/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO PERFORM LINEAR SEARCH

\*/

#include <stdio.h>

int main()

{

int n,i,item,flag,c;

printf("Enter length of array\n");

scanf("%d",&n);

int a[n];

printf("Enter elements of array\n");

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

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

printf("Enter item to be searched\n");

scanf("%d",&item);

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

{

if(item==a[i])

{

flag=1;

c=i;

break;

}

}

if(flag==1)

printf("Item exists at index %d\n",c);

else

printf("Item does not exist in array\n");

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO PERFORM BINARY SEARCH

\*/

#include <stdio.h>

int main()

{

int n,i,item,flag,c;

printf("Enter length of array\n");

scanf("%d",&n);

int a[n];

printf("Enter elements of sorted array\n");

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

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

printf("Enter item to be searched\n");

scanf("%d",&item);

int f=0,l=n,m=n/2;

while(m<=l&&m>=f)

{

if(a[m]==item)

{

flag=1;

c=m;

break;

}

else if(a[m]<item)

f=m;

else

l=m;

m=(f+l)/2;

}

if(flag==1)

printf("Item exists at index %d\n",c);

else

printf("Item does not exist in array\n");

}

OUTPUT :

A screenshot of a computer

Description automatically generated with medium confidence

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO PERFORM POLYNOMIAL ADDITION

\*/

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

typedef struct{int cf;int xp;}poly;

static poly terms[MAX];

static int avail=0;

int comp(int a,int b)

{ if(a>b)

return 1;

else if(b>a)

return -1;

else

return 0;

}

void attach(int cof,int exp)

{ if(avail>=MAX)

{

fprintf(stderr,"Too many terms");

exit(0);

}

terms[avail].cf=cof;

terms[avail].xp=exp;

avail++;

}

void padd(int startA,int finishA,int startB,int finishB)

{ int coeff;

int startR=avail;

while(startA<=finishA&&startB<=finishB)

{

switch(comp(terms[startA].xp,terms[startB].xp))

{

case-1:attach(terms[startB].cf,terms[startB].xp);

startB++;

break;

case 1:attach(terms[startA].cf,terms[startA].xp);

startA++;

break;

case 0:coeff=terms[startA].cf+terms[startB].cf;

if(coeff!=0)

{

attach(coeff,terms[startA].xp);

startA++;

startB++;

}

break;

}

}

for(;startA<=finishA;startA++)

attach(terms[startA].cf,terms[startA].xp);

for(;startB<=finishB;startB++)

attach(terms[startB].cf,terms[startB].xp);

int finishR=avail-1;

for(;startR<=finishR;startR++)

printf("%dx^%d+",terms[startR].cf,terms[startR].xp);

printf("\nIs sum of\n");

}

int main()

{

int n1,n2,i,j;

int as=avail;

printf("Enter number of terms of 1st polynomial\n");

scanf("%d",&n1);

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

{

printf("Enter coefficient and exponent\n");

scanf("%d",&terms[avail].cf);

scanf("%d",&terms[avail].xp);

avail++;

}

int af=avail-1;int bs=avail;

printf("Enter number of terms of 2nd polynomial\n");

scanf("%d",&n2);

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

{

printf("Enter coefficient and exponent\n");

scanf("%d",&terms[avail].cf);

scanf("%d",&terms[avail].xp);

avail++;

}

int bf=avail-1;

padd(as,af,bs,bf);

for(;as<=af;as++)

printf("%dx^%d+",terms[as].cf,terms[as].xp);

printf("\n");

for(;bs<=bf;bs++)

printf("%dx^%d+",terms[bs].cf,terms[bs].xp);

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO IMPLEMENT SPARSE MATRIX

\*/

#include<stdio.h>

int A[50][3], B[50][3], sum[50][3], AT[50][3], BT[50][3];

void matrixtotuple(int m[50][50], int r, int c, int t[50][3]) {

t[0][0] = r;

t[0][1] = c;

t[0][2] = 0;

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

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

if(m[i][j] != 0) {

t[0][2]++;

t[t[0][2]][0] = i;

t[t[0][2]][1] = j;

t[t[0][2]][2] = m[i][j];

