No of nodes in the list are: %d \n",n);

}

void insertafter(struct node \*start,int data,int x)

{

struct node \*temp,\*p;

p=start;

while(p!=NULL)

{

if(p->info==x)

break;

p=p->link;

}

if(p==NULL)

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

else

{

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

temp->info=data;

temp->link=p->link;

p->link=temp;

}

}

struct node \*insertbefore(struct node \*start,int data,int x)

{

struct node \*temp,\*p;

if(start==NULL)

{

printf("list is empty \n");

return start;

}

if(x==start->info)

{

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

temp->info=data;

temp->link=start;

start=temp;

return start;

}

p=start;

while(p->link!=NULL)

{

if(p->link->info==x)

break;

p=p->link;

}

if(p->link==NULL)

printf("%d not found in the list \n",x);

else

{

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

temp->info=data;

temp->link=p->link;

p->link=temp;

}

return start;

}

struct node \*insertatposition(struct node \*start,int data,int k)

{

struct node \*temp,\*p;

int i;

if(k==1)

{

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

temp->info=data;

temp->link=start;

start=temp;

return start;

}

p=start;

for(i=1; i<k-1 && p!=NULL; i++)

p=p->link;

if(p==NULL)

printf("you can insert only upto %dth position \n\n",i);

else

{

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

temp->info=data;

temp->link=p->link;

p->link=temp;

}

return start;

}

void search(struct node \*start,int x)

{

struct node \*p;

int position=1;

p=start;

while(p!=NULL)

{

if(p->info==x)

break;

position++;

p=p->link;

}

if(p==NULL)

printf("%d not foung on the list \n", x);

else

printf("%d was found at %d positon: \n",x,position);

}

struct node \*deletenode(struct node \*start, int x)

{

struct node \*temp,\*p;

if(start==NULL)

{

printf("List is empty\n");

return start;

}

if(start->info==x)

{

temp=start;

start=start->link;

free(temp);

return start;

}

p=start;

while(p->link!=NULL)

{

if(p->link->info==x)

break;

p=p->link;

}

if(p->link==NULL)

printf("Element %d is not in list \n\n",x);

else

{

temp=p->link;

p->link=temp->link;

free(temp);

}

return start;

}

struct node \*reverselist(struct node \*start)

{

struct node \*prev,\*ptr,\*next;

prev=NULL;

ptr=start;

while(ptr!=NULL)

{

next=ptr->link;

ptr->link=prev;

prev=ptr;

ptr=next;

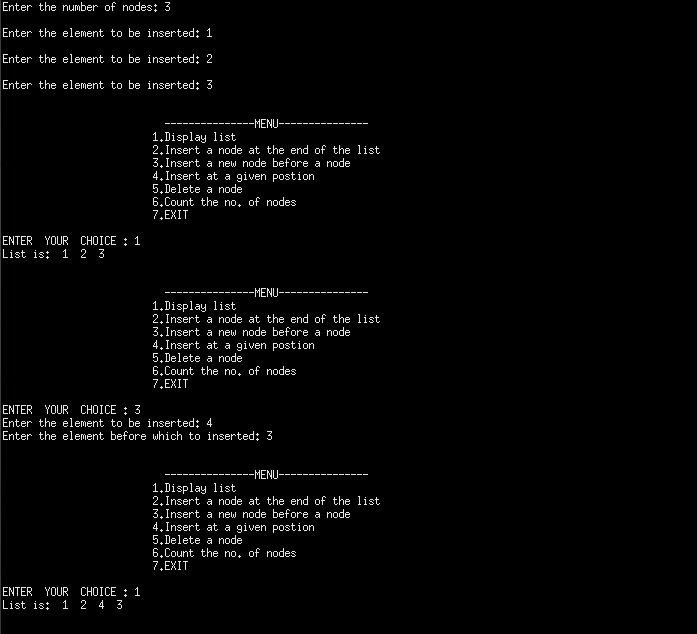
}

start=prev;

return start;

}

**OUTPUT : -**





***EXPERIMENT – 13***

**Objective :-** MENU DRIVEN SIMPLE LINKED LIST WITH HEAD NODE. INSERT(START, END, AFTER), DELETE, SEARCH,COUNT OPERATIONS.

