Double ended queue :

#include&lt;iostream&gt;

**using** **namespace** std;

#define MAX\_size 10     // Maximum size of array or Dequeue

// Deque class

**class** Deque

{

**int**  array[MAX\_size];

**int**  front;

**int**  rear;

**int**  size;

**public** :

Deque(**int** size) {

front = -1;

rear = 0;

**this**-&gt;size = size;

    }

    // Operations on Deque:

**void**  insertfront(**int** key);

**void**  insertrear(**int** key);

**void**  deletefront();

**void**  deleterear();

**int**  getFront();

**int**  getRear();

    // Check if Deque is full

**bool**  isFull(){

**return** ((front == 0 &amp;&amp; rear == size-1)||front == rear+1);

    }

    // Check if Deque is empty

**bool**  isEmpty(){

**return** (front == -1);

    }

};

// Insert an element at front of the deque

**void** Deque::insertfront(**int** key)

{

**if** (isFull())  {

cout &lt;&lt; "Overflow!!\n" &lt;&lt; endl;

**return**;

    }

    // If queue is initially empty,set front=rear=0; start of deque

**if** (front == -1)  {

front = 0;

rear = 0;

    }

**else** **if** (front == 0)              // front is first position of queue

front = size - 1 ;

**else** // decrement front 1 position

front = front-1;

array[front] = key ;            // insert current element into Deque

}

// insert element at the rear end of deque

**void** Deque ::insertrear(**int** key)

{

**if** (isFull()) {

cout &lt;&lt; " Overflow!!\n " &lt;&lt; endl;

**return**;

    }

    //  If queue is initially empty,set front=rear=0; start of deque

**if** (front == -1) {

front = 0;

rear = 0;

    }

**else** **if** (rear == size-1)               // rear is at last position of queue

rear = 0;

**else**                            // increment rear by 1 position

rear = rear+1;

array[rear] = key ;         // insert current element into Deque

}

// Delete element at front of Deque

**void** Deque ::deletefront()

{

**if** (isEmpty())

   {

cout &lt;&lt; "Queue Underflow!!\n" &lt;&lt; endl;

**return** ;

    }

    // Deque has only one element

**if** (front == rear)

    {

front = -1;

rear = -1;

    }

**else**

        // back to initial position

**if** (front == size -1)

front = 0;

**else** // remove current front value from Deque;increment front by 1

front = front+1;

}

// Delete element at rear end of Deque

**void** Deque::deleterear()

{

**if** (isEmpty())

    {

cout &lt;&lt; " Underflow!!\n" &lt;&lt; endl ;

**return** ;

    }

    // Deque has only one element

**if** (front == rear)

    {

front = -1;

rear = -1;

    }

**else** **if** (rear == 0)

rear = size-1;

**else**

rear = rear-1;

}

// retrieve front element of Deque

**int** Deque::getFront()

{

**if** (isEmpty())   {

cout &lt;&lt; " Underflow!!\n" &lt;&lt; endl;

**return** -1 ;

    }

**return** array[front];

}

// retrieve rear element of Deque

**int** Deque::getRear()

{

**if**(isEmpty() || rear &lt; 0)  {

cout &lt;&lt; " Underflow!!\n" &lt;&lt; endl;

**return** -1 ;

    }

**return** array[rear];

}

//main program

**int** main()

{

    Deque dq(5);

cout &lt;&lt; "Insert element 1 at rear end \n";

dq.insertrear(1);

cout &lt;&lt; "insert element 3 at rear end \n";

dq.insertrear(3);

cout &lt;&lt; "rear element of deque " &lt;&lt; " " &lt;&lt; dq.getRear() &lt;&lt; endl;

dq.deleterear();

cout &lt;&lt; "After deleterear, rear = " &lt;&lt; dq.getRear() &lt;&lt; endl;

cout &lt;&lt; "inserting element 5 at front end \n";

dq.insertfront(5);

cout &lt;&lt; "front element of deque " &lt;&lt; " "

&lt;&lt; dq.getFront() &lt;&lt; endl;

dq.deletefront();

cout &lt;&lt; "After deletefront, front = " &lt;&lt; dq.getFront() &lt;&lt; endl;

**return** 0;

}

Prac1 :

#include <iostream> using namespace std;

int main(){ int op;

cout<<"\n1. Linear Search \n2. Binary Search \n3. Fibonacci Search"<<"\n\nWhich opearation you wanna perform : ";

cin>>op;

switch(op){

case 1:

{

int size;

cout << "\nEnter the size of array : "; cin >> size;

int arr[size];

cout << "Enter the elements inside the array : "; for (int i = 0; i < size; i++)

{

cin >> arr[i];

}

int search;

cout << "Enter the number which you wanna search : "; cin >> search;

int count = 0; int temp = 0; int i=0;

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

{

if (arr[i] == search)

{

temp = 1;

cout << "\nElement is found at location " << i+1<< endl; break;

}

count++;

}

if (temp == 0)

{

cout << "\nElement not found."<<endl; cout<<"count is "<<i<<endl<<endl;

}

else{

cout << "The count is " <<i+1<<endl<<endl;

}

break;