}

}

}

}

void displaytupleform(int m[50][3]) {

for(int i=0; i<=m[0][2]; i++)

printf("%d %d %d \n", m[i][0], m[i][1], m[i][2]);

}

void transpose(int m[50][3], int t[50][3]) {

t[0][0] = m[0][1];

t[0][1] = m[0][0];

t[0][2] = 0;

for(int i=0; i<m[0][1]; i++) {

for(int j=1; j<=m[0][2]; j++) {

if(m[j][1]==i) {

t[0][2]++;

t[t[0][2]][0] = m[j][1];

t[t[0][2]][1] = m[j][0];

t[t[0][2]][2] = m[j][2];

}

}

}

}

void add() {

sum[0][0] = A[0][0];

sum[0][1] = A[0][1];

sum[0][2] = 0;

int a = 1;

int b = 1;

while( (a<=A[0][2]) && (b<=B[0][2]) ) {

if( (A[a][0]==B[b][0]) && (A[a][1]==B[b][1]) ) {

if(A[a][2]+B[b][2] != 0) {

sum[0][2]++;

sum[sum[0][2]][0] = A[a][0];

sum[sum[0][2]][1] = A[a][1];

sum[sum[0][2]][2] = A[a][2]+B[b][2];

}

a++;

b++;

}

else {

if( A[a][0] < B[b][0] ) {

sum[0][2]++;

sum[sum[0][2]][0] = A[a][0];

sum[sum[0][2]][1] = A[a][1];

sum[sum[0][2]][2] = A[a][2];

a++;

}

else if( B[b][0] < A[a][0] ) {

sum[0][2]++;

sum[sum[0][2]][0] = B[b][0];

sum[sum[0][2]][1] = B[b][1];

sum[sum[0][2]][2] = B[b][2];

b++;

}

else {

if( A[a][1] < B[b][1] ) {

sum[0][2]++;

sum[sum[0][2]][0] = A[a][0];

sum[sum[0][2]][1] = A[a][1];

sum[sum[0][2]][2] = A[a][2];

a++;

}

else {

sum[0][2]++;

sum[sum[0][2]][0] = B[b][0];

sum[sum[0][2]][1] = B[b][1];

sum[sum[0][2]][2] = B[b][2];

b++;

}

}

}

}

while ( a<=A[0][2] ) {

sum[0][2]++;

sum[sum[0][2]][0] = A[a][0];

sum[sum[0][2]][1] = A[a][1];

sum[sum[0][2]][2] = A[a][2];

a++;

}

while ( b<=B[0][2] ) {

sum[0][2]++;

sum[sum[0][2]][0] = B[b][0];

sum[sum[0][2]][1] = B[b][1];

sum[sum[0][2]][2] = B[b][2];

b++;

}

}

void main() {

int a[50][50], b[50][50], ar, ac, br, bc;

printf("number of rows in A : ");

scanf("%d", &ar);

printf("number of columns in A : ");

scanf("%d", &ac);

printf("enter matrix A \n");

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

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

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

printf("number of rows in B : ");

scanf("%d", &br);

printf("number of columns in B : ");

scanf("%d", &bc);

printf("enter matrix B :\n");

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

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

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

matrixtotuple(a, ar, ac, A);

matrixtotuple(b, br, bc, B);

printf("matrix A in tuple form : \n");

displaytupleform(A);

printf("matrix B in tuple form : \n");

displaytupleform(B);

transpose(A, AT);

transpose(B, BT);

printf("A transpose : \n");

displaytupleform(AT);

printf("B transpose : \n");

displaytupleform(BT);

if( (A[0][0]==B[0][0]) && (A[0][1]==B[0][1]) ) {

printf("\nA + B = \n");

add();

displaytupleform(sum);

}

else

printf("matrices cannot be added \n");

}

OUTPUT:

A screenshot of a computer screen

Description automatically generated with medium confidence

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO IMPLEMENT STACK

\*/

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

void main() {

int stack[MAX\_SIZE];

int top = -1,i, proceed;

proceed = 1;

int option,item;

do {

printf("1.push\n2.pop\n3.view\n4.exit\n");

scanf("%d", &option);

switch(option)

