**SSN COLLEGE OF ENGINEERING**

**DEPARTMENT OF CSE**

**UCS 1312 DATA STRUCTURES LABORATORY**

**MINI PROJECT**

**BUSINESS WHALE-BUSINESS MANAGEMENT SYSTEM**

**S.MADHUMITHA-185001086**

**R.PRANEETHA-18500109**

**S.RAJASHREE-18500122**

**ABSTRACT:**

The ever-expanding network of business companies and start-up units demands lot of labour for managing the business transactions like taking note of orders, rerouting the orders to the nearest branch, dispatching and tracking the orders from the client side. Also jotting down the important events and business meetings is important for an employee in order to lessen the burden on his brain and to overcome the setbacks that occur as a result of absent-mindedness. Business-Whale works to overcome these awkward situations and reduces a lot of paperwork.

**METHODOLOGY:**

Here we maintain three main domains

1.Employee

To list out the important proceedings in the company the events for a day in the company are maintained in a queue data structure that is accessible by all employees and can be viewed by them in Time sorted order. Once any meeting is over then it is dequeued out.

2.Client

To get an order form the client: the order for the product form the client is stored in a priority queue and the highest amounting order is dispatched first from the priority queue.

The client also gets to check if his order for the product has been dispatched from the respective branch or not by giving his order ID.

More over a client is also given the freedom to customize the location where his/her order will be placed. The client is asked to enter a location and depending on that the nearest branch of the company is tracked down using the shortest path algorithms in graphs data structure. That location is set and the client’s order is assigned to that branch.

The client also can view the contact information of the employees in the company so that he can reach out to them in case of any discrepancy in placing an order or tracking down his/her order. This is printed using the Binary search tree that stored the employee information in hierarchical order based on their Rank.

