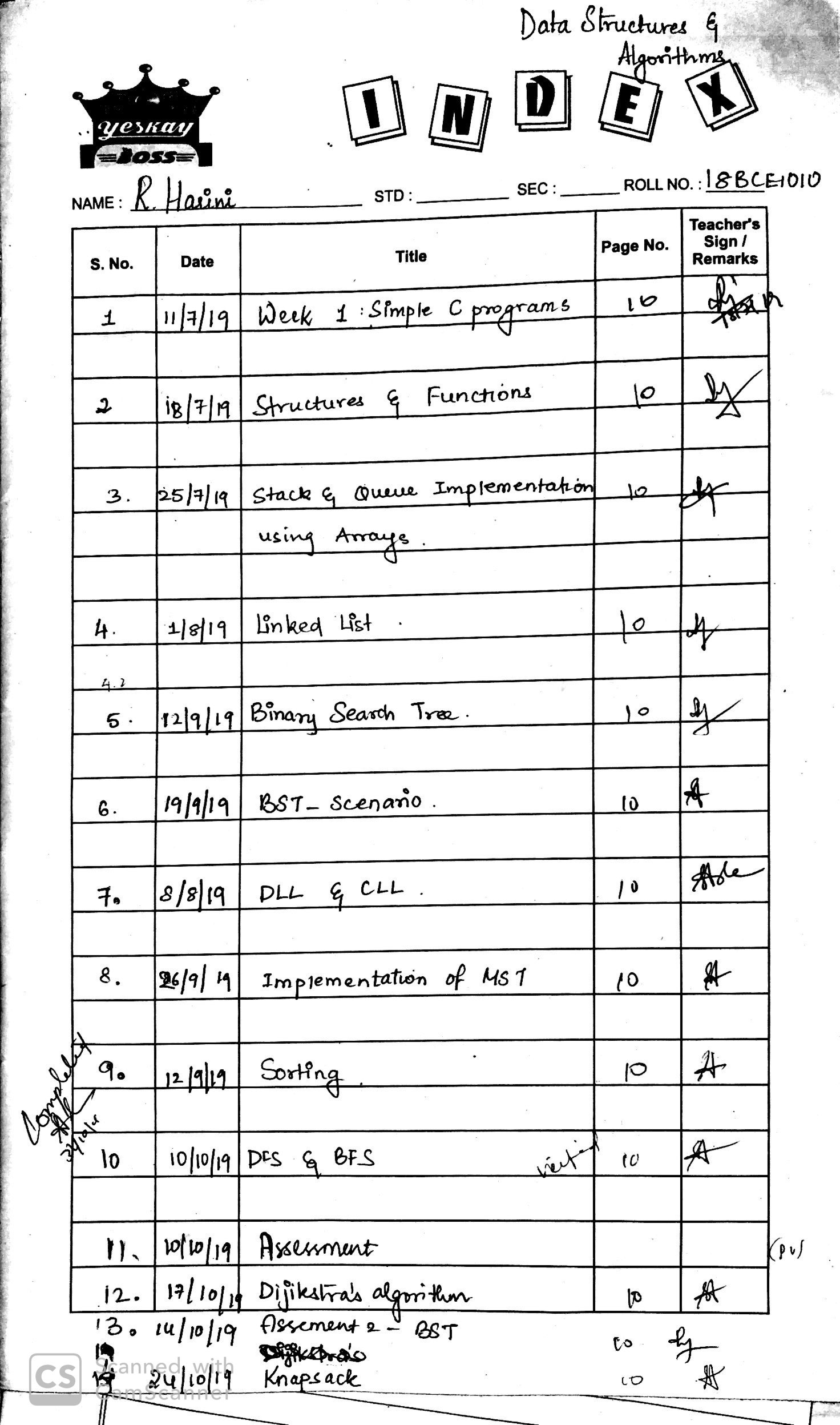
**R.HARINI**

**18BCE1010**

**DSA LAB CODES**

**Prof. Rajalakshmi**

****

**Lab 1**

1)

#include<stdio.h>

void main(){

int a[10],n,h=0,x;

printf("Enter n:");

scanf("%d",&n);

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

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

}

if (n%2==0){

x=(n/2)+1;

}

else{

x=n/2;

}

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

if (a[i]!=a[n-i-1]){

h++;

}

}

if (h!=0){

printf ("Not same");

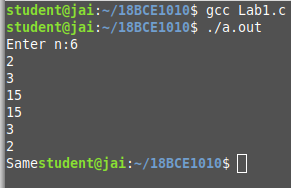
}

else{

printf("Same");

}

}



2)

#include<stdio.h>

void main(){

int a[10],a1[10],a2[10];

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

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

}

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

a1[i]=a[i];

}

int j=0;

for (int i=5;i<10;i++){

a2[j]=a[i];

j++;

}

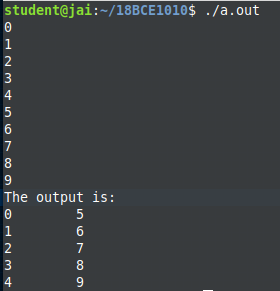
printf("The output is: \n");

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

printf("%d\t %d \n",a1[i],a2[i]);

}

}



3)

#include<stdio.h>

void main(){

int n,x,y;

printf("Enter n:");

scanf("%d",&n);

printf ("Enter x and y:");

scanf("%d %d",&x,&y);

int a[n];

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

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

}

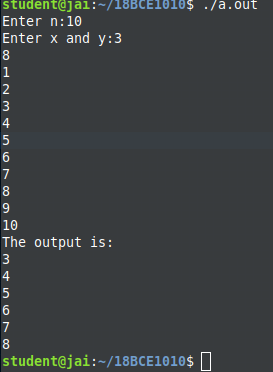
printf("The output is: \n");

for (int i=x;i<y+1;i++){

printf("%d \n",a[i-1]);

}

}



4)

#include<stdio.h>

void main(){

int n,sum;

printf("Enter n:");

scanf("%d",&n);

int a[n],b[n];

printf("Enter the elements of the array:");

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

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

}

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

sum=0;

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

if(j==i){

continue;

}

else{

sum+=a[j];

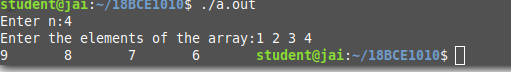
}

}

printf("%d \t", sum);

}

}



**Lab 2**

1)

#include<stdio.h>

void main(){

int a[10],n,h=0,x;

printf("Enter n:");

scanf("%d",&n);

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

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

}

if (n%2==0){

x=(n/2)+1;

}

else{

x=n/2;

}

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

if (a[i]!=a[n-i-1]){

h++;

}

}

if (h!=0){

printf ("Not same");

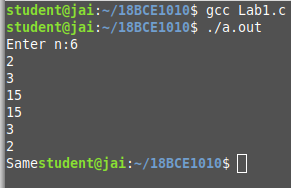
}

else{

printf("Same");

}

}



2)

#include<stdio.h>

void main(){

int a[10],a1[10],a2[10];

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

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

}

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

a1[i]=a[i];

}

int j=0;

for (int i=5;i<10;i++){

a2[j]=a[i];

j++;

}

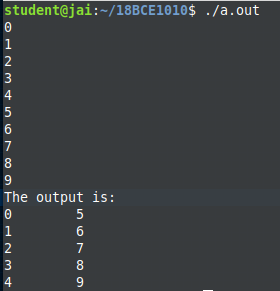
printf("The output is: \n");

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

printf("%d\t %d \n",a1[i],a2[i]);

}

}



3)

#include<stdio.h>

void main(){

int n,x,y;

printf("Enter n:");

scanf("%d",&n);

printf ("Enter x and y:");

scanf("%d %d",&x,&y);

int a[n];

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

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

}

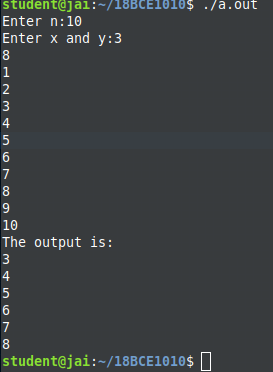
printf("The output is: \n");

for (int i=x;i<y+1;i++){

printf("%d \n",a[i-1]);

}

}



4)

#include<stdio.h>

void main(){

int n,sum;

printf("Enter n:");

scanf("%d",&n);

int a[n],b[n];

printf("Enter the elements of the array:");

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

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

}

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

sum=0;

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

if(j==i){

continue;

}

else{

sum+=a[j];

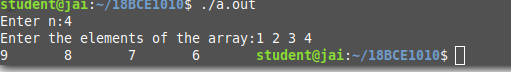
}

}

printf("%d \t", sum);

}

}



**Lab 3**

a)Implementing stacks using arrays

#include<stdio.h>

#define MAX 5

int stack[20],top=-1;

int is\_empty(int\* s){

if (top==-1){

return 1;

}

else{

return 0;

}

}

int is\_full(int\* s){

if (top==MAX){

return 1;

}

else{

return 0;

}

}

void push(int\* s, int i){

if (is\_full(s)){

printf("Stack is full\n");

}

else{

top++;

s[top]=i;

}

}

void pop(int\*s){

if (is\_empty(s)){

printf("Stack is empty\n");

}

else{

printf("%d\n",s[top]);

top--;

}

}

void display(int\* s){

if (is\_empty(s)){

printf("Stack is empty\n");

}

else{

printf("Displaying the stack:\n");

for (int i=0;i<top+1;i++){

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

}

}

}

int main(){

push(stack,5);

push(stack,4);

push(stack,3);

push(stack,2);