}

case 2:

{

int n;

cout<<"\nEnter the size of array : "; cin>>n;

int arr[n];

cout<<"Enter the elements inside the array (in ascending order) : "; for(int i=0; i<n; i++){

cin>>arr[i];

}

int key;

cout<<"Enter the number which you wanna search : "; cin>>key;

int start=0, end=n; int count=0;

while(start<=end){

int mid = (start+end)/2;

count++;

if(arr[mid] == key){

cout<<"\nElement is present at position "<<mid+1<<"."<<endl; break;

}

else if(abs(arr[mid]) > abs(key)){ end = mid-1;

}

else{

start = mid+1;

}

}

if(start>end)

cout<<"Element not found."<<endl;

cout<<"Count is "<<count<<endl<<endl; break;

}

case 3:

{

int size, i;

cout << "\nEnter the total number of elements : "; cin >> size;

int arr[size];

cout << "Enter elements in array (in ascending order) : "; for (i = 0; i < size; i++)

{

cin >> arr[i];

}

int search;

cout << "Enter element to be searched : "; cin >> search;

int fbK2 = 0; int fbK1 = 1;

int fbK = fbK2 + fbK1; int offset = -1;

int flag = 0;

while (fbK < size)

{

fbK2 = fbK1; fbK1 = fbK;

fbK = fbK2 + fbK1;

}

int count = 0; while (fbK > 1)

{

count++;

int i = min(offset + fbK2, size - 1);

if (arr[i] < search)

{

fbK = fbK1; fbK1 = fbK2;

fbK2 = fbK - fbK1; offset = i;

}

else if (arr[i] > search)

{

fbK = fbK2;

fbK1 = fbK1 - fbK2; fbK2 = fbK - fbK1;

}

else

{

cout << "\nElement found at position " << i + 1 << endl; flag = 1;

break;

}

}

if (flag == 0)

{

cout <<"\nElement not found." << endl;

}

cout<<"Count is "<<count<<endl<<endl; break;

}

default:

cout<<"\nInvalid Operation. "<<endl<<endl;

}

return 0;

}

Prac 2:

#include <iostream> #include <string.h> using namespace std;

struct node

{

int empid;

char empname[20]; struct node \*next;

};

struct node \*head;

void createList(int n);

void insertNode(int, char[]); void deleteNode();

void displayData();

int main()

{

int n;

cout<<"\nHow many employee's data you wanna enter : ";

cin>>n; createList(n); displayData(); while (1)

{

int choose;

cout<<"\n\n\n\t \*\*\*\*\* Opreations Available \*\*\*\*\* \n\nChoose Operation\n1. Insert Employee Data\n2. Delete Employee Data\n3. Display Employee Data \n4. Exit\n";

cout<<"\nEnter your choice : "; cin>>choose;

switch (choose)

{

case 1:

{

int ch, empid;

char empname[20];

cout<<"\nEnter the id of employee : "; cin>>empid;

cout<<"Enter the name of employee : "; cin>>empname;

insertNode(empid, empname); break;

}

case 2:

{

deleteNode(); break;

}

case 3:

{

displayData(); break;

}

case 4:

{

exit(0);

}

default:

{

cout<<"Invalid choice ";

break;

}

}

}

}

void createList(int n)

{

struct node \*newnode, \*temp; int empid;

char empname[20];

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

{

cout<<"Unable to allocate memory"; exit(0);

}

cout<<"\nEnter the id of employee 1: "; cin>>empid;

cout<<"Enter the name of employee 1: "; cin>>empname;

head->empid = empid;

strcpy(head->empname, empname); head->next = NULL;

temp = head;

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

{

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

{

cout<<"Unable to allocate memory"; exit(0);

}

cout<<"\nEnter the id of employee "<<i<<" : "; cin>>empid;

cout<<"Enter the name of employee "<<i<<" : "; cin>>empname;

int flag = 0;

struct node \*temp1 = head; while (temp1 != NULL) {

if(temp1->empid == empid){ flag = 1;

break;

}

temp1 = temp1->next;

}

switch (flag)

case 0:

{

newnode->empid = empid; strcpy(newnode->empname, empname); newnode->next = NULL;

temp->next = newnode; temp = temp->next; break;

}

case 1:

{

cout<<"\nYou can't insert duplicate data.\n"; break;

}

default:

break;

}

}

}

void displayData()

{

struct node \*temp; if (head == NULL)

{

cout<<"EMPTY LIST";

exit(0);

}

cout<<"\nEmployee ID\tEmployee Name\n ";

temp = head;

while (temp != NULL)

cout<<endl<<temp->empid<<"\t\t"<<temp->empname; temp = temp->next;

}

}

void insertNode(int empid, char empname[])

{

struct node \*temp;

struct node \*newp=(struct node \*)malloc(sizeof(struct node)); newp->empid = empid;

strcpy(newp->empname, empname); newp->next=NULL;

int key = empid;

if(head==NULL || empid < head->empid)

{

newp->next=head; head=newp;

}

else

{

temp=head;

while(temp->next!=NULL && temp->next->empid < key)

{

temp=temp->next;