{

case 1: if(top==MAX\_SIZE-1)

printf("Stack overflow \n");

else

{

printf("Enter the element to be added");

scanf("%d",&item);

stack[++top]=item;

}

break;

case 2: if(top==-1)

printf("Stack underflow");

else

printf("Deleting element = %d \n",stack[top--]);

break;

case 3: printf("Elements are \n");

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

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

break;

case 4: exit(0);

break;

default:printf("invalid choice");

}

} while(proceed = 1);

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO IMPLEMENT QUEUE

\*/

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 5

int q[MAX\_SIZE];

int front = -1;

int rear = -1;

void display() {

printf("Queue : \n");

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

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

printf("\n");

}

void delete() {

if(front == rear)

printf("U N D E R F L O W ...!\n");

else {

front++;

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

}

display();

}

void add() {

printf("enter element to add : ");

int item;

scanf("%d", &item);

if(rear == MAX\_SIZE-1)

printf("O V E R F L O W ...!\n");

else {

rear++;

q[rear]=item;

}

display();

}

void main() {

int choice;

repeat:

printf("\n1.Add\n2.Delete\n3.Exit\nenter choice : ");

scanf("%d", &choice);

switch(choice) {

case 1: add();

break;

case 2: delete();

break;

case 3: exit(0);

break;

default: printf("wrong choice\n");

}

goto repeat;

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO IMPLEMENT CIRCULAR QUEUE

\*/

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 5

int q[MAX\_SIZE];

int front = -1;

int rear = -1;

void display() {

printf("Queue : \n");

if(front > rear) {

for(int i=front; i<MAX\_SIZE; i++)

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

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

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

}

else { // add condition if front = rear = -1

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

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

}

printf("\n");

}

void add() {

int item;

printf("enter element to add : ");

scanf("%d", &item);

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

front = rear = 0;

q[rear] = item;

}

else if( (rear+1)%MAX\_SIZE == front) {

printf("overflow..!!\n");

}

else {

rear = (rear+1)%MAX\_SIZE;

q[rear] = item;

}

display();

}

void delete() {

if(front == -1)

printf("underflow..!!\n");

else if(front == rear) {

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

front=rear=-1;

}

else {

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

front = (front+1)%MAX\_SIZE;

}

display();

}

void main() {

int choice;

repeat:

printf("\n1.Add\n2.Delete\n3.Exit\nenter choice : ");

scanf("%d", &choice);

switch(choice) {

case 1: add();

break;

case 2: delete();

break;

case 3: exit(0);

break;

default: printf("invalid choice \n");

}

goto repeat;

}

OUTPUT :

A screenshot of a computer

Description automatically generated with medium confidence

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO IMPLEMENT PRIORITY QUEUE

\*/

#include <stdio.h>

#include <stdlib.h>

#define maxsize 100

int pq[maxsize][2];

int size = -1;

void display() {

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

printf(" %d(%d) ", pq[i][0], pq[i][1]);

printf("\n");

}

void enqueue() {

if(size == maxsize-1)

printf("\nOverflow\n");

else {

int data, prio;

printf("\nData : ");

scanf("%d", &data);

printf("Priority : ");

scanf("%d", &prio);

size++;

pq[size][0] = data;

pq[size][1] = prio;

}

}

void dequeue() {

if(size == -1)

printf("\nUnderflow\n");

else {

int pos = 0;

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

if(pq[i][1] < pq[pos][1])

pos = i;

for(int i=pos; i<size; i++) {

pq[i][0] = pq[i+1][0];

pq[i][1] = pq[i+1][1];

}

size--;

}

display();

}

void main() {

int choice;

printf("1.Add\n2.Delete\n3.Display\n4.Exit\n");

repeat:

printf("\nEnter Choice : ");

scanf("%d", &choice);

switch(choice) {

case 1: enqueue();

break;

case 2: dequeue();

break;

case 3: display();

break;

case 4: exit(0);

break;

default : printf("Wrong Input\n");

}

goto repeat;

}

OUTPUT:

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO IMPLEMENT DEQUEUE

\*/

#include <stdio.h>

#include <stdlib.h>

#define size 10

int dq[size];

int end = -1;

void display() {

printf("\n");