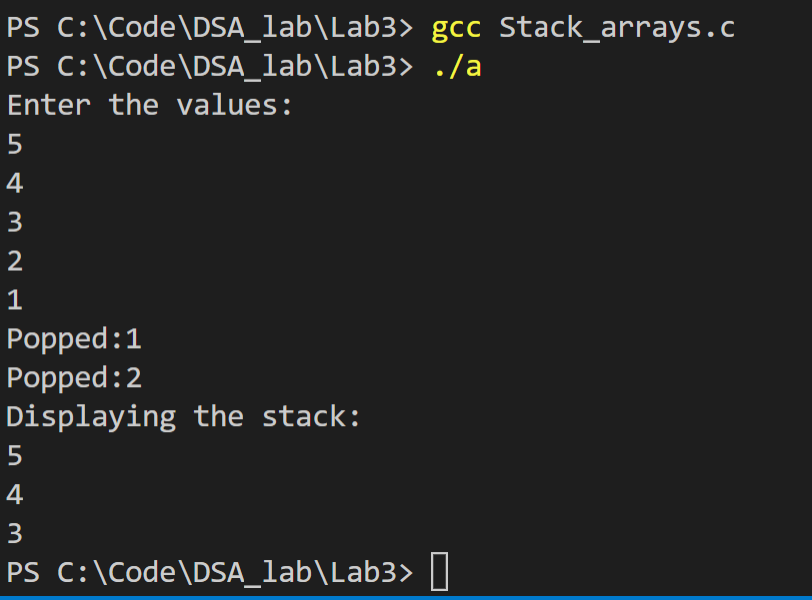
push(stack,1);

pop(stack);

pop(stack);

display(stack);

}



b) Balancing Parenthesis

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#define MAX 5

struct Stack{

int top;

char a[30];

};

int is\_empty(struct Stack\* s){

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

return 1;

}

else{

return 0;

}

}

int is\_full(struct Stack\* s){

if (s->top==MAX){

return 1;

}

else{

return 0;

}

}

void push(struct Stack\* s, char i){

if (is\_full(s)){

printf("Stack is full\n");

}

else{

s->top++;

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

}

}

void pop(struct Stack\* s){

if (is\_empty(s)){

printf("Stack is empty\n");

}

else{

//printf("%c",s->a[s->top]);

s->top--;

}

}

void display(struct Stack\* s){

for (int i=0;i<(s->top+1);i++){

printf("%c\n",s->a[i]);

}

}

int main(){

struct Stack\* s=(struct Stack\*)malloc(sizeof(struct Stack));

s->top=-1;

char p[30];

scanf("%s",p);

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

if (p[i]=='('){

push(s,p[i]);

}

else if (p[i]==')'){

pop(s);

}

else{

continue;

}

}

if (is\_empty(s)){

printf("Balanced");

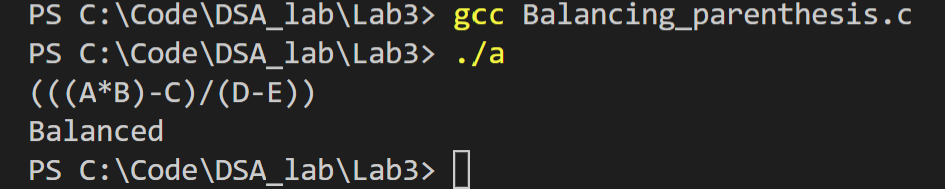
}

else{

printf("Not balanced");

}

}



c) Implementing Queues using arrays

#include<stdio.h>

#define MAX 5

int queue[30],front=-1, rear=-1;

int is\_empty(int\* q){

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

return 1;

}

else{

return 0;

}

}

int is\_full(int\* q){

if (rear==MAX-1){

return 1;

}

else{

return 0;

}

}

void enqueue(int\* q, int i){

if (is\_full(q)){

printf("Queue is full\n");

}

else{

rear++;

q[rear]=i;

}

}

void dequeue(int\* q){

if (is\_empty(q)){

printf("Queue is empty\n");

}

else{

front++;

printf("%d\n",q[front]);

}

}

void display(int\* q){

if (is\_empty(q)){

printf("Queue is empty\n");

}

else{

printf("Displaying the queue:\n");

for (int i=front+1;i<rear+1;i++){

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

}

}

}

int main(){

enqueue(queue,5);

enqueue(queue,4);

enqueue(queue,3);

enqueue(queue,2);

enqueue(queue,1);

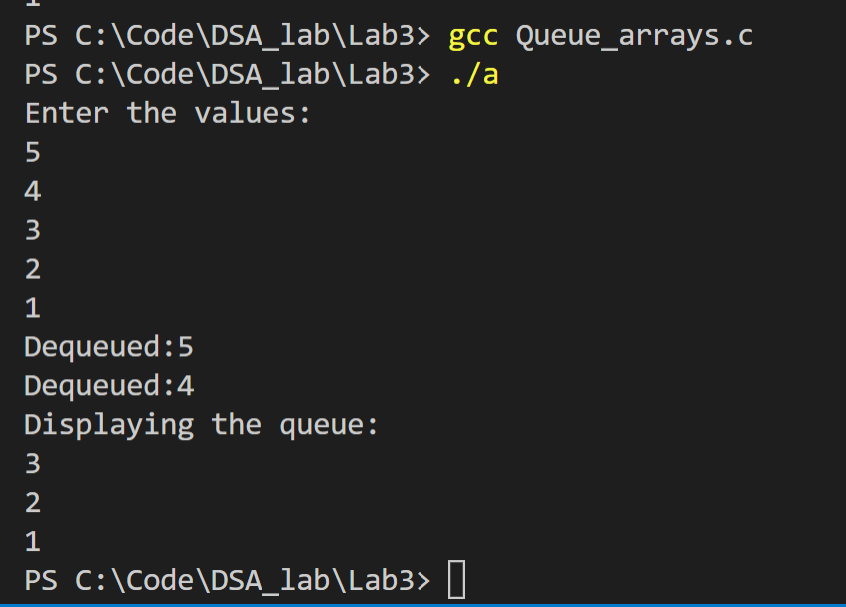
dequeue(queue);

dequeue(queue);

display(queue);

return 0;

}



d) Word problem using Disks (uses stacks and queues)

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MAX 5

struct Disks{

char type[10];

int ID;

int year;

char mname[10];

};

struct Stack{

int top;

struct Disks\* a[100];

};

int is\_empty(struct Stack\* s){

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

return 1;

}

else{

return 0;

}

}

int is\_full(struct Stack\* s){

if (s->top==(MAX-1)){

return 1;

}

else{

return 0;

}

}

void push(struct Stack\* s, struct Disks\* d){

if (is\_full(s)){

printf("Stack is full\n");

}

else{

s->top++;

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

}

}

struct Disks\* pop(struct Stack\* s){

if (is\_empty(s)){

printf("Stack is empty\n");

}

else{

struct Disks\* d=(struct Disks\*)malloc(sizeof(struct Disks));

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

//printf("%s\n%d\n%d\n%s\n", d->type, d->ID, d->year, d->mname);

s->top--;

return d;

}

}

void display\_s(struct Stack\* s){

for (int i=0;i<(s->top+1);i++){

struct Disks\* d=(struct Disks\*)malloc(sizeof(struct Disks));

d=s->a[i];

printf("%s\n%d\n%d\n%s\n ", d->type, d->ID, d->year, d->mname);

}

}

struct Queue{

int front;

int rear;

struct Disks\* a[50];

};

int qis\_full(struct Queue\* q){

if (q->front==q->rear && q->rear>-1){

return 1;

}

else{

return 0;

}

}

int qis\_empty(struct Queue\* q){

if (q->rear==-1){

return 1;

}

else{

return 0;

}

}

void enqueue(struct Queue\* q, struct Disks\* d){

if (qis\_full(q)){

printf("Queue is full\n");

}

else{

q->rear++;

q->a[q->rear]=d;

}

}

void dequeue(struct Queue\* q){

if (qis\_empty(q)){

printf("Queue is empty\n");

}

else{

q->front++;

struct Disks\* d=(struct Disks\*)malloc(sizeof(struct Disks));

d=q->a[q->front];

printf("%s\n%d\n%d\n%s\n ", d->type, d->ID, d->year, d->mname);

}

}

void display\_q(struct Queue\* q){

if (qis\_empty(q)){

printf("Queue is empty\n");

}

else{

for (int i=(q->front+1);i<(q->rear+1);i++){

struct Disks\* d=(struct Disks\*)malloc(sizeof(struct Disks));

d=q->a[i];

printf("%s\n%d\n%d\n%s\n ", d->type, d->ID, d->year, d->mname);

}

}

}

void check\_year(struct Queue\* q, struct Disks\* d){

if (d->year==2015 && strcmp(d->type, "CD")==0){

printf("\n%s is from 2015\n",d->mname);

enqueue(q,d);

}

}

void sort\_disks(struct Disks\* d,int n){

struct Disks t;

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

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

if (d[i].year>d[j].year){

t=d[i];

d[i]=d[j];

d[j]=t;

}

}

}

}

void move\_DVD\_stack(struct Stack\* s, struct Disks\* d){

if (strcmp(d->type,"DVD")==0){

push(s,d);

printf("\n%s is moved to the stack\n", d->mname);

}

}

void check\_2018(struct Stack\* s, struct Stack\* s1,struct Queue\* q){

struct Disks\* d=(struct Disks\*)malloc(sizeof(struct Disks));

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

d=pop(s);

if (strcmp(d->type,"DVD")==0 && d->year==2018){

printf("\n%s is added to the queue\n",d->mname);

enqueue(q,d);

}

else{

push(s1,d);