}

newp->next=temp->next; temp->next=newp;

}

displayData();

}

void deleteNode()

{

int roll;

cout<<"\nEnter the employee id of employee to be deleted : "; cin>>roll;

struct node \*prev, \*curr; while (head->empid == roll)

{

prev = head;

head = head->next; free(prev);

cout<<"Data deleted successfully\n"; displayData();

return;

}

prev = NULL; curr = head;

while (curr != NULL)

{

if (curr->empid == roll)

{

if (prev != NULL)

{

prev->next = curr->next; free(curr);

cout<<"Data deleted successfully\n"; displayData();

return;

}

}

prev = curr;

curr = curr->next;

}

cout<<"Employee id not found..\n";

}

Prac 3 :

#include<stdio.h> #include<stdlib.h>

struct node

{

float coeff; int expox; int expoy; int expoz;

struct node\* link;

};

struct node \*insert(struct node\* head,float co,int x,int y,int z)

{

struct node\* temp;

struct node\* newP=malloc(sizeof(struct node)); newP->coeff=co;

newP->expox=x; newP->expoy=y; newP->expoz=z; newP->link=NULL;

if(head == NULL || (x > head->expox && y > head->expoy && z > head->expoz))

{

newP->link = head; head=newP;

}

else

{

temp = head;

while(temp->link != NULL && (temp->link->expox >= x || temp->link->expoy

>= y || temp->link->expoz >= z))

{

temp = temp->link;

}

newP->link = temp->link; temp->link = newP;

}

return head;

}

void printRes(struct node\* head)

{

if(head == NULL)

{

printf("\nAddition not possible"); exit(0);

}

else{

struct node\* temp = head; while(temp != NULL)

{

printf("(%.1fx^%d y^%d z^%d)",temp->coeff,temp->expox,temp->expoy,temp->expoz); temp = temp->link;

if(temp!=NULL)

{

printf(" + ");

}

else{ printf("\n");

}

}

}

}

void polyadd(struct node\* head1,struct node\* head2)

{

struct node\* ptr1=head1; struct node\* ptr2=head2; struct node\* head3=NULL;

int flag;

while(ptr1 != NULL){ flag=0; ptr2=head2;

while(ptr2 != NULL){

if(ptr1->expox == ptr2->expox && ptr1->expoy == ptr2->expoy && ptr1->expoz == ptr2->expoz){

head3=insert(head3,ptr1->coeff+ptr2->coeff,ptr1->expox,ptr1-

>expoy,ptr1->expoz);

}

flag = 1;

ptr2=ptr2->link;

}

if(flag==0){

head3=insert(head3,ptr1->coeff,ptr1->expox,ptr1->expoy,ptr1->expoz);

}

ptr1=ptr1->link;

}

while(ptr2 != NULL){ flag=0; ptr1=head1;

while(ptr1 != NULL){

if(ptr1->expox == ptr2->expox && ptr1->expoy == ptr2->expoy && ptr1->expoz == ptr2->expoz){

flag = 1;

}

ptr1=ptr1->link;

}

if(flag==0){

head3=insert(head3,ptr2->coeff,ptr2->expox,ptr2->expoy,ptr2->expoz);

}

ptr2=ptr2->link;

}

printRes(head3);

}

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

{

int n,i,x,y,z; float co;

printf("Enter the number of terms : "); scanf("%d",&n);

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

{

printf("\nEnter the coefficient for the node %d : ",i+1); scanf("%f",&co);

printf("Enter the exponent of x for the node %d : ",i+1); scanf("%d",&x);

printf("Enter the exponent of y for the node %d : ",i+1); scanf("%d",&y);

printf("Enter the exponent of z for the node %d : ",i+1); scanf("%d",&z);

head=insert(head,co,x,y,z);

}

return head;

}

int main()

{

struct node\* head1 = NULL; struct node\* head2 = NULL;

printf("\nEnter the values of Polynomial 1 \n\n"); head1=create(head1);

printf("\nPolynomial 1 is \n"); printRes(head1);

printf("\n\nEnter the values of Polynomial 2\n\n"); head2=create(head2);

printf("\nPolynomial 2 is \n"); printRes(head2);

printf("\n\nAddition of polynomial is \n"); polyadd(head1,head2);

return 0;

}

Prac 4 :

#include <stdio.h> #include <stdlib.h> #include <string.h>

struct node{

struct node\* prev;

int vquantity,fquantity;

char vegname[30], frname[30]; struct node\* next;

};

struct node\* firstNode(struct node\* head, char name[20], int quan, int ch){ struct node\* temp = malloc(sizeof(struct node));

if(ch==1){

temp->prev = NULL;

strcpy(temp->vegname, name); temp->vquantity = quan;

temp->next = NULL;

}

else{

temp->prev = NULL; strcpy(temp->frname, name); temp->fquantity = quan; temp->next = NULL;

}

head = temp; return head;

}

struct node\* exceptFirstNode(struct node\* head, char name[20], int quan, int ch){ struct node\* temp, \*tp;

if(ch==1){

temp = malloc(sizeof(struct node)); temp->prev = NULL;

strcpy(temp->vegname, name); temp->vquantity = quan;

temp->next = NULL; tp = head;

while(tp->next != NULL)

tp = tp->next; tp->next = temp; temp->prev = tp;