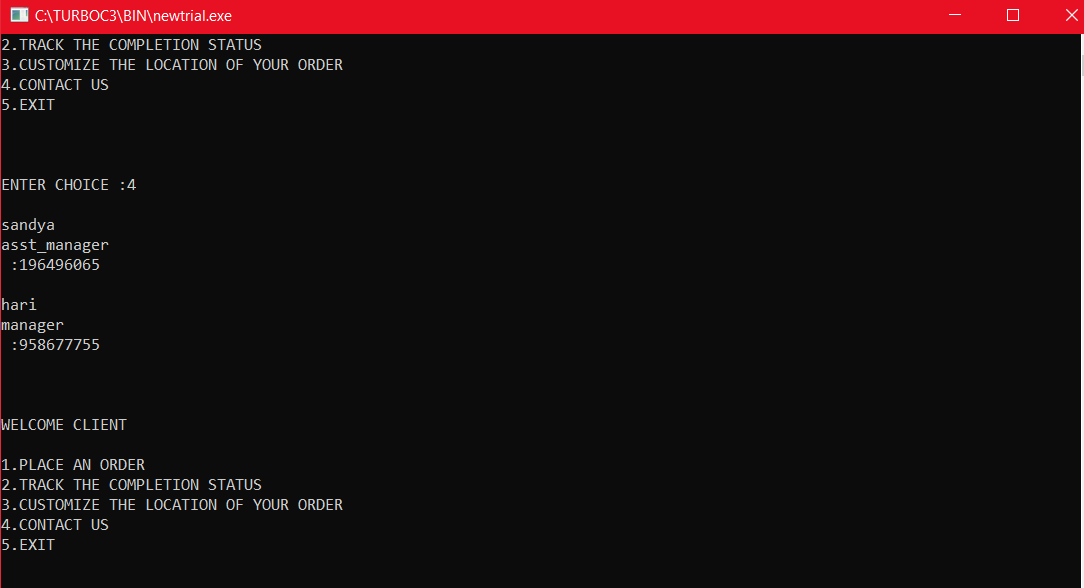
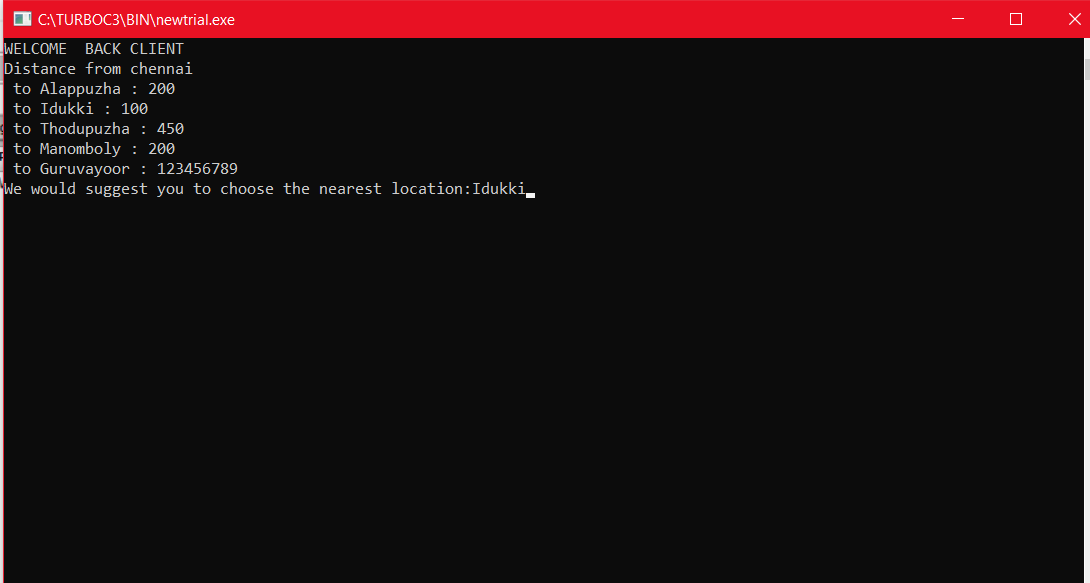
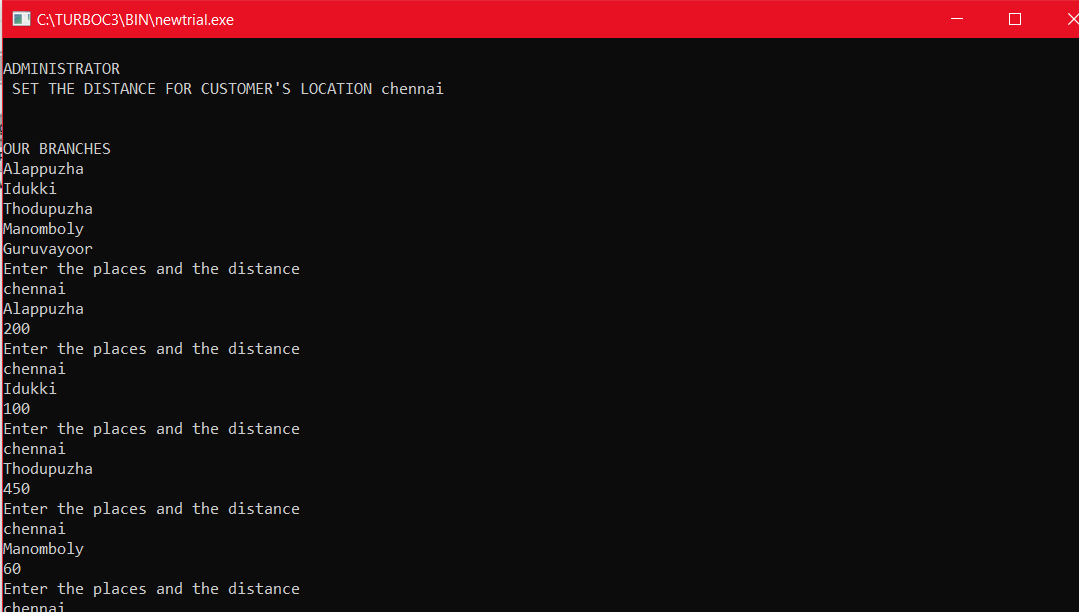