}

}

}

int main(){

struct Stack\* s=(struct Stack\*)malloc(sizeof(struct Stack));

s->top=-1;

struct Stack\* s1=(struct Stack\*)malloc(sizeof(struct Stack));

s1->top=-1;

struct Queue\* q=(struct Queue\*)malloc(sizeof(struct Queue));

q->rear=-1;

q->front=-1;

struct Queue\* q2=(struct Queue\*)malloc(sizeof(struct Queue));

q2->rear=-1;

q2->front=-1;

int n;

printf("Enter n:");

scanf("%d",&n);

struct Disks d[n];

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

printf("Enter the type(CD/DVD), ID, year, mname:\n");

scanf("%s %d %d %s",d[i].type, &d[i].ID, &d[i].year, d[i].mname);

}

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

check\_year(q,&d[i]);

}

sort\_disks(d,n);

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

move\_DVD\_stack(s,&d[i]);

}

check\_2018(s,s1,q2);

return 0;

}

**Lab 4**

**1)**

#include<stdio.h>

#include<stdlib.h>

struct Node{

int data;

struct Node\* next;

};

void display(struct Node\* head){

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

temp=head->next;

printf("Displaying the linked list\n");

while(temp!=NULL){

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

temp=temp->next;

}

}

void add(struct Node\* head, int i){

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

newnode->data=i;

newnode->next=NULL;

if (head->next==NULL){

head->next=newnode;

return;

}

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

temp=head->next;

while (temp->next!=NULL){

temp=temp->next;

}

temp->next=newnode;

}

void insert(struct Node\* head,int a, int i){

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

newnode->data=i;

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

temp=head->next;

while (temp!=NULL){

if (temp->data==a){

newnode->next=temp->next;

temp->next=newnode;

return;

}

else{

temp=temp->next;

}

}

printf("Node not found\n");

}

void find(struct Node\* head,int i){

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

temp=head->next;

while(temp!=NULL){

if(temp->data==i){

printf("%d found\n",i);

return;

}

else{

temp=temp->next;

}

}

printf("Not found\n");

}

int o=0,e=0;

void odd\_even(struct Node\* head){

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

temp=head->next;

while(temp!=NULL){

if (temp->data%2==0){

e++;

}

else{

o++;

}

temp=temp->next;

}

}

struct Node\* deletenode(struct Node\* head,int i){

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

if (head->next->data==i){

del=head->next;

head->next=del->next;

return del;

}

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

temp=head->next;

while(temp->next!=NULL){

if (temp->next->data==i){

del=temp->next;

temp->next=del->next;

return del;

}

temp=temp->next;

}

}

void remove\_odd(struct Node\* head, struct Node\* head1){

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

temp=head->next;

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

while (temp!=NULL){

if ((temp->data)%2==1){

n=deletenode(head, temp->data);

add(head1,n->data);

}

temp=temp->next;

}

}

int main(){

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

head->next=NULL;

struct Node\* head1=(struct Node\*)malloc(sizeof(struct Node));

head1->next=NULL;

int c=1,n,a;

int i;

while (c>0){

printf("\*\*\*\*MENU\*\*\*\*\*\n");

printf("1->Add a number\n2->Display the list\n3->Insert a number\n");

printf("4->Find a number\n5->Check how many odd or even numbers\n");

printf("6->Delete a number\n7->Remove all odd nos\n0->End the process\n\*\*\*\*\*\*\n");

scanf("%d",&c);

switch (c) {

case 1:

printf("Enter how many numbers you want to add:");

scanf("%d",&n);

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

printf("Enter the value:");

scanf("%d",&a);

add(head,a);

}

break;

case 2:

display(head);

break;

case 3:

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

scanf("%d",&a);

printf("Enter the number after which to be inserted:");

scanf("%d",&i);

insert(head,i,a);

break;

case 4:

printf("Enter the number to be found:");

scanf("%d",&a);

find(head,a);

break;

case 5:

odd\_even(head);

printf("Odd nos:%d\n Even nos:%d\n",o,e);

break;

case 6:

printf("Enter the number to be deleted:");

scanf("%d",&a);

deletenode(head,a);

break;

case 7:

remove\_odd(head,head1);

display(head1);

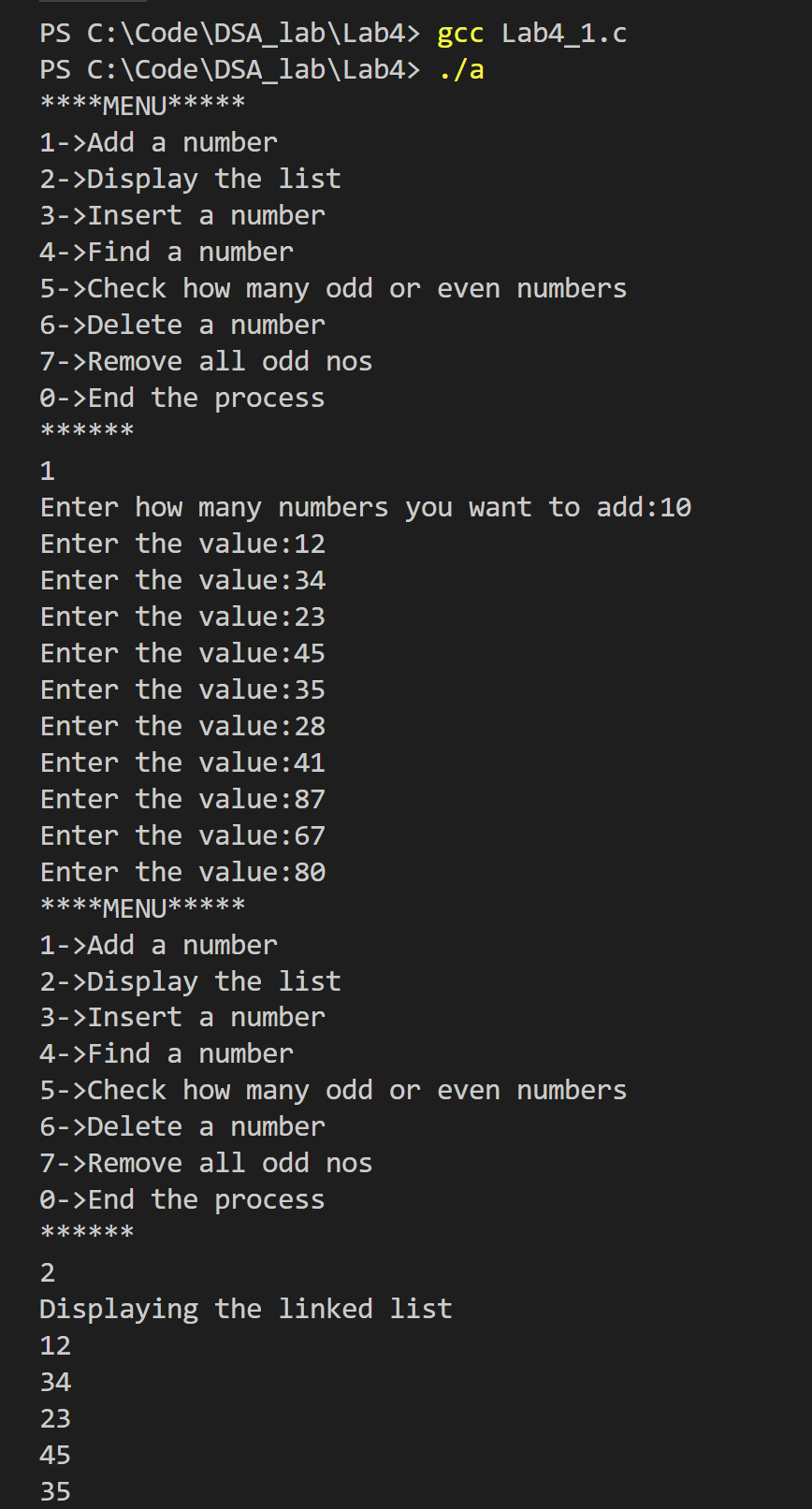
break;