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

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

printf("\n");

}

void insert(int where) {

if( end == size-1)

printf("OVERFLOW..!!\n");

else {

printf("enter data : ");

int data;

scanf("%d", &data);

end++;

if(where == 1) {

dq[end]=data;

}

else {

for(int i=end; i>0; i--)

dq[i] = dq[i-1];

dq[0]=data;

}

}

display();

}

void delete(int where) {

if(end == -1)

printf("UNDERFLOW..!!");

else {

if(where == 0)

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

dq[i] = dq[i+1];

end--;

}

display();

}

void main() {

int choice;

repeat:

printf("\n1.Insert at front\n2.Insert at back\n3.Delete from front\n4.Delete from back\n5.Exit\nenter choice: ");

scanf("%d", &choice);

switch(choice) {

case 1: insert(0);

break;

case 2: insert(1);

break;

case 3: delete(0);

break;

case 4: delete(1);

break;

case 5: exit(0);

default: printf("\ninvalid choice\n");

}

goto repeat;

}

OUTPUT :

A picture containing text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO IMPLEMENT SINGLY LINKED LIST

\*/

#include <stdio.h>

#include <stdlib.h>

struct Node

{

int data;

struct Node \*link;

};

struct Node \*head = NULL;

void display(){

struct Node \*ptr = head;

if (ptr == NULL){

printf("\nList empty");

return;

}

while (ptr != NULL){

printf("%d\t", ptr->data);

ptr = ptr->link;

}

printf("\n");

}

void insertAtBeginning(int data){

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

newNode->data = data;

newNode->link = head;

head = newNode;

}

void insertAtEnd(int data){

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

newNode->data = data;

newNode->link = NULL;

struct Node \*end = head;

if (end != NULL){

while (end->link != NULL){

end = end->link;

}

end->link = newNode;

}

else

head = newNode;

}

void insertAfterPosition(int position, int data){

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

newNode->data = data;

newNode->link = NULL;

struct Node \*curr = head;

while (curr != NULL && --position)

curr = curr->link;

if (curr == NULL){

printf("\nPosition exceeded list size");

return;

}

newNode->link = curr->link;

curr->link = newNode;

}

void deleteElement(int element){

struct Node \*curr = head, \*prev = NULL;

while (curr != NULL && curr->data != element){

prev = curr;

curr = curr->link;

}

if (curr == NULL){

printf("Element not found!\n");

return;

}

if (prev == NULL)

head = head->link;

else

prev->link = curr->link;

free(curr);

}

int main()

{

int ch, element, pos;

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

printf("\n1.Insert at beginning");

printf("\n2.Insert at end");

printf("\n3.Insert at position");

printf("\n4.Delete node containing a particular element ");

printf("\n5.Display linked list");

printf("\n\n");

do

{

printf("Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter element to insert : ");

scanf("%d", &element);

insertAtBeginning(element);

break;

case 2:

printf("Enter element to insert : ");

scanf("%d", &element);

insertAtEnd(element);

break;

case 3:

printf("Enter the position after which node will be inserted : ");

scanf("%d", &pos);

printf("Enter element to insert : ");

scanf("%d", &element);

insertAfterPosition(pos, element);

break;

case 4:

printf("Enter element to delete : ");

scanf("%d", &element);

deleteElement(element);

break;

case 5:

display();

break;

}

} while (ch < 6);

return 0;

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO REVERSE LINKED LIST

\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct node {

int data;

struct node \*link;

}listnode;

listnode \*front = NULL;

listnode \*create(int value){

listnode \*node;

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

node->data = value;

node->link = NULL;

return node;

}

void display(){

listnode \*temp = front;

if(temp==NULL){

printf("\nThe List is Empty !!!\n\n");

}

while(temp!=NULL){

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

temp = temp->link;

}

printf("\n");

}

void insert\_begin(int value){

listnode \*node = create(value);

if(node!=NULL){

if(front==NULL){

front = node;

}

else{

node->link = front;

front = node;

}

}

else{

printf("\nOVERFLOW!!!\n");

}

display();

}

void insert\_end(int value){

listnode \*node = create(value);

listnode \*temp = front;

while(temp->link!=NULL){

temp = temp->link;