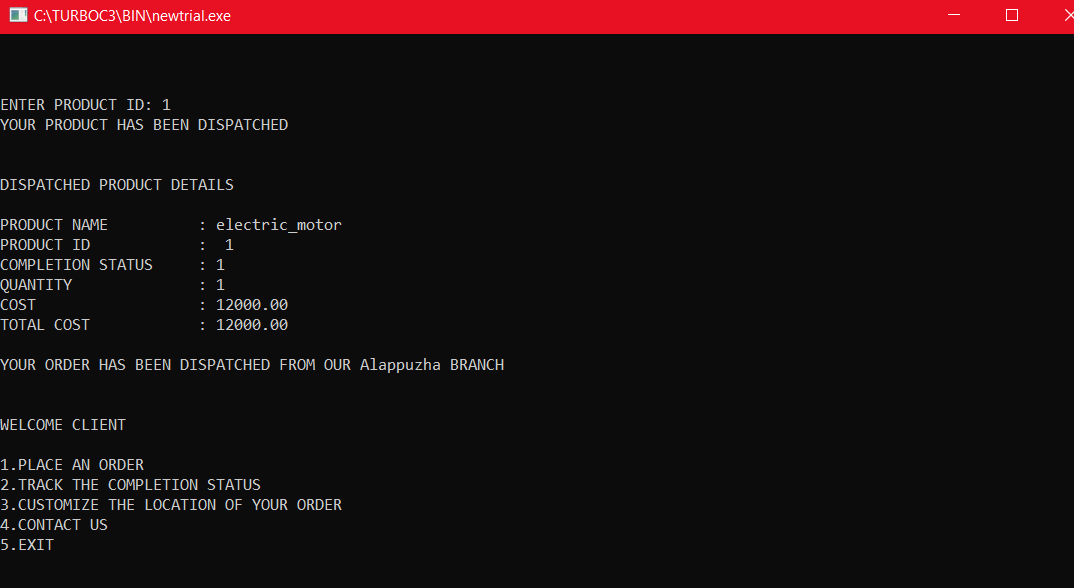
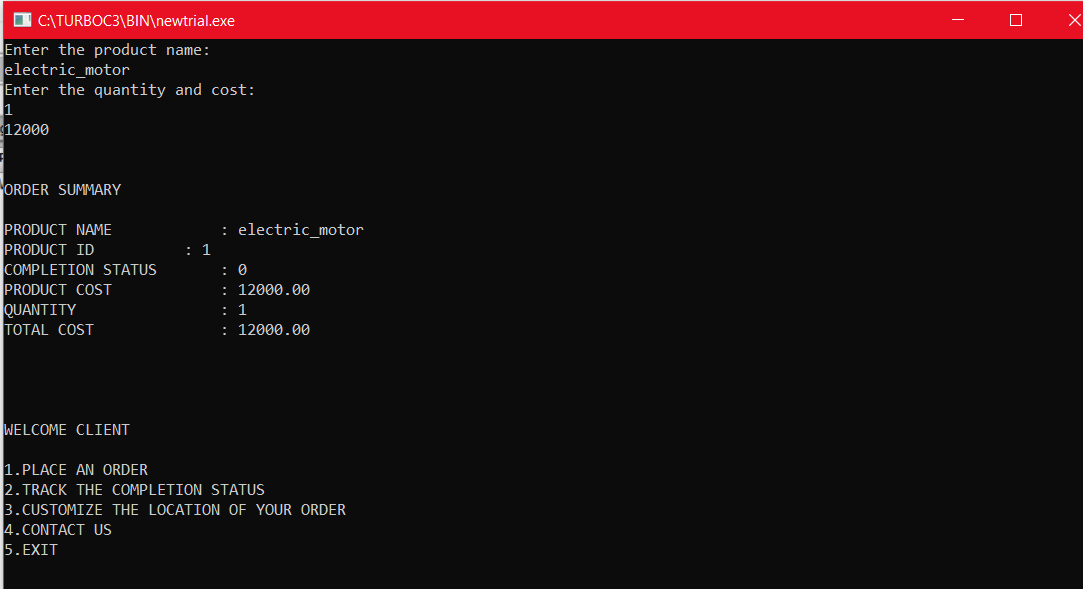