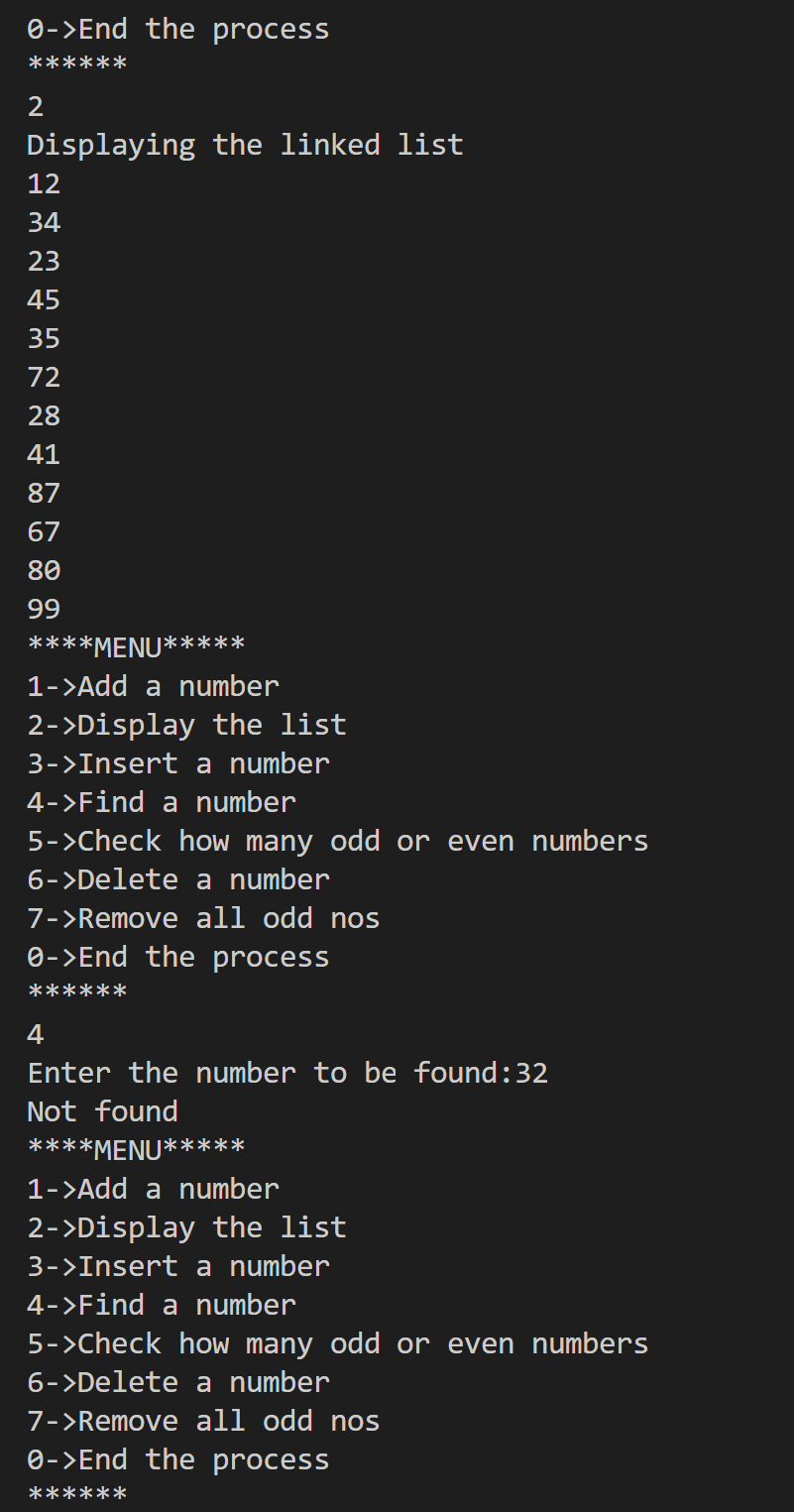
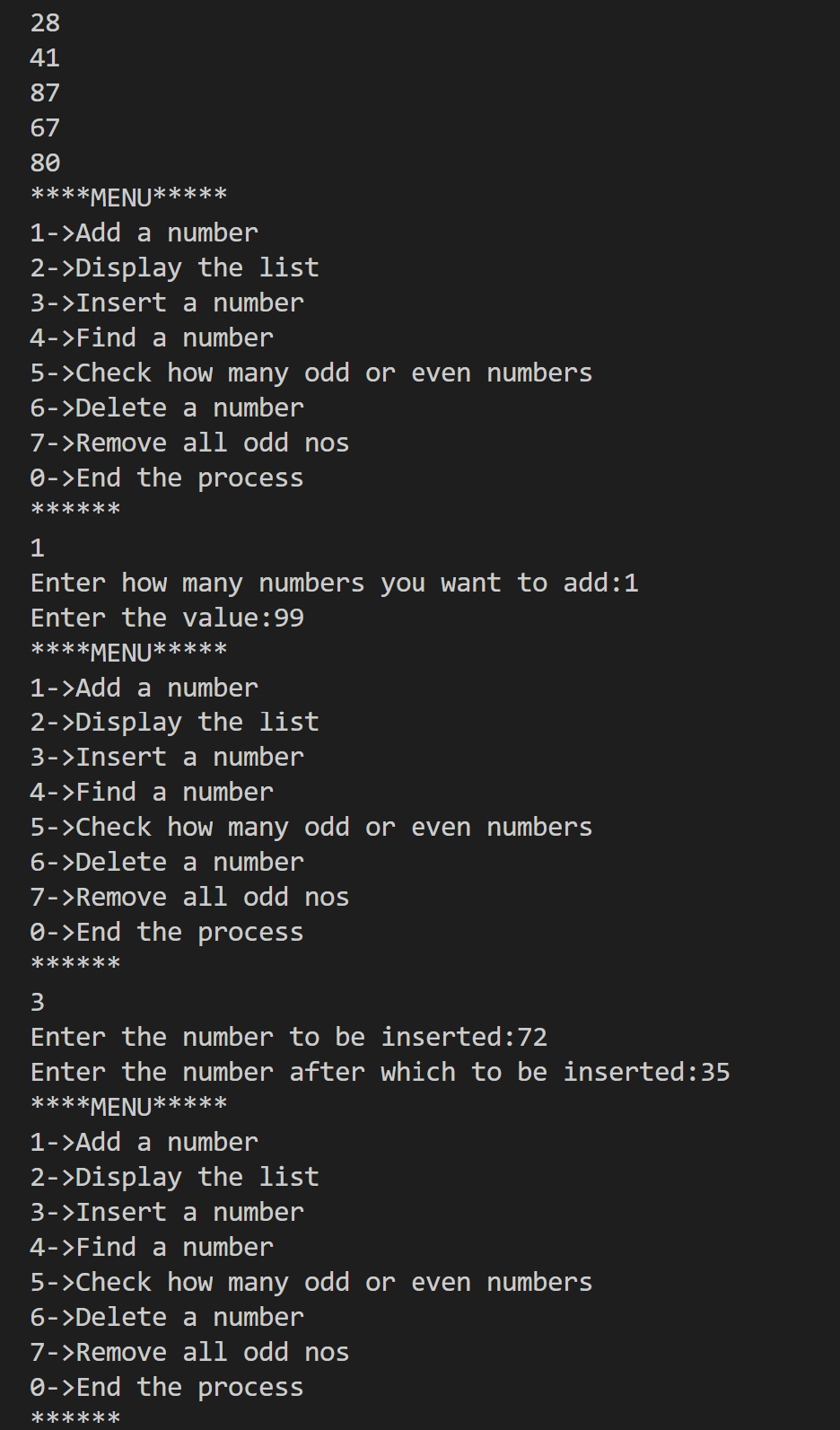
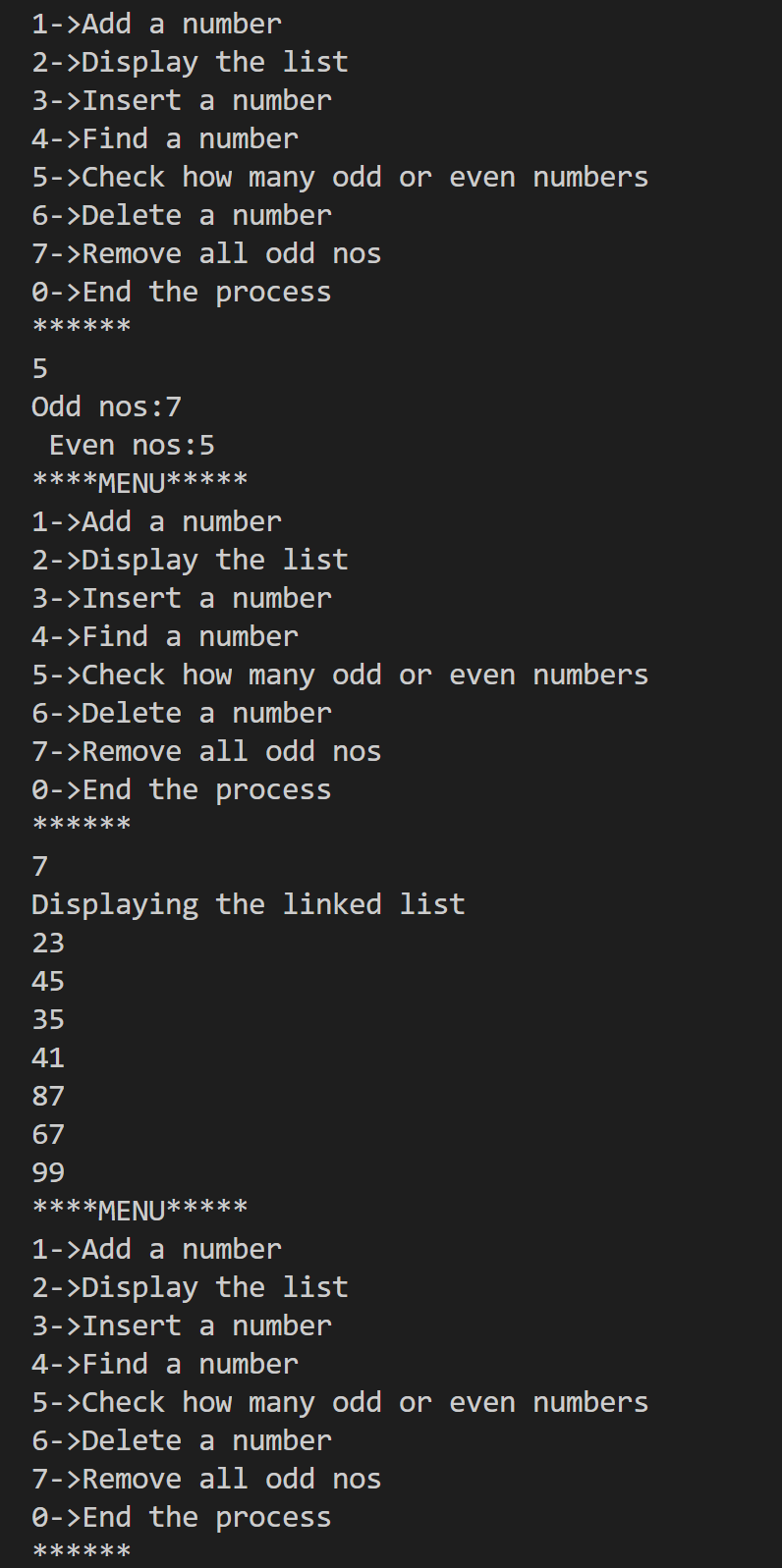
}

}

return 0;