}

else{

temp = malloc(sizeof(struct node)); temp->prev = NULL;

strcpy(temp->frname, name); temp->fquantity = quan; temp->next = NULL;

tp = head;

while(tp->next != NULL) tp = tp->next;

tp->next = temp; temp->prev = tp;

}

return head;

}

struct node\* createList(struct node\* head,int ch){ int n, quan, i;

char name[20]; if(ch==1){

printf("\nEnter the number of nodes for vegetable list : ");

scanf("%d", &n);

if(n==0) return head;

printf("\nEnter the vegetable name for the node 1 : "); scanf("%s",name);

printf("Enter the vegetable quantity for the node 1 : "); scanf("%d",&quan);

head = firstNode(head, name,quan,1);

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

printf("\nEnter the vegetable name for the node %d : ", i+1); scanf("%s", name);

printf("Enter the vegetable quantity for the node %d : ", i+1); scanf("%d",&quan);

head = exceptFirstNode(head, name, quan, 1);

}

}

else{

printf("\nEnter the number of nodes for fruit list : "); scanf("%d", &n);

if(n==0)

return head;

printf("\nEnter the fruit name for the node 1 : "); scanf("%s",name);

printf("Enter the fruit quantity for the node 1 : "); scanf("%d",&quan);

head = firstNode(head, name,quan,2);

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

printf("\nEnter the fruit name for the node %d : ", i+1); scanf("%s", name);

printf("Enter the fruit quantity for the node %d : ", i+1); scanf("%d",&quan);

head = exceptFirstNode(head, name, quan, 2);

}

}

return head;

}

struct node\* sortVegList(struct node\* head, int ch){

struct node \*current = NULL, \*index = NULL, \*ptr;

if(ch==1){

char temp1[30]; int temp2;

if(head == NULL) { return 0;

}

else {

for(current = head; current->next != NULL; current = current->next) {

for(index = current->next; index != NULL; index = index->next) {

if(current->vquantity > index->vquantity) { temp2 = current->vquantity; strcpy(temp1, current->vegname); current->vquantity = index->vquantity;

strcpy(current->vegname, index->vegname); index->vquantity = temp2;

strcpy(index->vegname, temp1);

}

}

}

}

}

else if(ch==2){ char temp1[30]; int temp2;

if(head == NULL) {

return 0;

}

else {

for(current = head; current->next != NULL; current = current->next) {

for(index = current->next; index != NULL; index = index->next) {

if(current->vquantity < index->vquantity) { temp2 = current->vquantity; strcpy(temp1, current->vegname); current->vquantity = index->vquantity;

strcpy(current->vegname, index->vegname); index->vquantity = temp2;

strcpy(index->vegname, temp1);

}

}

}

}

}

ptr = head;

printf("\n Vegetable Name\t\t\tVegetable Quantity"); printf("\n "); while(ptr != NULL){

printf("\n%s\t\t\t\t\t%d",ptr->vegname,ptr->vquantity);

ptr = ptr->next;

}

printf("\n\n");

return head;

}

struct node\* sortFrList(struct node\* head, int ch){

struct node \*current = NULL, \*index = NULL, \*ptr;

if(ch==1){

char temp1[30]; int temp2;

if(head == NULL) { return 0;

}

else {

for(current = head; current->next != NULL; current = current->next) {

for(index = current->next; index != NULL; index = index->next) {

if(current->fquantity > index->fquantity) { temp2 = current->fquantity;

strcpy(temp1, current->frname); current->fquantity = index->fquantity;

strcpy(current->frname, index->frname); index->fquantity = temp2;

strcpy(index->frname, temp1);

}

}

}

}

}

else if(ch==2){ char temp1[30]; int temp2;

if(head == NULL) { return 0;

}

else {

for(current = head; current->next != NULL; current = current->next) {

for(index = current->next; index != NULL; index = index->next) {

if(current->fquantity < index->fquantity) { temp2 = current->fquantity; strcpy(temp1, current->frname);

current->fquantity = index->fquantity; strcpy(current->frname, index->frname); index->fquantity = temp2;

strcpy(index->frname, temp1);

}

}

}

}

}

ptr = head;

printf("\n Fruit Name\t\t\tFruit Quantity");

printf("\n "); while(ptr != NULL){

printf("\n%s\t\t\t\t\t%d",ptr->frname,ptr->fquantity); ptr = ptr->next;

}

printf("\n\n");

return head;

}

struct node\* optionsOfList(struct node\* head){ while(1){

int ch,cch,ich,vquan,fquan,dvch,dfch; char vname[30], fname[30];

printf("\n\t\t---Welcome to JD Mall ---\n\n");

printf("1. Create list. \n2. Insertion in the list. \n3. Display vegetables list. \n4. Display Fruit list. \n5. Exit.\n");

printf("Enter your choice : "); scanf("%d",&ch);

switch (ch)

{

case 1:

{

printf("\nFor which item you wanna create list (1. Vegetables 2. Fruits) :"); scanf("%d",&cch);

if(cch==1){

head = createList(head,1);