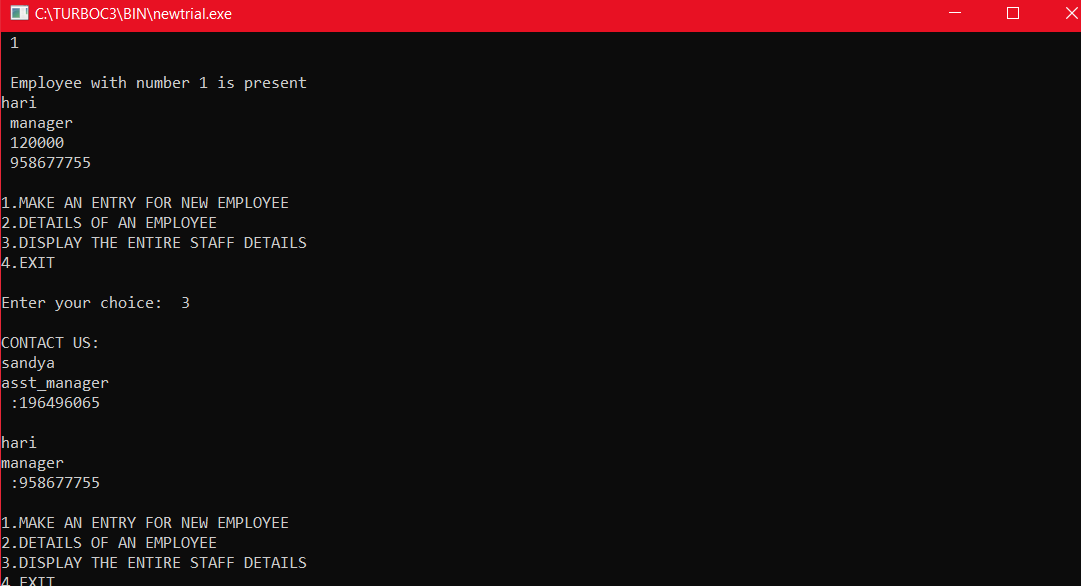
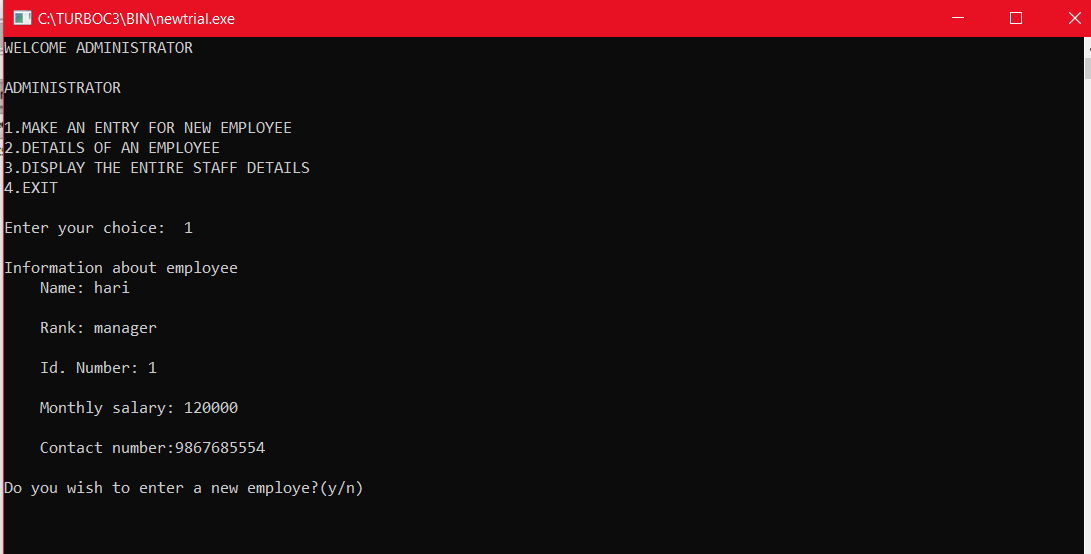
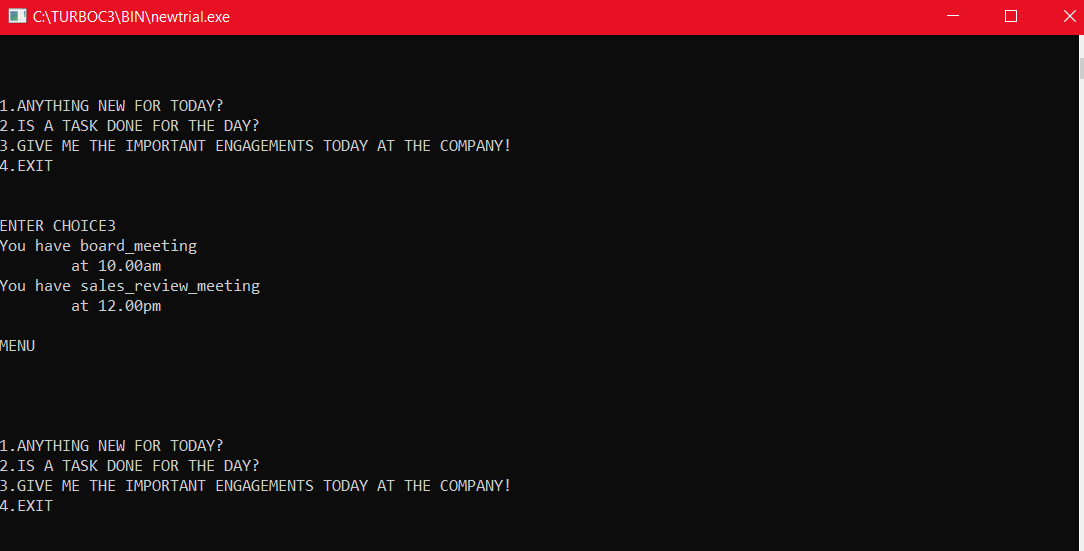