}

if(node!=NULL){

temp->link = node;

}

else{

printf("\nOVERFLOW!!!\n");

}

display();

}

void reverse(){

int count = 1,i,ct;

listnode \*temp = front;

listnode \*ptr;

while(temp->link!=NULL){

count++;

temp=temp->link;

}

ptr = temp;

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

temp = front->link;

front->link = ptr->link;

ptr->link = front;

front = temp;

}

front = ptr;

printf("\nThe REVERSED List is : \n");

display();

}

void main(){

int choice,value,check;

repeat:

printf("\n1.Insert at beginning\n2.Insert at end\n3.Reverse list\n4.Exit\nenter your choice : ");

scanf("%d",&choice);

switch(choice){

case 1 : printf("\nEnter the value to be inserted : ");

scanf("%d",&value);

insert\_begin(value);

break;

case 2 : printf("\nEnter the value to be inserted : ");

scanf("%d",&value);

insert\_end(value);

break;

case 3 : reverse();

break;

case 4 : exit(0);

break;

default: printf("\nEnter a valid choice !!\n");

}

goto repeat;

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO IMPLEMENT DOUBLY LINKED LIST

\*/

#include<stdio.h>

#include<stdlib.h>

struct Node {

struct Node\* prev;

int data;

struct Node\* next;

};

struct Node\* beg = NULL;

struct Node\* end = NULL;

void displayList() {

printf("\n");

struct Node\* temp = beg;

while(temp != NULL) {

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

temp = temp->next;

}

printf("\n");

}

void insertElement(int pos) {

int data;

printf("\nenter data : ");

scanf("%d", &data);

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

n->data = data;

n->prev = NULL;

n->next = NULL;

if(pos == 0) {

printf("enter position to insert data : ");

scanf("%d", &pos);

}

if((beg == NULL) && (end == NULL))

beg = end = n;

else if(pos == 1) {

beg->prev = n;

n->next = beg;

beg = n;

}

else {

struct Node\* temp = beg;

int count = 0;

while(temp != NULL) {

count++;

if(count == pos)

break;

temp = temp->next;

}

if(temp != NULL) {

n->prev = temp->prev;

temp->prev->next = n;

n->next = temp;

temp->prev = n;

}

else {

end->next = n;

n->prev = end;

end = n;

}

}

/\*

if((beg == NULL) && (end == NULL))

beg = end = n;

else if(pos == 0) {

beg->prev = n;

n->next = beg;

beg = n;

}

else if(pos == 1) {

end->next = n;

n->prev = end;

end = n;

}

else {

printf("enter position to insert data : ");

int p;

scanf("%d", &p);

}

\*/

displayList();

}

void deleteElement() {

if( (beg == NULL) && (end == NULL) ) {

printf("underflow...!\n");

return;

}

int ele;

printf("\nenter element to delete : ");

scanf("%d", &ele);

struct Node\* temp = beg;

while(temp != NULL) {

if(temp->data == ele) {

if((temp == beg) && (temp == end))

beg = end = NULL;

else if(temp == beg) {

beg = beg->next;

beg->prev = NULL;

}

else if(temp == end) {

end = end->prev;

end->next = NULL;

}

else {

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

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

}

free(temp);

break;

}

temp = temp->next;

}

displayList();

}

void main() {

int choice;

repeat:

printf("\n1.Insert at beginning\n2.Insert at end\n3.Insert at a position\n4.Delete a node\n5.Exit\nenter your choice : ");

scanf("%d", &choice);

switch(choice) {

case 1: insertElement(1);

break;

case 2: insertElement(-1);

break;

case 3: insertElement(0);

break;

case 4: deleteElement();

break;

case 5: exit(0);

break;

default: printf("wrong choice...!\n");

}

goto repeat;

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO IMPLEMENT CIRCULAR LINKED LIST

\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

int data;

struct node \*next;

}listNode;

listNode \*front;

void display(){

listNode \*ptr;

ptr = front;

if(front == NULL){

printf("EMPTY!!");

}

else{

printf("The List : ");

while(ptr->next != front){

printf(" %d ", ptr -> data);

ptr = ptr->next;