}

else if(cch==2){

head = createList(head,2);

}

else{

printf("Invalid Choice... ");

}

break;

}

case 2:

{

printf("\nIn which list you wanna enter (1. Vegetables 2. Fruits) : "); scanf("%d",&ich);

if(ich == 1){

printf("Enter the name of vegetable : "); scanf("%s",vname);

printf("Enter the quantity of vegetable : "); scanf("%d",&vquan);

head = exceptFirstNode(head, vname, vquan, ich);

}

else if(ich == 2){

printf("Enter the name of fruit : "); scanf("%s",fname);

printf("Enter the quantity of fruit : "); scanf("%d",&fquan);

head = exceptFirstNode(head, fname, fquan, ich);

}

else{

printf("Invalid Choice... ");

}

break;

}

case 3:

{

");

printf("\nIn which order you wanna display vegetables (1. Ascending 2. Descending) :

scanf("%d",&dvch); if(dvch == 1){

sortVegList(head,1);

}

else if(dvch == 2){ sortVegList(head,2);

}

else{

printf("Invalid Choice... ");

}

break;

}

case 4:

{

printf("In which order you wanna display fruits (1. Ascending 2. Descending) : "); scanf("%d",&dfch);

if(dfch == 1){ sortFrList(head,1);

}

else if(dfch == 2){

sortFrList(head,2);

}

else{

printf("Invalid Choice... ");

}

break;

}

case 5:

{

exit(0);

}

default:

{

printf("Invalid Choice..."); break;

}

}

}

}

int main(){

struct node\* head = NULL; head = optionsOfList(head);

return 0;

}

Prac 5 :

#include <iostream> #include <string> #include <cstring> #include<math.h> using namespace std;

int asize = 100; string stack[100]; int top = -1;

bool flag = 0;

void push(string item)

{

if(top >= asize-1)

{

}

else

{

}

}

cout<<"\nStack Overflow.";

top = top+1; stack[top] = item;

bool check\_number(string item) { for (int i = 0; i < item.length(); i++)

if (isdigit(item[i]) == false) return false;

return true;

}

bool check\_alpha(string item) {

for (int i = 0; i < item.length(); i++) if (isalpha(item[i]) == false)

return false; return true;

}

bool is\_operator(string symbol)

{

if(symbol == "^" || symbol == "\*" || symbol == "/" || symbol == "+" || symbol =="-") return true;

else

return false;

}

string pop()

{

string item;

if(top <0)

{

}

else{

}

printf("stack under flow"); exit(1);

item = stack[top]; top = top-1; return(item);

int precedence(string symbol)

{

if(symbol == "^")

return 3;

else if(symbol == "\*" || symbol == "/") return 2;

else if(symbol == "+" || symbol == "-") return 1;

else

return 0;

}

int evaluate(string postf[], int size)

{

int i=0,op1,op2; string item=postf[i]; while(i < size)

{

else

if(check\_number(item) || check\_alpha(item) )

{

push(item);

}

{

op2=stoi(pop());

op1=stoi(pop());

int resadd = op1+op2;

int ressub = op1-op2; int resmul = op1\*op2; int resdiv = op1/op2;

int respow = pow(op1,op2);

char char\_array[2]; strcpy(char\_array, item.c\_str());

switch(char\_array[0])

{

case '+': push(to\_string(resadd)); break;

case '-': push(to\_string(ressub)); break;

case '\*': push(to\_string(resmul)); break;

case '/': push(to\_string(resdiv)); break;

case '^': push(to\_string(respow)); break;

}

} i++;

}

item = postf[i];

return stoi(stack[top]);

}

int main(){

string infix\_exp[asize], postfix\_exp[asize], item;

cout<<"\nEnter Infix expression : "; int i=0;

while(1){ cin>>infix\_exp[i]; i++;

if (cin.get() == '\n') { break;

}

}

int size = i;

cout<<"\nInfix expression : "; for (int i = 0; i < size; i++)

cout<<infix\_exp[i]<<" ";

int j; string x;

push("("); infix\_exp[size] = ")";

i=0; j=0;

item=infix\_exp[i];

while(i <= size)

{

if(item == "(")

{

push(item);

}

else if(check\_number(item) || check\_alpha(item))

{

if(check\_number(item)) flag = 1;

postfix\_exp[j] = item; j++;

}

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

{

x=pop();

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

{

postfix\_exp[j] = x; j++;

x=pop();

}

push(x);

push(item);

}

else if(item == ")")

{

x = pop(); while(x != "(")

{

postfix\_exp[j] = x;

j++;

x = pop();

}

}

else

{

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

exit(1);

} i++;

item = infix\_exp[i];

}

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

cout<<"\nPostfix expression : "; for (int i = 0; i < size; i++)

cout<<postfix\_exp[i]<<" ";

if(flag){

cout<<"\nEvaluated expression is: "<<evaluate(postfix\_exp,size);

}

return 0;

}

Prac 6 :

#include <iostream> using namespace std;

int front = -1; int rear = -1;

int btQueue[100]; int wt[100];

int tat[100];

void enQueue(int samplebt)