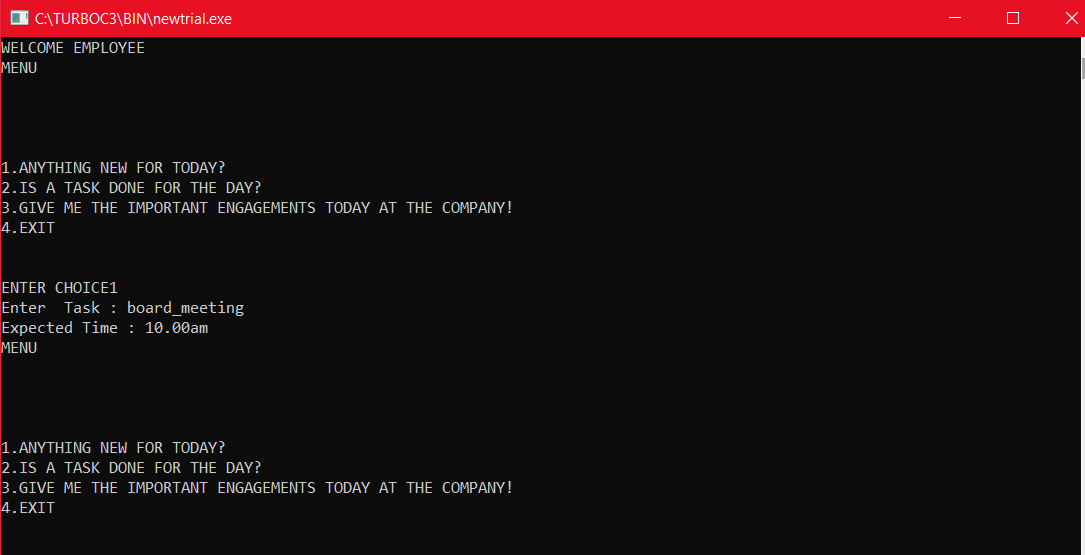
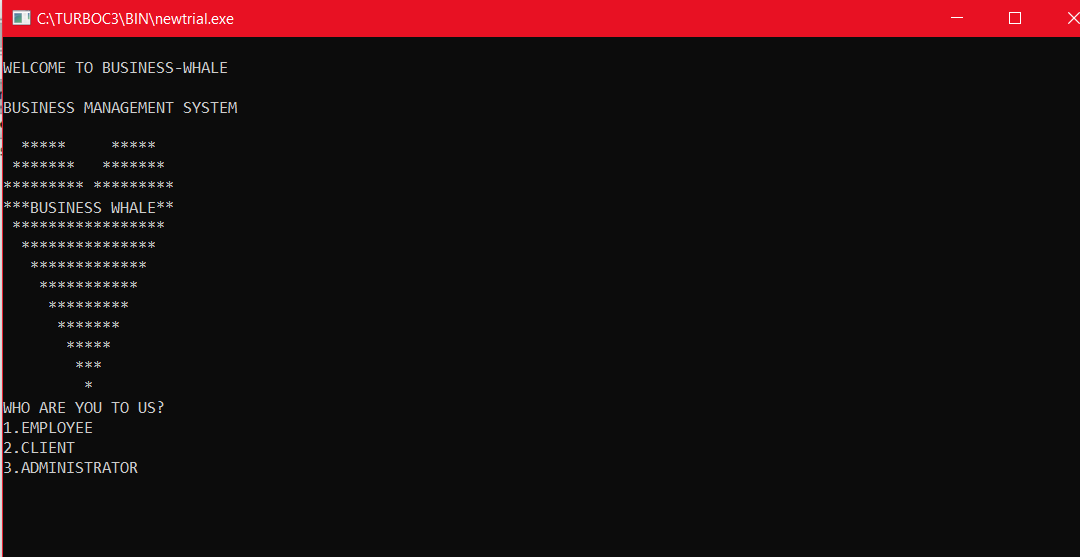
3.Administrator