}

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

}

}

void insert\_begin(){

listNode \*ptr,\*temp;

int item;

ptr = (listNode\*)malloc(sizeof(listNode));

if(ptr == NULL)

{

printf("OVERFLOW!!!");

}

else

{

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

scanf("%d",&item);

ptr -> data = item;

if(front == NULL){

front = ptr;

ptr->next = front;

}

else{

temp = front;

while(temp->next != front)

temp = temp->next;

ptr->next = front;

temp -> next = ptr;

front = ptr;

}

}

}

void insert\_last(){

listNode \*ptr,\*temp;

int item;

ptr = (listNode \*)malloc(sizeof(listNode));

if(ptr == NULL){

printf("OVERFLOW");

}

else{

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

scanf("%d",&item);

ptr->data = item;

if(front == NULL){

front = ptr;

ptr->next = front;

}

else{

temp = front;

while(temp -> next != front){

temp = temp -> next;

}

temp -> next = ptr;

ptr->next = front;

}

}

}

void delete\_begin(){

listNode \*ptr;

if(front == NULL){

printf("UNDERFLOW!!!");

}

else if(front->next == front){

front = NULL;

free(front);

}

else{

ptr = front;

while(ptr->next != front)

ptr = ptr->next;

ptr->next = front->next;

free(front);

front = ptr->next;

}

display();

}

void delete\_last(){

listNode \*ptr, \*temp;

if(front == NULL){

printf("UNDERFLOW!!!");

}

else if(front->next == front){

front = NULL;

free(front);

}

else{

ptr = front;

while(ptr->next != front){

temp=ptr;

ptr = ptr->next;

}

temp->next = ptr->next;

free(ptr);

}

display();

}

void main(){

int choice =0;

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

printf("\n1.Insert at Begining\n2.Insert at End");

printf("\n3.Delete from Beginning\n4.Delete from last\n5.Display\n6.Exit\n\n");

while(choice!=6){

printf("\nEnter the Menu choice : ");

scanf("%d",&choice);

switch(choice){

case 1 : insert\_begin();

break;

case 2 : insert\_last();

break;

case 3 : delete\_begin();

break;

case 4 : delete\_last();

break;

case 5 : display();

break;

case 6 : return;

break;

default: printf("\nEnter a valid choice !!\n");

}

}

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO PERFORM POLYNOMIAL ADDITION USING LINKED LIST

\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct Node {

int coeff;

int pow;

struct Node\* link;

} expression;

expression\* heada = NULL;

expression\* headb = NULL;

expression\* headresult = NULL;

void displayExpression(expression\* exp) {

expression\* temp = exp;

printf("\n");

while(temp!=NULL) {

if(temp == exp)

printf("%d(x^%d)", temp->coeff, temp->pow);

else

printf("%+d(x^%d)", temp->coeff, temp->pow);

temp = temp->link;

}

printf("\n");

}

void sortExpression(expression\* exp) {

expression\* i = exp;

int swapped = 0;

if(exp == NULL)

return;

do {

i = exp;

swapped = 0;

while(i->link != NULL) {

if(i->pow < i->link->pow) {

int temppow = i->pow;

int tempcoeff = i->coeff;

i->pow = i->link->pow;

i->coeff = i->link->coeff;

i->link->pow = temppow;

i->link->coeff = tempcoeff;

swapped = 1;

}

i = i->link;

}

}while(swapped);

}

expression\* inputExpression() {

int num;

expression\* temp;

expression\* first = NULL;

printf("enter number of terms : ");

scanf("%d", &num);

printf("\n");

for(int i=1; i<=num; i++) {

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

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

printf("coefficient : ");

scanf("%d", &node->coeff);

printf("power : ");

scanf("%d", &node->pow);

node->link = NULL;

if(first == NULL) {

temp = first = node;

}

else {

temp->link = node;

temp = node;

}

}

sortExpression(first);

displayExpression(first);

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

return first;

}

void findSum(expression\* expa, expression\* expb) {

expression\* temp;

expression\* node;

while((expa != NULL) && (expb != NULL)) {

if(expa->pow == expb->pow) {

int sum = expa->coeff + expb->coeff;

if(sum) {

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

node->coeff = sum;

node->pow = expa->pow;

node->link = NULL;

if(headresult==NULL)

headresult = temp = node;

else {

temp->link = node;

temp = node;