}



**2)**

#include<stdio.h>

#include<stdlib.h>

#define MAX 10

struct Node{

int regno;

char name[50];

float cgpa;

struct Node\* next;

};

void display(struct Node\* head){

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

temp=head->next;

printf("Displaying the linked list\n");

while (temp!=NULL){

printf("%d\n%s\n%f\n",temp->regno,temp->name,temp->cgpa);

temp=temp->next;

}

}

void add(struct Node\* head, struct Node\* n){

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

if (head->next==NULL){

n->next=NULL;

head->next=n;

return;

}

temp=head->next;

while (temp->next!=NULL){

temp=temp->next;

}

temp->next=n;

n->next=NULL;

}

void search(struct Node\* head, int r){

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

temp=head->next;

while(temp!=NULL){

if (temp->regno==r){

printf("%s\n",temp->name);

return;

}

temp=temp->next;

}

printf("Register not found\n");

}

void update(struct Node\* head, int r, float cg){

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

temp=head->next;

while(temp!=NULL){

if(temp->regno==r){

temp->cgpa=cg;

return;

}

temp=temp->next;

}

}

void topper(struct Node\* head){

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

temp=head->next;

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

float t=0;

while (temp!=NULL){

if (temp->cgpa>t){

topper=temp;

t=temp->cgpa;

}

temp=temp->next;

}

printf("%d\n%s\n%f\n",topper->regno,topper->name,topper->cgpa);

}

struct Node\* del(struct Node\* head, int r){

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

if (head->next->regno==r){

d=head->next;

head->next=d->next;

return d;

}

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

temp=head->next;

while (temp->next!=NULL){

if (temp->next->regno==r){

d=temp->next;

temp->next=d->next;

return d;

}

temp=temp->next;

}

}

struct Queue{

int front;

int rear;

struct Node\* a[50];

};

int qis\_full(struct Queue\* q){

if (q->front==q->rear && q->rear>-1){

return 1;

}

else{

return 0;

}

}

int qis\_empty(struct Queue\* q){

if (q->rear==-1){

return 1;

}

else{

return 0;

}

}

void enqueue(struct Queue\* q, struct Node\* n){

if (qis\_full(q)){

printf("Queue is full\n");

}

else{

q->rear++;

q->a[q->rear]=n;

}

}

void dequeue(struct Queue\* q){

if (qis\_empty(q)){

printf("Queue is empty");

}

else{

q->front++;

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

n=q->a[q->front];

printf("%d\n%s\n%f",n->regno,n->name, n->cgpa);

}

}

void display\_q(struct Queue\* q){

if (qis\_empty(q)){

printf("Queue is empty\n");

}

else{

printf("Displaying the queue:");

for (int i=(q->front+1);i<(q->rear+1);i++){

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

n=q->a[i];

printf("%d\n%s\n%f", n->regno, n->name,n->cgpa);

}

}

}

void cgpa\_8(struct Node\* head, struct Queue\* q){

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

temp=head->next;

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

while (temp!=NULL){

if (temp->cgpa>8.5){

n=del(head, temp->regno);

enqueue(q,n);

}

temp=temp->next;

}

}

struct Stack{

int top;

struct Node\* a[50];

};

int is\_empty(struct Stack\* s){

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

return 1;

}

else{

return 0;

}

}

int is\_full(struct Stack\* s){

if (s->top==(MAX-1)){

return 1;

}

else{

return 0;

}

}

void push(struct Stack\* s, struct Node\* n){

if (is\_full(s)){

printf("Stack is full\n");

}

else{

s->top++;

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

}

}

void pop(struct Stack\* s){

if (is\_empty(s)){

printf("Stack is empty\n");

}

else{

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

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

printf("%d\n%s\n%f",n->regno,n->name,n->cgpa);

s->top--;

}

}

void display\_s(struct Stack\* s){

for (int i=0;i<(s->top+1);i++){

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

n=s->a[i];

printf("%d\n%s\n%f", n->regno, n->name, n->cgpa);

}

}

void Nine\_pointer(struct Node\* head, struct Stack\* s){

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

temp=head->next;

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

while (temp!=NULL){

if (temp->cgpa>9){

n=del(head, temp->regno);

push(s,n);

}

temp=temp->next;

}

}

int main(){

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

head->next=NULL;

int c=1,n,k=0;

struct Node newnode[30];

int r;

float cg;

struct Queue\* q=(struct Queue\*)malloc(sizeof(struct Queue));

q->rear=-1;

q->front=-1;

struct Stack\* s=(struct Stack\*)malloc(sizeof(struct Stack));

s->top=-1;

/\*newnode[0].regno=1010;

newnode[0].name="harini";

newnode[0].cgpa=9.3;

newnode[1].regno=1009;

newnode[1].name="madhu";

newnode[1].cgpa=8.3;

add(head,&newnode[0]);

add(head,&newnode[1]);

display(head);

cgpa\_8(head, q);

display(head);

display\_q(q);\*/

while(c>0){

printf("\*\*\*\*MENU\*\*\*\*\n");

printf("1->Add student details\n2->Display the list\n3->Search by regno\n");

printf("4->Search for topper\n5->CGPA>8.5\n6->Update CGPA\n");

printf("7->Display queue\n8->Nine pointer stack\n9>Display stack\n");

printf("0->End the process\n");

scanf("%d",&c);

switch(c){

case 1:

printf("Enter how many you want to add:");

scanf("%d",&n);

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

//struct Node\* newnode=(struct Node\*)malloc(sizeof(struct Node));

printf("Enter regno, name and cgpa:");

scanf("%d\n%s\n%f",&newnode[k].regno, newnode[k].name, &newnode[k].cgpa);

add(head,&newnode[k]);

k++;

}

break;

case 2:

display(head);

break;

case 3:

printf("Enter the register number:");

scanf("%d",&r);

search(head,r);

break;

case 4:

topper(head);

break;

case 5:

cgpa\_8(head,q);

break;

case 6:

printf("Enter the register number:");

scanf("%d",&r);

printf("Enter the updated cgpa:");

scanf("%f",&cg);

update(head,r,cg);

break;

case 7:

display\_q(q);

break;

case 8:

Nine\_pointer(head,s);

break;

case 9:

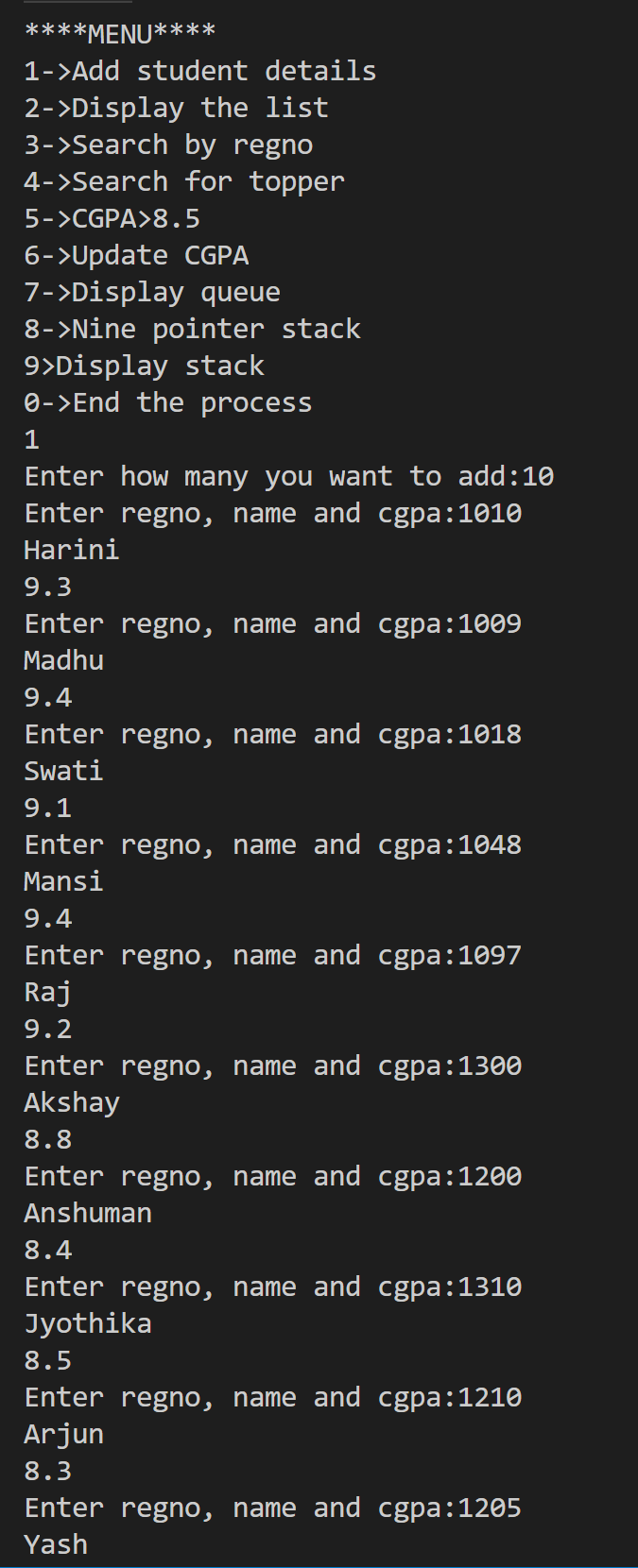
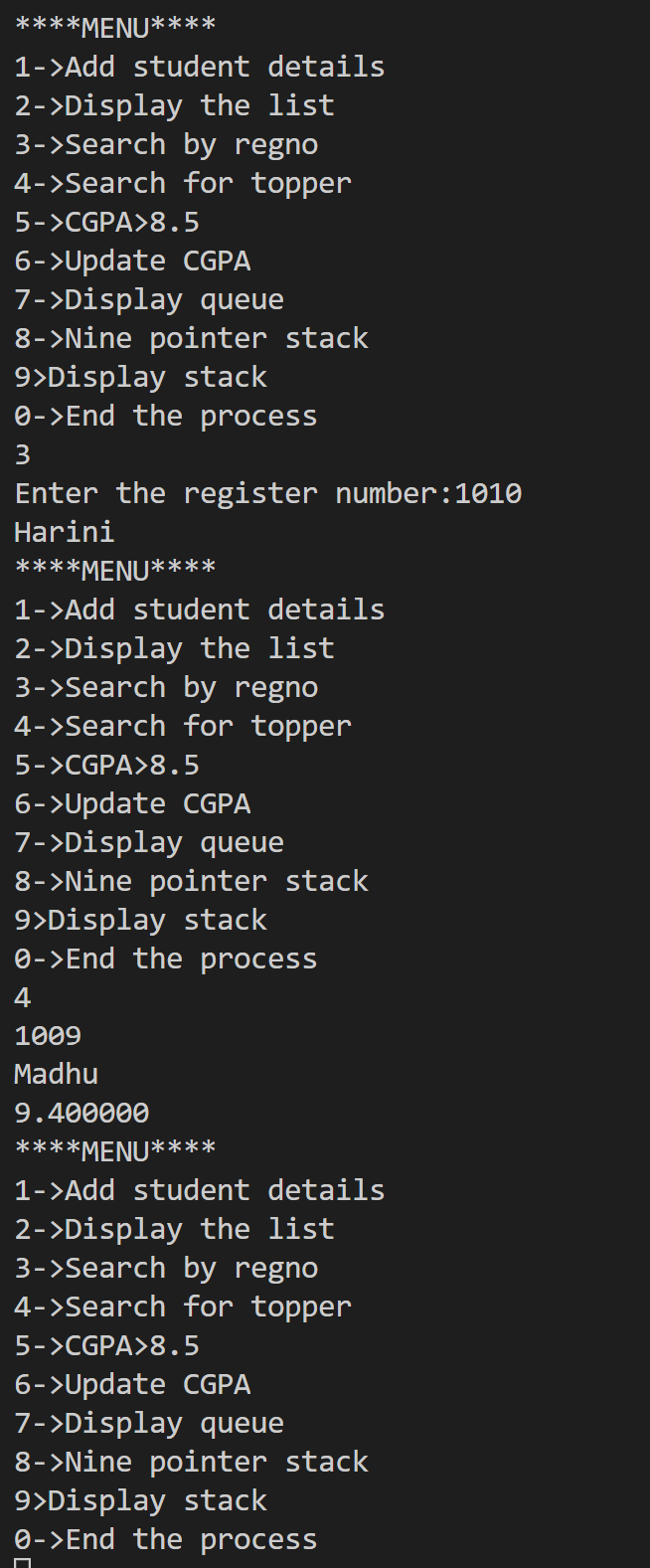
display\_s(s);