{

if (rear > 100)

{

cout << "\nQueue Overflow...";

}

else if (rear == -1)

{

front++; rear++;

btQueue[rear] = samplebt;

}

else

{

rear++;

btQueue[rear] = samplebt;

}

}

void deQueue()

{

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

{

cout << "\nQueue is empty...";

}

else

{

front++;

cout << "\nElement deleted sucessfully...\n";

}

}

void waitingTime()

{

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

{

wt[i] = 0;

}

wt[0] = 0;

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

{

wt[i + 1] = wt[i] + btQueue[i];

}

}

void turaroundTime()

{

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

{

tat[i] = 0;

}

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

{

tat[i] = wt[i] + btQueue[i];

}

}

void display()

{

cout << "\n\tSr.no. \tBurst time \tWaiting time Turnaround time \n"; cout << " \n"; int srno = 1;

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

{

cout << "\t" << srno << "\t\t " << btQueue[i] << "\t\t " << wt[i] << " " << tat[i]

<< "\n";

srno++;

}

int totalwt = 0, totaltat = 0, size = 0; for (int i = front; i <= rear; i++)

{

totalwt += wt[i]; totaltat += tat[i]; size++;

}

int avgwt = totalwt / size; int avgtat = totaltat / size;

cout << " \n"; cout << "\t"

<< "Average"

<< "\t\t "

<< "\t\t " << avgwt << " " << avgtat << "\n";

}

void withoutPriority()

{

int samplebt,n;

while (1)

{

int ch;

cout << "\nWhich operation you wanna perform \n1. enQueue \n2. deQueue \n3. Display

\n4. Exit \nEnter your choice : "; cin >> ch;

switch (ch)

{

case 1:

{

cout<<"\nHow many burst time you wanna enter : "; cin>>n;

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

{

cout << "\nEnter burst time of process "<<i+1<<" : "; cin >> samplebt;

enQueue(samplebt);

}

break;

}

case 2:

{

deQueue(); break;

}

case 3 :

{

waitingTime();

turaroundTime();

display();

break;

}

case 4:

{

exit(0);

}

default:

cout << "Invalid choice ";

break;

}

}

}

void withPriority()

{

int r; while (1)

{

int ch;

cout << "\nWhich operation you wanna perform \n1. enQueue \n2. deQueue \n3. Display

\n4. Exit \nEnter your choice : "; cin >> ch;

switch (ch)

{

case 1:

cout << "Enter the number of rows : "; cin >> r;

int temprow = r; int tempcol = r + 1;

int arr[temprow][tempcol];

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

{

for (int j = 0; j < tempcol; j++)

{

arr[i][j] = 0;

}

}

int position = tempcol - 1; int check = temprow - 1;

int tempbt = 0, pr = 0, my\_index = 0, flag = 0; int ch;

while (r > my\_index)

{

if (flag == 1)

{

cout << "\nDo you wanna continue (1. Yes 2. No) : "; cin >> ch;

}

if (flag == 0)

{

ch = 1;

}

if (ch == 1)

{

here:

cout << "\nEnter the burst time : "; cin >> tempbt;

cout << "Enter the priority : "; cin >> pr;

my\_index = arr[pr][position];

int cont;

if (my\_index == r)

{

cout << "You reached the limit in this priority...";

cout << "Do you wanna continue in other priority (1.Yes 2.No) : "; cin >> cont;

if (cont == 1)

{

goto here;

}

}

arr[pr][my\_index] = tempbt; arr[pr][position]++;

flag = 1;

}

else

{

break;

}

}

int size = r \* r;

int tbt[size], kvalue = 0; int counter = 0;

int btsize = 0;

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

{

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

{

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

{

}

else

{

btsize++;

}

tbt[counter] = arr[i][j]; counter++;

}

}

int bt[btsize - 1]; int ct = 0;

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

{

if (tbt[i] == 0)

{

continue;

}

enQueue(tbt[i]); if (ct < btsize)

{

ct++;

}

}

break;

}

case 2:

{

deQueue(); break;

}

case 3:

{

waitingTime(); turaroundTime(); display();

break;

}

case 4:

{

exit(0);

}

default:

cout << "Invalid choice ";

break;

}

}

}

int main()

{

while (1)

{

int choice;

cout << "\nChoose from the following : \n1.Queue without priority. \n2.Queue with priority.

\n3.Exit \nEnter your choice : "; cin >> choice;

switch (choice)

{

case 1:

withoutPriority(); break;

case 2:

withPriority(); break;

case 3:

exit(0);

default:

cout << "Invalid Choice ";

}

}

return 0;

}

Prac 7 :

// Redo Undo Functionality Using Doubly Linked List #include<bits/stdc++.h>

using namespace std;

class Node{ public:

string action;

Node\* next;

Node\* prev;

Node(string action){

this->action = action; this->next = NULL; this->prev = NULL;

}

};

/\* Function to insert action into UndoList \*/ Node\* Write(string action,Node\* head){

Node\* newNode = new Node(action); if (head == NULL)

{

head = newNode; return head;

}else if (head->next == NULL)

{

head->next = newNode;