The administrator holds a key position in this system. He is empowered to add employees to the company and their contact details, rank, employee id etc in a Binary search tree data structure.

The administrator also sets the distance from the client’s location to the location of the various branches of the company so that the client can get an idea of the nearest branch to place his order using graph structure.

The administrator can also search for certain employees in the company given the employee number so that it can be used for administrative purposes.

**OUTPUT SCREEN SHOTS:**



**CODE:**

**MAIN FILE:**

#include<stdio.h>

#include<string.h>

#define maxVertices 100

#define INF 123456789

#include "priorqueue.h"

#include "queues.h"

#include "graphs.h"

#include "binarytree.h"

#include "graphics.h"

main(){

int i,ch,c;

int proid=0;

char choice='y';

node \*root;

char branch[30]="Alappuzha";

printf("\n");

printf("WELCOME TO BUSINESS-WHALE \n\nBUSINESS MANAGEMENT SYSTEM\n");

printf("\n");

//system("cls");

star();

while(choice=='y'||choice=='Y'){

printf("WHO ARE YOU TO US?\n1.EMPLOYEE\n2.CLIENT\n3.ADMINISTRATOR\n");

scanf("%d",&ch);

system("cls");

switch(ch){

case 1:printf("WELCOME EMPLOYEE\n");

struct node\* p;

int ch;

do

{

char x[100],y[100];

printf("MENU\n");

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

printf("\n1.ANYTHING NEW FOR TODAY? \n2.IS A TASK DONE FOR THE DAY? \n3.GIVE ME THE IMPORTANT ENGAGEMENTS TODAY AT THE COMPANY!\n4.EXIT\n\n\nENTER CHOICE");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Enter Task : ");

scanf("%s",&x);

printf("Expected Time : ");

scanf("%s",&y);

Enqueue(x,y);

break;

case 2:

Dequeue();

break;

case 3:

Print();

break;

case 4:

break;

}

}while(ch!=4);

break;

case 2:

do{

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

printf("WELCOME CLIENT\n\n1.PLACE AN ORDER\n2.TRACK THE COMPLETION STATUS\n3.CUSTOMIZE THE LOCATION OF YOUR ORDER\n4.CONTACT US\n5.EXIT\n");

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

printf("ENTER CHOICE :");

scanf("%d",&c);

pq temp;

switch(c){

case 1:printf("\NENTER THE PRODUCT DETAILS \n");

PQ \*Q;

Q=createPQ(100);

int n;

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

scanf("%d",&n);

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

{

insert(Q);

}

//printf("\nThe Queue after insertion : \n\n");

for(int j=1;j<Q->size;j++)

{

printf("\n\nORDER SUMMARY\n\n");//I am printing the position in the array only for reference

printf("PRODUCT NAME : %s\n" ,Q->arr[j].pname);

printf("PRODUCT ID : %d\n",Q->arr[j].id);

printf("COMPLETION STATUS : %d\n" ,Q->arr[j].ct);

printf("PRODUCT COST : %.2f\n",Q->arr[j].pcost);

printf("QUANTITY : %d\n",Q->arr[j].quantity);

printf("TOTAL COST : %.2f\n\n",Q->arr[j].tcost);

}

temp=deletemax(Q);

temp.ct=1;

break;

case 2:

printf("\nCOMPLETION STATUS OF YOUR ORDER\n\n");

printf("\n\nENTER PRODUCT ID: ");scanf("%d",&proid);

if(temp.id==proid){printf("YOUR PRODUCT HAS BEEN DISPATCHED\n");

printf("\n\nDISPATCHED PRODUCT DETAILS \n\n");

printf("PRODUCT NAME : %s\n" ,temp.pname);

printf("PRODUCT ID : %d\n",temp.id);

printf("COMPLETION STATUS : %d\n" ,temp.ct);

printf("QUANTITY : %d\n",temp.quantity);

printf("COST : %.2f\n",temp.pcost);

printf("TOTAL COST : %.2f\n\n",temp.tcost);

printf("YOUR ORDER HAS BEEN DISPATCHED FROM OUR %s BRANCH",branch);

}

else printf("\n\nPRODUCT WILL BE DISPATCHED SOON\n\n");

break;

case 3:printf("CHOOSE WHERE YOUR PRODUCT IS SHIPPED FROM...\n\n");

int graph[maxVertices][maxVertices],size[maxVertices]={0},visited[maxVertices]={0};

int distance[maxVertices][maxVertices];

int vertices,edges,iter,jter,z;

/\* vertices represent number of vertices and edges represent number of edges in the graph. \*/

//printf("enter the no of vertices and edges\n");

//scanf("%d%d",&vertices,&edges);

char places[20][20]={"Alappuzha","Idukki","Thodupuzha","Manomboly","Guruvayoor"};

printf("\n\nWE ARE AVAILABLE AT\n\n");z=0;

for(i=0;i<5;i++){

printf("%s\n",places[i]);

}

int i=5,j=i;int k;

char mylocation[20];

vertices=i;edges=i;

printf("\nEnter your location\n");

scanf("%s",places[i++]);

strcpy(mylocation,places[j]);

system("cls");

printf("\nADMINISTRATOR\n SET THE DISTANCE FOR CUSTOMER'S LOCATION %s\n",mylocation);

printf("\n\nOUR BRANCHES\n");

for(i=0;i<5;i++){

printf("%s\n",places[i]);

}

/\*initialize distance between all pairs as infinity\*/

init(distance);

int weight;

char vertex1[10],vertex2[10];int v1,v2;

int min=INF,position=0;

/\* Here graph[i][j] represent the weight of edge joining i and j \*/

for(iter=0;iter<edges;iter++)