}

}

expa = expa->link;

expb = expb->link;

}

else if(expa->pow > expb->pow) {

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

node->coeff = expa->coeff;

node->pow = expa->pow;

node->link = NULL;

if(headresult==NULL)

headresult = temp = node;

else {

temp->link = node;

temp = node;

}

expa = expa->link;

}

else {

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

node->coeff = expb->coeff;

node->pow = expb->pow;

node->link = NULL;

if(headresult==NULL)

headresult = temp = node;

else {

temp->link = node;

temp = node;

}

expb = expb->link;

}

}

while(expa != NULL) {

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

node->coeff = expa->coeff;

node->pow = expa->pow;

node->link = NULL;

if(headresult==NULL)

headresult = temp = node;

else {

temp->link = node;

temp = node;

}

expa = expa->link;

}

while(expb != NULL) {

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

node->coeff = expb->coeff;

node->pow = expb->pow;

node->link = NULL;

if(headresult==NULL)

headresult = temp = node;

else {

temp->link = node;

temp = node;

}

expb = expb->link;

}

}

void main() {

printf("\nexpression 1 :\n");

heada = inputExpression();

printf("\nexpression 2 :\n");

headb = inputExpression();

findSum(heada, headb);

printf("\nSum of both expressions : \n");

displayExpression(headresult);

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO PERFORM POLYNOMIAL MULTIPLICATION USING LINKED LIST

\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct Node {

int coeff;

int pow;

struct Node\* link;

} expression;

expression\* heada = NULL;

expression\* headb = NULL;

expression\* headresult = NULL;

void displayExpression(expression\* exp) {

expression\* temp = exp;

printf("\n");

while(temp!=NULL) {

if(temp == exp)

printf("%d(x^%d)", temp->coeff, temp->pow);

else

printf("%+d(x^%d)", temp->coeff, temp->pow);

temp = temp->link;

}

printf("\n");

}

void sortExpression(expression\* exp) {

expression\* i = exp;

int swapped = 0;

if(exp == NULL)

return;

do {

i = exp;

swapped = 0;

while(i->link != NULL) {

if(i->pow < i->link->pow) {

int temppow = i->pow;

int tempcoeff = i->coeff;

i->pow = i->link->pow;

i->coeff = i->link->coeff;

i->link->pow = temppow;

i->link->coeff = tempcoeff;

swapped = 1;

}

i = i->link;

}

}while(swapped);

}

expression\* inputExpression() {

int num;

expression\* temp;

expression\* first = NULL;

printf("enter number of terms : ");

scanf("%d", &num);

printf("\n");

for(int i=1; i<=num; i++) {

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

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

printf("coefficient : ");

scanf("%d", &node->coeff);

printf("power : ");

scanf("%d", &node->pow);

node->link = NULL;

if(first == NULL) {

temp = first = node;

}

else {

temp->link = node;

temp = node;

}

}

sortExpression(first);

displayExpression(first);

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

return first;

}

void filterExpression() {

expression\* term = headresult;

expression\* prev = headresult;

while((term != NULL) && (term->link != NULL)) {

if(term->pow == term->link->pow) {

int sum = term->coeff + term->link->coeff;

if(sum) {

term->coeff = sum;

term->link = term->link->link;

}

else {

if(term == prev) {

headresult = term->link->link;

term = prev = headresult;

}

else {

prev->link = term->link->link;

term = prev->link;

}

}

continue;

}

prev = term;

term = term->link;

}

}

void findProduct(expression\* expa, expression\* expb) {

expression\* temp;

expression\* a = expa;

expression\* b = expb;

while(a != NULL) {

b = expb;

while(b != NULL) {

int c = a->coeff \* b->coeff;

int p = a->pow + b->pow;

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

node->coeff = c;

node->pow = p;

node->link = NULL;

if(headresult == NULL)

headresult = temp = node;

else {

temp->link = node;

temp = node;

}

b = b->link;

}

a = a->link;

}

sortExpression(headresult);

filterExpression();