**CODE : -**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int info;

struct node \*link;

};

void createlist(struct node \*head);

void displaylist(struct node \*head);

void insertatend(struct node \*head,int data);

void insertbefore(struct node \*head,int data,int x);

void insertatposition(struct node \*head,int data,int k);

void deletenode(struct node \*head,int data);

void countnodes(struct node \*head);

main()

{

int x,choice,k,data;

struct node \*head;

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

head->info=0;

head->link=NULL;

createlist(head);

while(1)

{

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

printf("\t\t\t 1.Display list \n ");

printf("\t\t\t 2.Insert a node at the end of the list \n ");

printf("\t\t\t 3.Insert a new node before a node \n ");

printf("\t\t\t 4.Insert at a given postion \n ");

printf("\t\t\t 5.Delete a node \n ");

printf("\t\t\t 6.Count the no. of nodes \n ");

printf("\t\t\t 7.EXIT \n ");

printf("\nENTER YOUR CHOICE : ");

scanf("%d",&choice);

if(choice==7)

break;

switch(choice)

{

case 1:

displaylist(head);

break;

case 2:

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

scanf("%d",&data);

insertatend(head,data);

break;

case 3:

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

scanf("%d",&data);

printf("Enter the element before which to inserted: ");

scanf("%d",&x);

insertbefore(head,data,x);

break;

case 4:

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

scanf("%d",&data);

printf("Enter the postion before which to insert: ");

scanf("%d",&k);

insertatposition(head,data,k);

break;

case 5:

printf("Enter the element to delete: ");

scanf("%d",&data);

deletenode(head,data);

break;

case 6:

countnodes(head);

default:

printf("Invalid choice");

}

}

}

void insertatend(struct node \*head, int data)

{

struct node \*p,\*temp;

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

temp->info=data;

p=head;

while(p->link!=NULL)

p=p->link;

p->link=temp;

temp->link=NULL;

}

void createlist(struct node \*head)

{

int i,n,data;

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

scanf("%d",&n);

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

{

printf("\nEnter the element to be inserted: ");

scanf("%d",&data);

insertatend(head,data);

}

}

void insertbefore(struct node \*head,int data,int x)

{

struct node \*temp,\*p;

p=head;

while(p->link!=NULL)

{

if(p->link->info==x)

break;

p=p->link;

}

if(p->link==NULL)

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

else

{

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

temp->info=data;

temp->link=p->link;

p->link=temp;

}

}

void displaylist(struct node \*head)

{

struct node \*p;

if(head->link==NULL)

{

printf("List is empty\n");

return;

}

p=head->link;

printf("List is: ");

while(p!=NULL)

{

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

p=p->link;

}

printf("\n");

}

void insertatposition(struct node \*head,int data,int k)

{

struct node \*temp,\*p;

int i;

p=head;

for(i=1; i<=k-1 && p!=NULL; i++)

p=p->link;

if(p==NULL)

printf("You can insert only upto %dth position\n\n",i-1);

else

{

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

temp->info=data;

temp->link=p->link;

p->link=temp;

}

}

void deletenode(struct node \*head, int data)

{

struct node \*temp,\*p;

p=head;

while(p->link!=NULL)

{

if(p->link->info==data)

break;

p=p->link;

}

if(p->link==NULL)

printf("Element %d not found \n",data);

else

{

temp=p->link;

p->link=temp->link;

free(temp);

}

}

void countnodes(struct node \*head)

{

int n=0;

struct node \*p;

p=head;

while(p!=NULL)

{

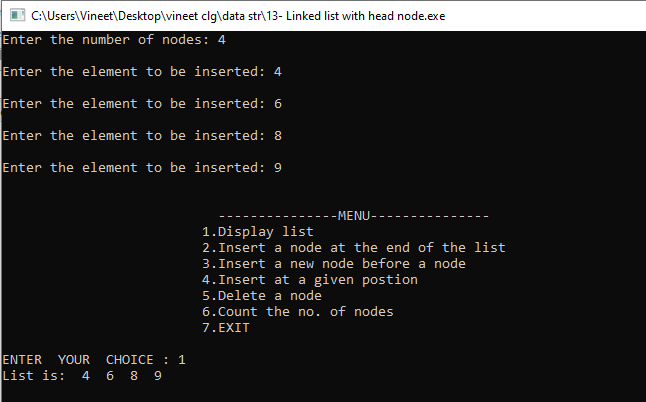
n++;

p=p->link;

}

printf("No of nodes in the list are: %d \n",n-1);