{

printf("Enter the places and the distance\n");

scanf("%s\n%s\n%d",vertex1,vertex2,&weight);

// assert(vertex1>=0 && vertex1<vertices);

//assert(vertex2>=0 && vertex2<vertices);

for(k=0;k<j+1;k++){

if(!strcmp(vertex1,places[k]))v1=k;

if(!strcmp(vertex2,places[k]))v2=k;

}

graph[v1][v2] = weight;

distance[v1][v2] = weight;

}

system("cls");

printf("WELCOME BACK CLIENT\n");

FloydWarshall(distance,vertices);

printf("Distance from %s\n",mylocation);

for(jter=0;jter<vertices;jter++)

{

if(distance[j][jter]<min){

min=distance[j][jter];

position=jter;

}

printf(" to %s : %d\n",places[jter],distance[j][jter]);

}

j=i;

printf("We would suggest you to choose the nearest location:%s",places[position]);

strcpy(branch,places[position]);

getch();

system("cls");

break;

case 4:inorder(root);break;

case 5:break;

}}while(c!=5);

break;

case 3:printf("WELCOME ADMINISTRATOR\n");

int choice;

char ans='N';

int key;

node \*new\_node,\*tmp,\*parent;

node \*get\_node();

root=NULL;

printf("\nADMINISTRATOR \n");

do

{

printf("\n1.MAKE AN ENTRY FOR NEW EMPLOYEE");

printf("\n2.DETAILS OF AN EMPLOYEE");

printf("\n3.DISPLAY THE ENTIRE STAFF DETAILS");

printf("\n4.EXIT");

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

scanf("%d",&choice);

switch(choice)

{

case 1:

do

{

new\_node=get\_node();

printf("\nInformation about employee ");

printf("\n Name: ");

scanf("%s",&new\_node->name);

printf("\n Rank: ");

scanf("%s",&new\_node->position);

printf("\n Id. Number: ");

scanf("%d",&new\_node->id);

printf("\n Monthly salary: ");

scanf("%d",&new\_node->salary);

printf("\n Contact number:");

scanf("%ld",&new\_node->data);

if(root==NULL) /\* Tree is not Created \*/

root=new\_node;

else

insertll(root,new\_node);

printf("\nDo you wish to enter a new employe?(y/n)");

ans=getch();

}while(ans=='y');

break;

case 2:

printf("\nEnter employee id to load the details:\n ");

scanf("%d",&key);

tmp = search(root,key,&root);

printf("\n%s\n %s\n %d\n %ld\n",

tmp->name,tmp->position,tmp->salary,tmp->data);

break;

case 3:

if(root==NULL)

printf(" ");

else

{

printf("\nCONTACT US: ");

inorder(root);

}

break;

}

}while(choice!=4);

break;

}

printf("CONTINUE EXPLORING..\n");

scanf(" %c",&choice);

system("cls");

}

}

**PRIORQUEUE.H**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct pq{

char pname[50];

int ct,quantity,id;

float pcost,tcost;

}pq;

typedef struct PriorityQ

{

int size;

int capacity;

pq \*arr;

}PQ;

PQ\* createPQ(int max)

{

PQ \*Q;

Q=(PQ\*)malloc(sizeof(PQ));

Q->capacity=max;

Q->arr=(pq\*)malloc(sizeof(pq)\*max);

strcpy(Q->arr[0].pname,"No");

Q->arr[0].ct=0;

Q->arr[0].id=0;

Q->arr[0].quantity=0;

Q->arr[0].pcost=0.0;

Q->arr[0].tcost=0.0;

Q->size=1;

return Q;

}

void insert(PQ \*Q)

{

int a;

static int number=1;

pq temp;

if(Q->size==Q->capacity)

{

return;

}

else

{

a=Q->size;

Q->size++;

printf("Enter the product name:\n");

scanf("%s",Q->arr[a].pname);

printf("Enter the quantity and cost:\n");

scanf("%d %f",&Q->arr[a].quantity,&Q->arr[a].pcost);

Q->arr[a].id=number++;

Q->arr[a].tcost=Q->arr[a].quantity\*Q->arr[a].pcost;

while(a!=1 && Q->arr[a].tcost > Q->arr[a/2].tcost)

{

temp=Q->arr[a];

Q->arr[a]=Q->arr[a/2];

Q->arr[a/2]=temp;

a=a/2;

}

}

}

pq deletemax(PQ \*Q)

{

int i,c;pq temp,last,max;

max=Q->arr[1];

last=Q->arr[Q->size-1];

Q->arr[1]=last;

--Q->size;

for(int j=0;j<Q->size;j++)