}else{

newNode->prev = head; return head;

Node\* temp = head; while(temp->next!=NULL){

temp = temp->next;

}

temp->next = newNode; newNode->prev = temp; return head;

}

return head;

}

void Read(Node\* head){ Node\* temp = head;

cout << "Text Editor String : "; while(temp!= NULL){

cout << temp->action[6]; temp = temp->next;

}

cout << endl;

}

Node\* Undo(Node\* Redo,Node\* Undo){

/\* Traversing through the Undo List \*/ Node\* temp = Undo;

while(temp->next != NULL)

temp = temp->next;

/\* Clear previous node next field \*/ if (temp->prev != NULL)

{

temp->prev->next = NULL; // Important Part of Program

}

if (Redo == NULL)

{

Redo = temp;

temp->next = NULL; temp->prev = NULL; return Redo;

}

else if (Redo->next == NULL)

{

Redo->next = temp; temp->prev = Redo; Redo = Redo->next; return Redo;

}

return Redo;

}

Node\* Redo(Node\* Redo,Node\* Undo){

Node\* Undotemp = Undo; Node\* Redotemp = Redo; if (Redo->prev !=NULL)

{

Redo = Redo->prev;

}

else{

}

Redo->next = NULL;

Redo = NULL;

if (Undotemp == NULL)

{

Undotemp = Redotemp;

}

else if (Undotemp->next == NULL)

{

}

else

{

Undotemp->next = Redotemp; Redotemp->prev = Undotemp;

while(Undotemp->next!=NULL){ Undotemp = Undotemp->next;

}

Undotemp->next = Redotemp; Redotemp->prev = Undotemp;

}

return Redo;

}

int main()

{

Node\* UndoHead = NULL; Node\* RedoHead = NULL; while(true)

{

int choice;

cout << "Choose From the following : " << endl; cout << "1.Write" << endl;

cout << "2.Read" << endl; cout << "3.Undo" << endl; cout << "4.Redo" << endl; cout << "5.Exit" << endl;

cout << "Enter Your choice : "; cin >> choice;

switch(choice){

case 1:{

char c;

cout << "Enter Your character : "; cin >> c;

string b = "Write "; string act = b+c; cout << act << endl;

UndoHead = Write(act,UndoHead);

}

break; case 2:{

Read(UndoHead);

}

break; case 3:{

RedoHead = Undo(RedoHead,UndoHead);

}

break;

case 4:{

RedoHead = Redo(RedoHead,UndoHead);

}

break; case 5:{

exit(0);

}

default:{

cout << "Invalid Choice" << endl;

}

}

}

return 0;

}

Prac 8 :

from drawtree import draw\_bst class node:

def init (self,data): self.data=data self.left = None self.right = None

def draw(l):

draw\_bst(l)

def display(root) :

if (root != None) : l.append(root.data) display(root.left) display(root.right)

def postorder(root):

if(root!=None):

postorder(root.left) postorder(root.right) print(root.data,end=" ")

def construct(start, end): list = []

if (start > end) : list.append(None) return list

for i in range(start, end + 1): lefttree = construct(start, i - 1) righttree = construct(i + 1, end) for j in range(len(lefttree)) :

left = lefttree[j]

for k in range(len(righttree)): right = righttree[k] newnode = node(i) newnode.left = left newnode.right = right list.append(newnode)

return list

l = []

print("Enter limits :") n1 = int(input())

n2 = int(input())

alltrees = construct(n1,n2) print("all constructed BSTs are")

for i in range(len(alltrees)): l = []

display(alltrees[i]) draw(l)

print() print("-"\*10) print()

print("postorder of all BST's are :") for i in range(len(alltrees)):

postorder(alltrees[i]) print()

Prac 9 :

#include <iostream>

using namespace std;

void heapify(int arr[], int n, int root)

{

int largest = root;

int l = 2 \* root + 1;

int r = 2 \* root + 2;

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

largest = l;

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

largest = r;

if (largest != root)

{

swap(arr[root], arr[largest]);

heapify(arr, n, largest);

}

}

void displayArray(int arr[], int n)

{

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

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

}

void heapSort(int arr[], int n)

{

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

heapify(arr, n, i);

int count = 1;

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

{

displayArray(arr, n);

cout << " now sort " << count++ << "th element \n\n";

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

heapify(arr, i, 0);

}

}

int main()

{

int n;

cout << "\nEnter the size of array : ";

cin >> n;

int heap\_arr[n];

cout << "\nEnter the elements of array : ";

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

{

cin >> heap\_arr[i];

}

cout << "Input array : ";

displayArray(heap\_arr, n);

cout << "\n\n"

<< "Convert array into MAX HEAP :\n\n";

heapSort(heap\_arr, n);

displayArray(heap\_arr, n);

cout << " heap contains only 1 element so it is already sorted ";

cout << "\n\n\n\nSorted array is : ";

displayArray(heap\_arr, n);

}

Prac 10 :

#include <stdio.h>

#include <conio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX], int n, int startnode);

int main()

{

int G[MAX][MAX], i, j, n, u;

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

scanf("%d", &n);

printf("\nEnter the adjacency matrix:\n");