}

**OUTPUT:-** 

***EXPERIMENT – 14***

**Objective :-** Maintain A sorted Linked List , Implementation Insert, Delete , Search , Count .

**CODE : -**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int info;

struct node \*link;

};

struct node \*createlist(struct node \*start);

void displaylist(struct node \*start);

void countnodes(struct node \*start);

void search(struct node \*start,int x);

struct node \*insertinbeginning(struct node \*start,int data);

void insertatend(struct node \*start,int data);

void insertafter(struct node \*start,int data,int x);

struct node \*insertbefore(struct node \*start,int data,int x);

struct node \*insertatposition(struct node \*start,int data,int k);

struct node \*deletenode(struct node \*start,int data);

struct node \*reverselist(struct node \*start);

main()

{

struct node \*start=NULL;

int choice,data,x,k;

start=createlist(start);

while(1)

{

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

printf("\t\t\t 1.Display list \n");

printf("\t\t\t 2.Count no of nodes \n");

printf("\t\t\t 3.Search for an element \n");

printf("\t\t\t 4.Add to empty list/add at the beginning \n");

printf("\t\t\t 5.Add a node at the end of the list \n");

printf("\t\t\t 6.Add a after a specefied node \n");

printf("\t\t\t 7.Add a before a specefied node \n");

printf("\t\t\t 8.Add a node at a given position \n");

printf("\t\t\t 9.Delete a node \n");

printf("\t\t\t 10.Quit \n");

printf("ENTER YOUR CHOICE: ");

scanf(" %d ", &choice);

if(choice==10)

break;

switch(choice)

{

case 1:

displaylist(start);

break;

case 2:

countnodes(start);

break;

case 3:

printf("Enter element you want to search");

scanf(" % d",&data );

search(start,data);

break;

case 4:

printf("Enter an element you want to insert: ");

scanf(" %d ",&data);

start=insertinbeginning(start,data);

break;

case 5:

printf("Enter element you want to enter: ");

scanf("%d",&data);

insertatend(start,data);

break;

case 6:

printf("Enter element you want to enter: ");

scanf("%d",data);

printf("enter the element after which you want to insert: ");

scanf("%d",&x);

insertafter(start,data,x);

break;

case 7:

printf("Enter element you want to enter: ");

scanf("%d",data);

printf("enter the element before which you want to insert: ");

scanf("%d",&x);

start=insertbefore(start,data,x);

break;

case 8:

printf("Enter element you want to enter: ");

scanf("%d",data);

printf("enter the position at which to insert: ");

scanf("%d",&k);

start=insertatposition(start,data,k);

break;

case 9:

printf("Enter the element you want to delete: ");

scanf("%d", &x);

start=deletenode(start,x);

break;

default:

printf(" INVALID CHOICE ");

}

}

}

struct node \*createlist(struct node \*start)

{

int i,n,data;

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

scanf("%d",&n);

if(n==0)

return start;

printf("\nEnter the first element to be inserted \n");

scanf("%d",&data);

start=insertinbeginning(start,data);

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

{

printf("Enter the next elements to be inserted \n ");

scanf("%d",&data);

insertatend(start,data);

}

return start;

}

struct node \*insertinbeginning(struct node \*start,int data)

{

struct node \*temp;

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

temp->info=data;

temp->link=start;

start=temp;

return start;

}

void insertatend(struct node \*start,int data)

{

struct node \*p,\*temp;

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

temp->info=data;

p=start;

while(p->link!=NULL)

p=p->link;

p->link=temp;

temp->link=NULL;

}

void displaylist(struct node \*start)

{

struct node \*p;

if(start==NULL)

{

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

return;

}

printf("List is: ");

p=start;

while(p!=NULL)

{

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

p=p->link;

}

printf(" \n ");

}

void countnodes(struct node \*start)

{

int n=0;

struct node \*p;

p=start;

while(p!=NULL)

{

n++;

p=p->link;

}

printf("No of nodes in the list are: %d ",n)

}

void insertafter(struct node \*start,int data,int x)

{

struct node \*temp,\*p;

p=start;

while(p!=NULL)

{

if(p->info==x)

break;

p=p->link;

}

if(p==NULL)

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

else

{

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

temp->info=data;

temp->link=p->link;

p->link=temp;

}

}

struct node \*insertbefore(struct node \*start,int data,int x)