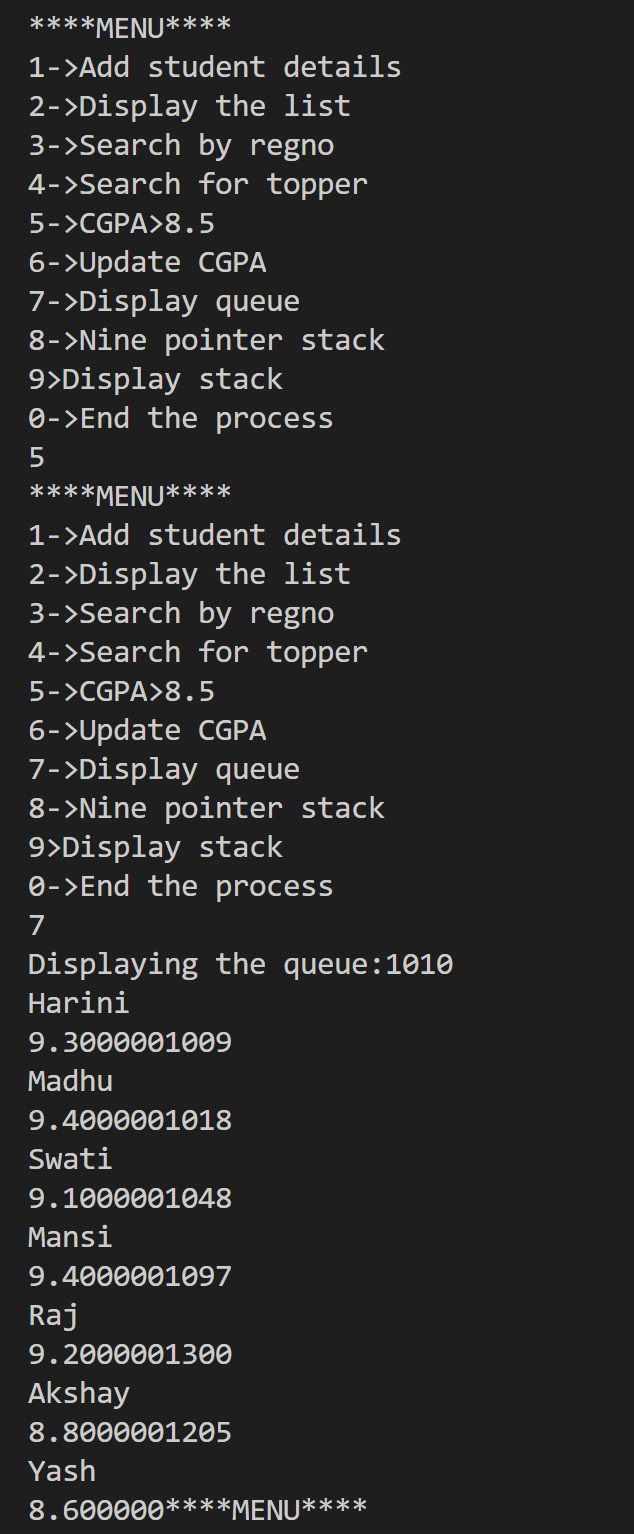
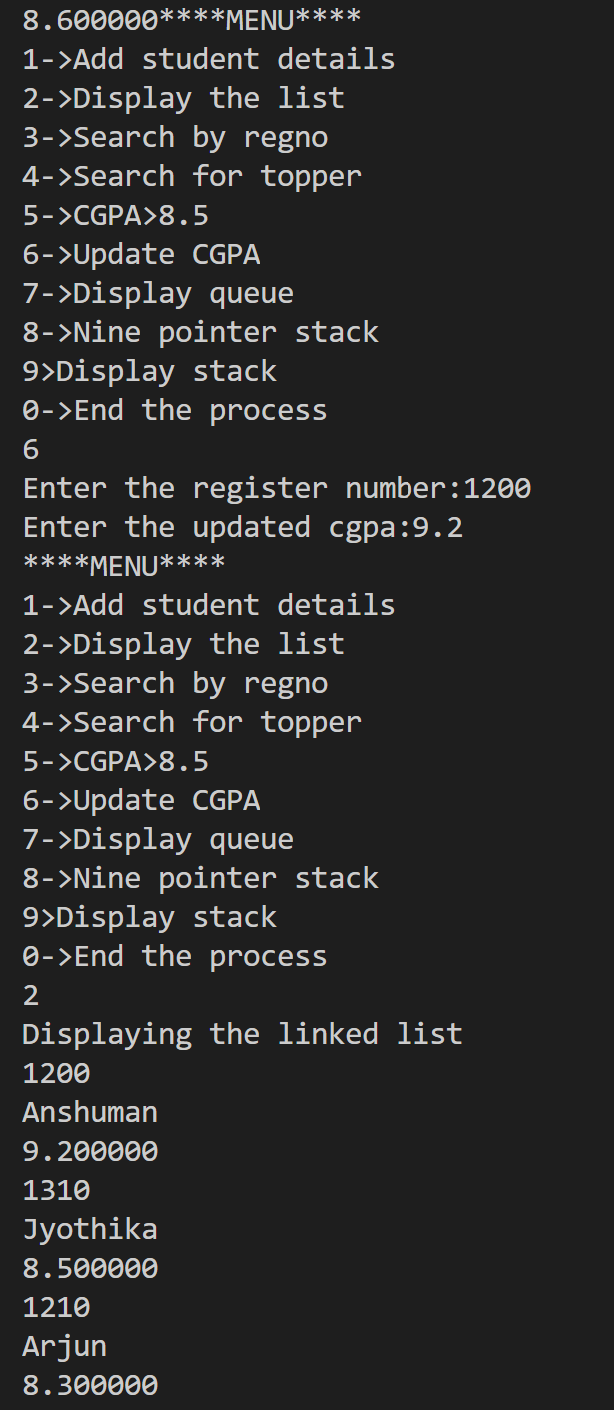
break;