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

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

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

printf("\nEnter the starting node:");

scanf("%d", &u);

dijkstra(G, n, u);

return 0;

}

void dijkstra(int G[MAX][MAX], int n, int startnode)

{

int cost[MAX][MAX], distance[MAX], pred[MAX];

int visited[MAX], count, mindistance, nextnode, i, j;

// pred[] stores the predecessor of each node

// count gives the number of nodes seen so far

// create the cost matrix

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

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

if (G[i][j] == 0)

cost[i][j] = INFINITY;

else

cost[i][j] = G[i][j];

// initialize pred[],distance[] and visited[]

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

{

distance[i] = cost[startnode][i];

pred[i] = startnode;

visited[i] = 0;

}

distance[startnode] = 0;

visited[startnode] = 1;

count = 1;

while (count < n - 1)

{

mindistance = INFINITY;

// nextnode gives the node at minimum distance

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

if (distance[i] < mindistance && !visited[i])

{

mindistance = distance[i];

nextnode = i;

}

// check if a better path exists through nextnode

visited[nextnode] = 1;

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

if (!visited[i])

if (mindistance + cost[nextnode][i] < distance[i])

{

distance[i] = mindistance + cost[nextnode][i];

pred[i] = nextnode;

}

count++;

}

// print the path and distance of each node

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

if (i != startnode)

{

printf("\nDistance of node%d=%d", i, distance[i]);

printf("\nPath=%d", i);

j = i;

do

{

j = pred[j];

printf("<-%d", j);

} while (j != startnode);

}

}

Prac 11 :

from tkinter import \*

from tkinter import messagebox

import random as r

def button(frame):

b = Button(frame, padx=1, bg="black", width=3, text=" ",

font=('arial', 60, 'bold'), relief="sunken", bd=10)

return b

def change\_a():

global a

for i in ['O', 'X']:

if not(i == a):

a = i

break

def reset():

global a

for i in range(3):

for j in range(3):

b[i][j]["text"] = " "

b[i][j]["state"] = NORMAL

a = r.choice(['O', 'X'])

def check():

for i in range(3):

if(b[i][0]["text"] == b[i][1]["text"] == b[i][2]["text"] == a or b[0][i]["text"] == b[1][i]["text"] == b[2][i]["text"] == a):

messagebox.showinfo("Congrats!!", "'"+a+"' Has Won")

reset()

if(b[0][0]["text"] == b[1][1]["text"] == b[2][2]["text"] == a or b[0][2]["text"] == b[1][1]["text"] == b[2][0]["text"] == a):

messagebox.showinfo("Congrats!!", "'"+a+"' Has Won")

reset()

elif(b[0][0]["state"] == b[0][1]["state"] == b[0][2]["state"] == b[1][0]["state"] == b[1][1]["state"] == b[1][2]["state"] == b[2][0]["state"] == b[2][1]["state"] == b[2][2]["state"] == DISABLED):

messagebox.showinfo("Tied!!", "The match ended in a draw")

reset()

def click(row, col):

b[row][col].config(text=a, state=DISABLED, disabledforeground=colour[a])

check()

change\_a()

label.config(text=a+"'s Chance")

root = Tk()

root.title("Tic-Tac-Toe")

a = r.choice(['O', 'X'])

colour = {'O': "white", 'X': "red"}

b = [[], [], []]

for i in range(3):

for j in range(3):

b[i].append(button(root))

b[i][j].config(command=lambda row=i, col=j: click(row, col))

b[i][j].grid(row=i, column=j)

label = Label(text=a+"'s Chance", font=('arial', 20, 'bold'))

label.grid(row=3, column=0, columnspan=3)

root.mainloop()

Tower of honoi :

#include <stdio.h>

// C recursive function to solve tower of hanoi puzzle

void towerOfHanoi(int n, char from\_rod, char to\_rod, char aux\_rod)

{

if (n == 1)

{

printf("\n Move disk 1 from rod %c to rod %c", from\_rod, to\_rod);

return;

}

towerOfHanoi(n-1, from\_rod, aux\_rod, to\_rod);

printf("\n Move disk %d from rod %c to rod %c", n, from\_rod, to\_rod);

towerOfHanoi(n-1, aux\_rod, to\_rod, from\_rod);

}

int main()

{

int n = 4; // Number of disks

towerOfHanoi(n, \'A\', \'C\', \'B\'); // A, B and C are names of rods

return 0;

}

// C++ recursive function to

// solve tower of hanoi puzzle

#include <bits/stdc++.h>

**using** **namespace** std;

**void** towerOfHanoi(**int** n, **char** from\_rod,

**char** to\_rod, **char** aux\_rod)

{

**if** (n == 0)

    {

**return**;

    }

    towerOfHanoi(n - 1, from\_rod, aux\_rod, to\_rod);

    cout << "Move disk " << n << " from rod " << from\_rod <<

                                " to rod " << to\_rod << endl;

    towerOfHanoi(n - 1, aux\_rod, to\_rod, from\_rod);

}

// Driver code

**int** main()

{

**int** n = 4; // Number of disks

    towerOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods

**return** 0;

}