{

struct node \*temp,\*p;

if(start==NULL)

{

printf("list is empty \n");

return start;

}

if(x==start->info)

{

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

temp->info=data;

temp->link=start;

start=temp;

return start;

}

p=start;

while(p->link!=NULL)

{

if(p->link->info==x)

break;

p=p->link;

}

if(p->link==NULL)

printf("%d not found in the list ",x);

else

{

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

temp->info=data;

temp->link=p->link;

p->link=temp;

}

return start;

}

struct node \*insertatposition(struct node \*start,int data,int k)

{

struct node \*temp,\*p;

int i;

if(k==1)

{

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

temp->info=data;

temp->link=start;

start=temp;

return start;

}

p=start;

for(i=1; i<k-1 && p!=NULL; i++)

p=p->link;

if(p==NULL)

printf("you can insert only upto %dth position \n\n",i);

else

{

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

temp->info=data;

temp->link=p->link;

p->link=temp;

}

return start;

}

void search(struct node \*start,int x)

{

struct node \*p;

int position=1;

p=start;

while(p!=NULL)

{

if(p->info==x)

break;

position++;

p=p->link;

}

if(p==NULL)

printf("%d not foung on the list ", x);

else

printf("%d was found at %d positon: ",x,position);

}

struct node \*deletenode(struct node \*start, int x)

{

struct node \*temp,\*p;

if(start==NULL)

{

printf("List is empty\n");

return start;

}

if(start->info==x)

{

temp=start;

start=start->link;

free(temp);

return start;

}

p=start;

while(p->link!=NULL)

{

if(p->link->info==x)

break;

p=p->link;

}

if(p->link==NULL)

printf("Element %d is not in list \n\n",x);

else

{

temp=p->link;

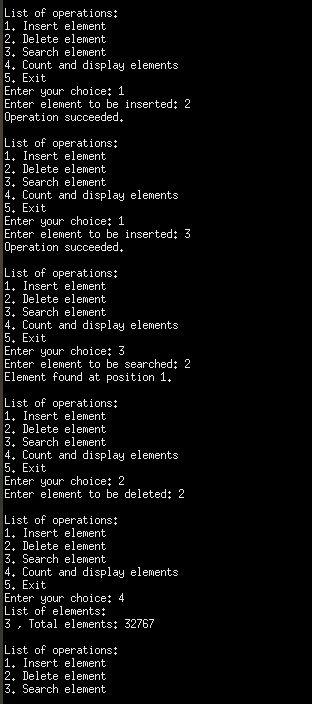
p->link=temp->link;

free(temp);

}

return start;

}

**Output :-**

***EXPERIMENT – 15***

**OBJECTIVES :-** IMPLEMENTATION OF INSERT, DELETE, SEARCH, INORDER, PREORDER, POSTORDER, HEIGHT,COUNTNODES, FUNCTIONS FOR BINARY SEARCH TREE.

**CODE : -**

#include <iostream>

using namespace std;

struct Node{

int data;

Node \*left, \*right;

};

Node\* insert(Node \*root, int value){

if(root == NULL){

Node \*k = new Node;

k->data = value;

k->left = NULL;

k->right = NULL;

}

if(value > root->data){

root->right = insert(root->right, value);

}

else{

root->left = insert(root->left, value);

}

return root;

}

Node\* minNode(Node \*node){

Node \*current = node;

while(current != NULL && current->left != NULL){

current = current->left;

}

return current;

}

int count(Node \*node){

if(node == NULL)

return 0;

return count(node->left) + count(node->right) + 1;

}

bool search(Node \*node, int value){

if(node->data == value || node == NULL){

return node;

}

if(node->data < value){

return search(node->right, value);

}

return search(node->left, value);

}

int height(Node \*node){

if(node == NULL)

return 0;

return max(height(node->left), height(node->right)) + 1;

}

void preorder(Node \*node){

if(node == NULL)

return;

cout<<node->data<<" ";

preorder(node->left);

preorder(node->right);

}

void postorder(Node \*node){

if(node == NULL)

return;

postorder(node->left);

postorder(node->right);

cout<<node->data<<" ";

}

void inorder(Node \*node){

if(node == NULL)

return;

inorder(node->left);

cout<<node->data<<" ";

inorder(node->right);

}

Node\* deleteNode(Node \*node, int element){

if(node == NULL){

return node;