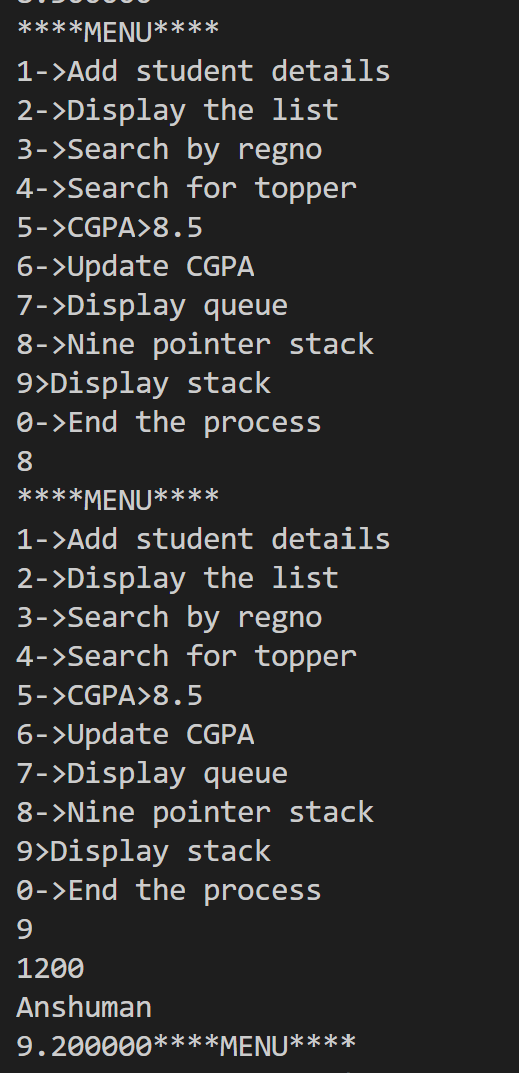
}

}

}



**Lab 5**

**BST Scenario Implementation**

**Code:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct Book{

int accno;

char title[50];

char status[50];

};

struct Node{

int eid;

char name[50];

struct Book b[10];

int tot;

struct Node\* rchild;

struct Node\* lchild;

};

struct Node\* root=NULL;

void add(){

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

printf("Enter the emp id:");

scanf("%d",&n->eid);

printf("Enter the name:");

scanf("%s", n->name);

printf("Enter the total books:");

scanf("%d",&n->tot);

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

printf("Enter the book access no:");

scanf("%d", &n->b[i].accno);

printf("Enter the book title:");

scanf("%s",n->b[i].title);

printf("Enter the status(Issued/Returned):");

scanf("%s",n->b[i].status);

}

struct Node\* temp;

struct Node\* parent;

n->lchild=NULL;

n->rchild=NULL;

if (root==NULL){

root=n;

}

else{

temp=root;

parent=NULL;

while (1){

parent=temp;

if (n->eid<parent->eid){

temp=temp->lchild;

if (temp==NULL){

parent->lchild=n;

return;

}

}

else{

temp=temp->rchild;

if (temp==NULL){

parent->rchild=n;

return;

}

}

}

}

}

void preorder(struct Node\* root){

if (root!=NULL){

printf("%d\n%s\n", root->eid,root->name);

preorder(root->lchild);

preorder(root->rchild);

}

}

void inorder(struct Node\* root){

if (root!=NULL){

inorder(root->lchild);

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

inorder(root->rchild);

}

}

struct Node\* find(struct Node\* root, int id){

struct Node\* n;

if (root->eid==id){

return root;

}

else{

if (id>root->eid){

n=find(root->rchild,id);

return n;

}

else{

n=find(root->lchild,id);

return n;

}

}

}

void return\_books(int id, int acc\_no){

struct Node\* n=find(root, id);

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

if (n->b[i].accno==acc\_no){

strcpy(n->b[i].status,"Returned");

struct Book temp=n->b[i];

n->b[i]=n->b[n->tot-1];

n->b[n->tot-1]=temp;

n->tot--;

}

}

printf("Total books: %d\n",n->tot);

printf("The returned book: %d\n", n->b[n->tot+1]);

}

struct Node\* FindMin(struct Node\* node){

struct Node\* temp=node;

while (temp && temp->lchild!=NULL){

temp=temp->lchild;

}

return temp;

}

struct Node\* del(struct Node \* root,int x) // Function to delete the nodes. It covers all the cases.

{

if(root == NULL)

;

else if(x < root->eid)

root->lchild = del(root->lchild,x);

else if(x > root->eid)

root->rchild = del(root->rchild,x);

else

{

if(root->lchild == NULL && root->rchild==NULL) // No child.

{

free(root);

root=NULL;

}

else if(root->lchild == NULL) // One child.

{

node \* k = root;

root = root->rchild;

free(k);

}

else if(root->rchild == NULL) // One child.

{

node \* k = NULL;

root = root->lchild;

free(k);

}

else // Two children.

{

node \* k = FindMin(root->rchild);

root->eid = k->eid;

root->rchild = del(root->rchild,k->eid);

}

}

}

int main(){

/\*add();

add();

add();

add();

add();

preorder(root);

int id;

printf("Enter the id to be found:");

scanf("%d",&id);

struct Node\* n=find(root,id);

printf("%s\n", n->name);

int accno;

printf("Enter the accno of the book:");

scanf("%d",&accno);

return\_books(id,accno);

n=find(root,id);

printf("%s\n", n->name);

printf("%d\n",n->tot);\*/

int c=1,id,accno;

while (c>0){

printf("\*\*\*\*\*\*MENU\*\*\*\*\*\*\n");

printf("1->Add\n2->Preorder traversal\n3->Inorder traversal\n");

printf("4->Find\n5->Return Books\n");

scanf("%d",&c);

switch(c){

case 1:

add();

break;

case 2:

preorder(root);

break;

case 3:

inorder(root);

break;

case 4:

printf("Enter the id to be found:");

scanf("%d",&id);

struct Node\* n=find(root,id);

printf("%s\n",n->name);

break;

case 5:

printf("Enter the emp id:");

scanf("%d",&id);

printf("enter the access no of the book:");

scanf("%d",&accno);

return\_books(id,accno);

break;

}

}

return 0;

}

**Lab :DLL**

#include<stdio.h>

#include<stdlib.h>

struct Node{

int ele;

struct Node\* next;

struct Node\* prev;

};

struct Node\* head=NULL;

struct Node\* tail=NULL;

void add(int i){

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

n->prev=NULL;

if (head==NULL){

n->next=NULL;

head=NULL;

tail=NULL;

return;

}

n->next=head;

head->prev=n;

head=n;

}

void display(){

struct Node\* temp=head;

if (head==NULL){

printf("List is empty\n");

return;

}

while (temp!=NULL){

}

}

**Lab 6:SORTING**

#include<stdio.h>

#include<stdlib.h>

int\* bubblesort(int\* a, int n){

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

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

if (a[i]>a[j]){

int t=a[i];

a[i]=a[j];

a[j]=t;

}

}

}

return a;

}

int\* insertionsort(int\* a, int n){

for (int j=1;j<n;j++){

int key=a[j];

int i=j-1;

while(i>=0 && a[i]>key){

a[i+1]=a[i];

i--;

}

a[i+1]=key;

}

return a;

}

int partition(int\* A, int p, int r){

int x=A[r];

int i=p-1;

for (int j=p;j<=r-1;j++){

if (A[j]<x){

i++;

int t=A[i];

A[i]=A[j];

A[j]=t;

}

}

int t=A[i+1];

A[i+1]=A[r];

A[r]=t;

return i+1;

}

int\* quicksort(int\* A, int p, int r){

if (p<r){

int q=partition(A,p,r);

quicksort(A,p,q-1);

quicksort(A,q+1,r);

}

return A;

}

void merge(int a[], 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]=a[l+i];

}

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

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

}

i=0;

j=0;

k=l;

while (i<n2 && j<n2){

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

a[k]=L[i];

i++;

}

else{

a[k]=R[j];

j++;

}

k++;

}

while (i<n1){

a[k]=L[i];

i++;

k++;

}

while (j<n2){

a[k]=R[j];

j++;

k++;

}

}

void mergesort(int a[], int l, int r){

if (l<r){

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

mergesort(a,l,m);

mergesort(a, m+1,r);

merge(a,l,m,r);

}

}

void print(int\* a, int n){

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

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

}

printf("\n");

}

int main(){

int a[9]={5,2,9,7,3,8,6,1,4};

int c=1;

while (c>0){

printf("\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\n");

printf("1->Bubble sort\n2->Insertion Sort\n3->Quick sort\n4->Merge sort\n");

scanf("%d",&c);

switch(c){

case 1:

bubblesort(a,9);

print(a,9);

break;

case 2:

insertionsort(a,9);

print(a,9);

break;

case 3:

quicksort(a,0,8);

print(a,9);

break;

case 4:

mergesort(a,0,8);

print(a,9);

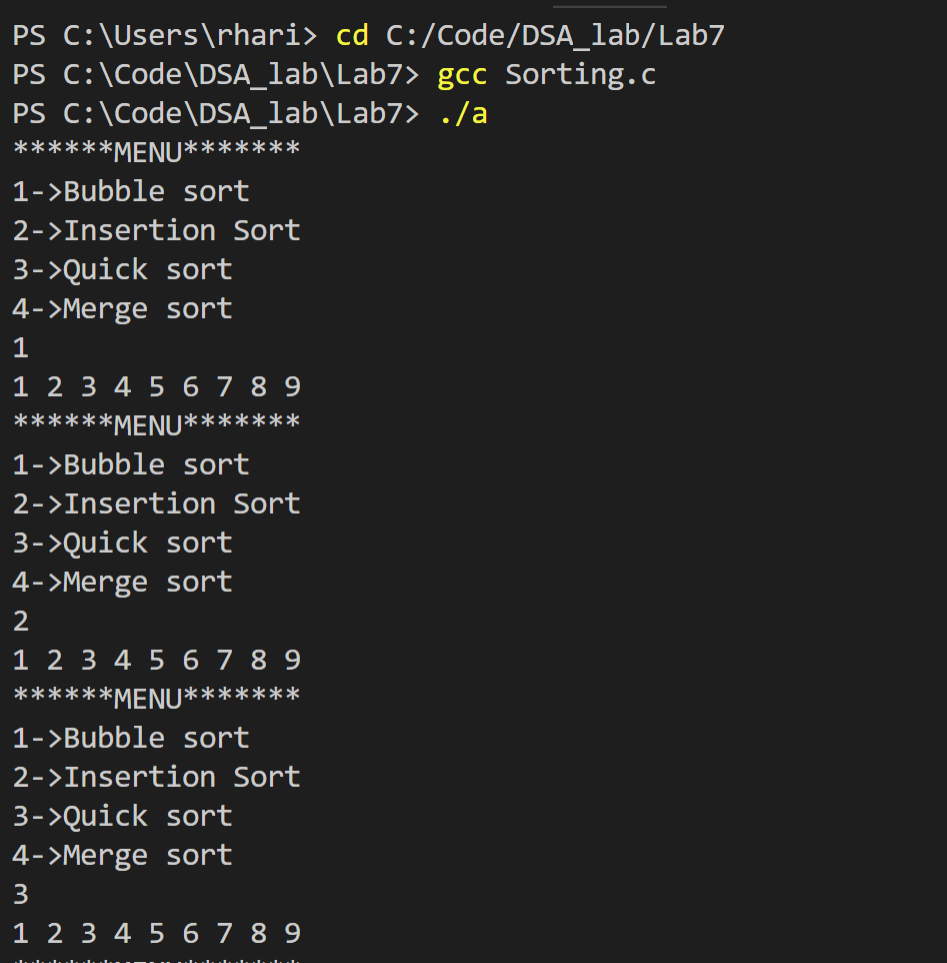
break;