{

for(i=1;(i\*2)<=Q->size;i=c)

{

c=i\*2;

if(c!=Q->size && Q->arr[c+1].tcost > Q->arr[c].tcost)

c++;

if(Q->arr[i].tcost < Q->arr[c].tcost)

{

temp=Q->arr[i];

Q->arr[i]=Q->arr[c];

Q->arr[c]=temp;

}

}

}

return max;

}

**QUEUES.H**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

struct Node {

char task[50];

char time[50];

struct Node\* next;

};

struct Node\* front = NULL;

struct Node\* rear = NULL;

void Enqueue(char x[],char y[]) {

struct Node\* temp =

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

strcpy(temp->task,x);

strcpy(temp->time,y);

temp->next = NULL;

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

front = rear = temp;

return;

}

rear->next = temp;

rear = temp;

}

void Dequeue() {

struct Node\* temp = front;

if(front == NULL) {

printf("Queue is Empty\n");

return;

}

if(front == rear) {

front = rear = NULL;

}

else {

front = front->next;

}

free(temp);

}

void Print() {

struct Node\* temp = front;

while(temp != NULL) {

printf("You have ");

puts(temp->task);printf("\t");

printf("at ");

puts(temp->time);

temp = temp->next;

}

printf("\n");

}

**GRAPHS.H**

#include<stdio.h>

#include<string.h>

#define maxVertices 100

#define INF 123456789

int min(int a,int b)

{

return (a<b)?a:b;

}

void init(int distance[maxVertices][maxVertices])

{

int iter,jter;

for(iter=0;iter<maxVertices;iter++)

{

for(jter=0;jter<maxVertices;jter++)

{

if(iter==jter)

{

distance[iter][jter] = 0;

}

else

{

distance[iter][jter] = INF;

}

}

}

}

void FloydWarshall(int distance[maxVertices][maxVertices],int vertices)

{

int from,to,via;

for(from=0;from<vertices;from++)

{

for(to=0;to<vertices;to++)

{

for(via=0;via<vertices;via++)

{

distance[from][to] = min(distance[from][to],

distance[from][via]+distance[via][to]);

}

}

}

}

**GRAPHICS.H**

#include <stdio.h>

#include <string.h>

void star()

{

int i, j, n;

char name[]="BUSINESS WHALE";

int len;

n=10;

len = strlen(name);

// upper heart

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

{

for(j=1; j<n-i; j+=2)

{

printf(" ");

}

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

{

printf("\*");

}

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

{

printf(" ");

}

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

{

printf("\*");

}

printf("\n");

}

// Prints lower triangle

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

{

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

{

printf(" ");

}

// Print the name inside

if(i == n)

{

for(j=1; j<=(n \* 2-len)/2; j++)

{

printf("\*");

}

printf("%s", name);

for(j=1; j<(n\*2-len)/2; j++)

{

printf("\*");

}

}

else

{

for(j=1; j<=(i\*2)-1; j++)

{

printf("\*");

}

}

printf("\n");

}

return ;

}

**BINARYTREE.H**

# include <stdio.h>

# include <conio.h>

# include <stdlib.h>

typedef struct BST

{

long int data;

char name[20];

char position[20];

int id;

int salary;

struct BST \*lchild,\*rchild;

}node;

node \*get\_node()

{

node \*temp;

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

temp->lchild=NULL;

temp->rchild=NULL;

return temp;

}

/\*

This function is for creating a binary search tree

\*/

void insertll(node \*root,node \*new\_node)

{

if(new\_node->salary < root->salary)

{

if(root->lchild==NULL)

root->lchild = new\_node;

else

insertll(root->lchild,new\_node);

}

if(new\_node->salary > root->salary)

{

if(root->rchild==NULL)

root->rchild=new\_node;

else

insertll(root->rchild,new\_node);

}

}

/\*

This function is for searching the node from

binary Search Tree

\*/

node \*search(node \*root,int key,node \*\*parent)

{

node \*temp;

temp=root;

while(temp!=NULL)

{

if(temp->id==key)

{

printf("\n Employee with number %d is present",temp->id);

return temp;

}

\*parent=temp;

if(temp->id>key)

temp=temp->lchild;

else

temp=temp->rchild;

}

return NULL;

}

/\*

This function displays the tree in inorder fashion

\*/

void inorder(node \*temp)

{

if(temp!=NULL)

{

inorder(temp->lchild);

printf("\n%s\n%s\n :%ld\n",temp->name,temp->position,temp->data);

inorder(temp->rchild);

}

}

**CONCLUSION:**

Business management system is built using the data structures queues, graphs ,priority queues and binary search trees.