}

if(element < node->data){

node->left = deleteNode(node->left,element);

}

else if(element > node->data){

node->right = deleteNode(node->right,element);

}

else{

if(node->left == NULL){

Node \*t = node->right;

return t;

}

else if(node->right == NULL){

Node \*t = node->left;

return t;

}

Node \*temp = minNode(node->right);

node->data = temp->data;

node->right = deleteNode(node->right, temp->data);

}

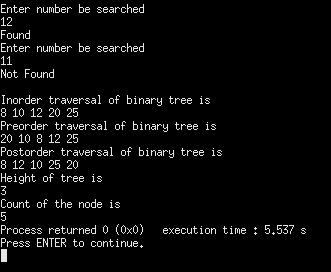
}

int main(){

return 0;

}

**OUTPUT :-**

****

***EXPERIMENT – 16***

**OBJECTIVES :-** IMPLEMENT PRIORITY QUEUE USING MAXHEAP: INSERTQ(), DELETEQ(), DISPLAYHEAP() ARE THE FUNCTIONS TO BE IMPLEMENTED.

**CODE : -**

#include <stdio.h>

#include <stdlib.h>

int size=0;

void swap(int \*x,int \*y)

{

int temp= \*y;

\*y = \*x;

\*x =temp;

}

//function to create a heap

void heapify(int array[],int size,int i)

{

if(size==1)

{

printf("Only one element in the heap\n");

}

else{

int largest=i;

int l= 2\*i+1;

int r=2\*i+2;

if(l<size&&array[l]>array[largest])

largest=l;

if(r<size&&array[r]>array[largest])

largest=r;

if(largest!=i)

{

swap(&array[i],&array[largest]);

heapify(array,size,largest);

}

}

}

void insert(int array[],int data)

{

if(size==0)

{

array[0]=data;

size+=1;

}

else

{

array[size]=data;

size+=1;

//to convert array into heap

for(int i=size/2-1;i>=0;i--)

{

heapify(array,size,i);

}

}

}

void deletee(int array[],int data)

{

int i;

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

{

if(data==array[i])

break;

}

swap(&array[i],&array[size-1]);

size-=1;

for(int i=size/2-1;i>=0;i--){

heapify(array,size,i);

}

}

void printarray(int array[],int size)

{

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

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

printf("\n");

}

int main()

{

int array[10],num,data;

int choice;

while(1)

{

printf("1.Insert\n");

printf("2.Delete\n");

printf("3.Display\n");

printf("Enter your choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

printf("Enter the number you want to insert:");

scanf("%d",&num);

insert(array,num);

break;

}

case 2:

{

printf("Enter the number you want to delete:");

scanf("%d",&data);

deletee(array,data);

break;

}

case 3:

{

printf("The elements of the heap are:");

printarray(array,size);

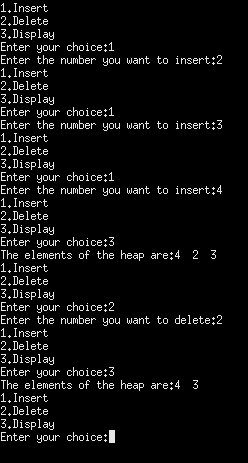
}

}

}

}

**OUTPUT: -**



***EXPERIMENT – 17***

**Objectives :-** Design Heapsort () to sort an array in Ascending Order .

**CODE : -**

#include <stdio.h>

#include<stdlib.h>

void swap(int \*x, int \*y)

{

int temp = \*x;

\*x = \*y;

\*y = temp;

}

void heapify(int arr[], int n, int i) {

// Find largest among root, left child and right child

int largest = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < n && arr[l] > arr[largest])

largest = l;

if (r < n && arr[r] > arr[largest])

largest = r;

if (largest != i) {

swap(&arr[i], &arr[largest]);

heapify(arr, n, largest);

}

}

// Main function to do heap sort

void heapSort(int arr[], int n)

{

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

// Heap sort

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

swap(&arr[0], &arr[i]);

// Heapify root element to get highest element at root again

heapify(arr, i, 0);

}

}

// Print an array

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

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

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

printf("\n");

}

// Driver code

int main() {

int array[10],n;

printf("Enter the number of elements that you want to enter:");

scanf("%d",&n);

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

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

{

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

}

heapSort(array, n);

printf("Sorted array is \n");

printArray(array, n);

}

**OUTPUT :-**