}

}

return 0;

}



**Lab 7: Graph traversals**

**Prims Algorithm:**

#include<stdio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]= {0}

,min,mincost=0,cost[10][10];

int main() {

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

scanf("%d",&n);

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

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

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

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

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

cost[i][j]=999;

}

visited[1]=1;

printf("\n");

while(ne<n) {

for (i=1,min=999;i<=n;i++)

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

if(cost[i][j]<min)

if(visited[i]!=0) {

min=cost[i][j];

a=u=i;

b=v=j;

}

if(visited[u]==0 || visited[v]==0) {

printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);

mincost+=min;

visited[b]=1;

}

cost[a][b]=cost[b][a]=999;

}

printf("\n Minimun cost=%d",mincost);

}

**DFS:**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

#define initial 1

#define waiting 2

#define visited 3

int adj[100][100];

int n, state[100];

int stack[100], top=-1;

void push(int vertex){

if (top==MAX-1){

printf("Stack is full\n");

}

else{

top++;

stack[top]=vertex;

}

}

int is\_empty(){

if (top==-1)

return 1;

else

return 0;

}

int pop(){

int d;

if (top==-1){

printf("Stack is empty\n");

exit(1);

}

d=stack[top];

top--;

return d;

}

void create\_graph(){

int count, max\_edge, origin,dest;

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

scanf("%d",&n);

max\_edge=n\*(n-1);

for (count=1;count<max\_edge;count++){

printf("Enter edge %d(-1 -1 to quit):", count);

scanf("%d %d",&origin, &dest);

if ((origin==-1)&&(dest==-1))break;

if (origin>=n||dest>=n||origin<0||dest<0){

printf("Invalid edge!\n");

count--;

}

else{

adj[origin][dest]=1;

}

}

}

void DFS(int v){

int i;

push(v);

state[v]=waiting;

while(!is\_empty()){

v=pop();

printf("%d",v);

state[v]=visited;

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

if (adj[v][i]==1 && state[i]==initial){

push(i);

state[i]=waiting;

}

}

}

printf("\n");

}

void DF\_Traversal()

{

int v;

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

state[v] = initial;

printf("Enter Start Vertex for DFS: \n");

scanf("%d", &v);

DFS(v);

}

int main()

{

create\_graph();

DF\_Traversal();

return 0;

}

**BFS:**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

#define initial 1

#define waiting 2

#define visited 3

int adj[100][100];

int n, state[100];

int queue[100], front=-1, rear=-1;

void enqueue(int vertex){

if (rear==MAX-1){

printf("Queue is full\n");

}

else{

if (front==-1){

front=0;}

rear++;

queue[rear]=vertex;

}

}

int is\_empty(){

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

return 1;

else

return 0;

}

int dequeue(){

int delete\_item;

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

printf("Queue is empty\n");

exit(1);

}

delete\_item=queue[front];

front=front+1;

return delete\_item;

}

void create\_graph(){

int count, max\_edge, origin,dest;

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

scanf("%d",&n);

max\_edge=n\*(n-1);

for (count=1;count<max\_edge;count++){

printf("Enter edge %d(-1 -1 to quit):", count);

scanf("%d %d",&origin, &dest);

if ((origin==-1)&&(dest==-1))break;

if (origin>=n||dest>=n||origin<0||dest<0){

printf("Invalid edge!\n");

count--;

}

else{

adj[origin][dest]=1;

}

}

}

void BFS(int v){

int i;

enqueue(v);

state[v]=waiting;

while(!is\_empty()){

v=dequeue();

printf("%d",v);

state[v]=visited;

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

if (adj[v][i]==1 && state[i]==initial){

enqueue(i);

state[i]=waiting;

}

}

}

printf("\n");

}

void BF\_Traversal()

{

int v;

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

state[v] = initial;

printf("Enter Start Vertex for BFS: \n");

scanf("%d", &v);

BFS(v);

}

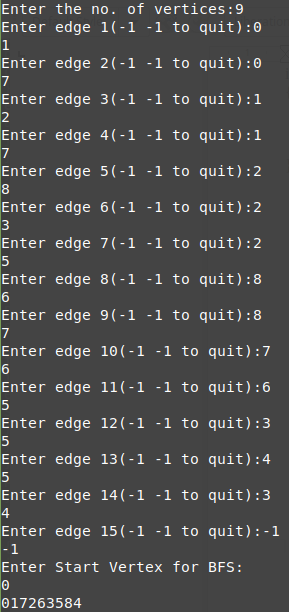
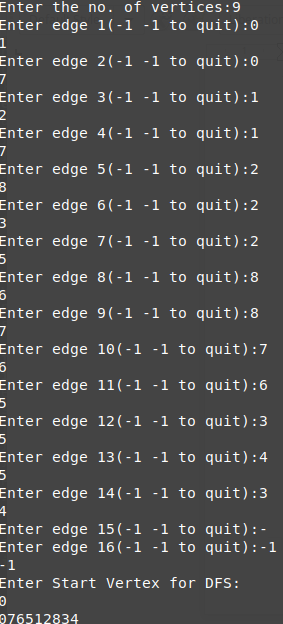
int main()

{

create\_graph();

BF\_Traversal();

return 0;

}

**Lab 8: Dijikstra’s Algorithm**

**Code:**

#include <limits.h>

#include <stdio.h>

#define V 6

int minDistance(int dist[], int sptSet[])

{

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (sptSet[v] == 0 && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

int printSolution(int dist[])

{

printf("Vertex \t\t Distance from Source\n");

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

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

}

void dijkstra(int graph[V][V], int src)

{

int dist[V];

int sptSet[V];

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

dist[i] = INT\_MAX, sptSet[i] = 0;

dist[src] = 0;

for (int count = 0; count < V - 1; count++) {

int u = minDistance(dist, sptSet);

sptSet[u] = 1;

for (int v = 0; v < V; v++)

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX

&& dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

printSolution(dist);

}

int main()

{

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

int a;

scanf("%d",&a);

int graph[V][V];

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

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

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

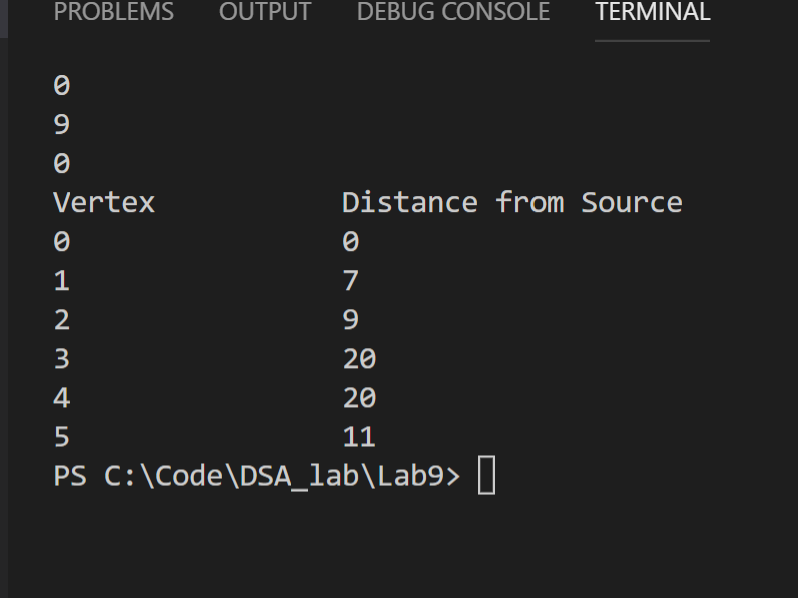
}

}

dijkstra(graph, 0);

return 0;

}



**Lab 9:Knapsack Problem**

**Code:**

#include<stdio.h>

int max(int a, int b) { return (a > b)? a : b; }

int knapSack(int W, int wt[], int val[], int n)

{

if (n == 0 || W == 0)

return 0;

if (wt[n-1] > W)

return knapSack(W, wt, val, n-1);

else return max( val[n-1] + knapSack(W-wt[n-1], wt, val, n-1),

knapSack(W, wt, val, n-1)

);

}

int main()

{

int m;

printf("Enter the no of items:");

scanf("%d",&m);

int val[m], wt[m];

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

printf("Enter the value:");

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

printf("Enter the weight:");

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

}

int W;

printf("Enter the max wt:");

scanf("%d",&W);

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

printf("%d", knapSack(W, wt, val, n));

return 0;

}

**Output:**