}

void main() {

printf("\nexpression 1 :\n");

heada = inputExpression();

printf("\nexpression 2 :\n");

headb = inputExpression();

findProduct(heada, headb);

printf("\nProduct of the two experssions is : \n");

displayExpression(headresult);

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO IMPLEMENT STACK USING LINKED LIST

\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct Node {

int data;

struct Node\* link;

} node;

node\* top = NULL;

void displayStack() {

node\* temp = top;

if (temp == NULL) {

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

return;

}

printf("\nStack : \n");

while(temp != NULL) {

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

temp = temp->link;

}

}

void push() {

int data;

printf("\nenter data : ");

scanf("%d", &data);

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

n->data = data;

n->link = NULL;

if(top == NULL)

top = n;

else {

n->link = top;

top = n;

}

displayStack();

}

void pop() {

if(top == NULL) {

printf("\nstack empty\n");

return;

}

else {

printf("\npop element : %d\n", top->data);

top = top->link;

}

displayStack();

}

void main() {

int choice;

repeat:

printf("\n1.Push\n2.Pop\n3.Exit\nenter your choice : ");

scanf("%d", &choice);

switch (choice) {

case 1: push();

break;

case 2: pop();

break;

case 3: exit(0);

break;

default: printf("invalid choice...!!\n");

}

goto repeat;

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO IMPLEMENT QUEUE USING LINKED LIST

\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct node {

int data;

struct node\* link;

} Node;

Node\* front = NULL;

Node\* rear = NULL;

void display() {

Node\* temp = front;

printf("\n");

while(temp != NULL) {

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

temp = temp->link;

}

printf("\n");

}

void add() {

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

printf("\nenter data : ");

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

n->link = NULL;

if(front == NULL)

front = rear = n;

else {

rear->link = n;

rear = n;

}

display();

}

void delete() {

if(front == NULL)

printf("\nunderflow...!");

else if(front == rear)

front = rear = NULL;

else

front = front->link;

display();

}

void main() {

int choice;

repeat:

printf("\n1.Add element\n2.Delete element\n3.Exit\nenter your choice : ");

scanf("%d", &choice);

switch(choice) {

case 1: add();

break;

case 2: delete();

break;

case 3: exit(0);

break;

default: printf("invalid choice.......!\n");

}

goto repeat;

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO CHECK FOR PALINDROME STRING

\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct node {

int data;

struct node \*prev;

struct node \*next;

}listNode;

listNode \*beg = NULL;

listNode \*end = NULL;

void display(){

printf("\n");

listNode\* temp = beg;

while(temp != NULL) {

printf("%c ", temp->data);

temp = temp->next;

}

printf("\n");

if(beg==end){

printf("EMPTY!!");

}

}

void insertElement(char data) {

int pos = -1;

listNode\* n = (listNode\*)malloc(sizeof(listNode));

n->data = data;

n->prev = NULL;

n->next = NULL;

if((beg == NULL) && (end == NULL))

beg = end = n;

else if(pos == 1) {

beg->prev = n;

n->next = beg;

beg = n;

}

else {

listNode\* temp = beg;

int count = 0;

while(temp != NULL) {

count++;

if(count == pos)

break;

temp = temp->next;

}

if(temp != NULL) {

n->prev = temp->prev;

temp->prev->next = n;

n->next = temp;

temp->prev = n;

}

else {

end->next = n;

n->prev = end;

end = n;

}

}

}

int isPalindrome(listNode \*start) {

if (start == NULL)

return 1;

listNode \*end = start;

while (end->next != NULL)

end = end->next;

while (start != end) {

if (start->data != end->data)

return 0;

start = start->next;

end = end->prev;

}

return 1;

}

void main() {

char str[100];

printf("Enter the string : ");

scanf("%s",&str);

int i=0,len = 0;

for(len=0;str[len]!='\0';len++);

printf("\nThe Length : %d",len);

while(i<len){

insertElement(str[i]);

i++;

}

printf("\nThe Doubly list : \n");

display();

int check = isPalindrome(beg);

if(check == 1)

printf("\nString is A PALINDROME !!");

else if(check == 0)

printf("\nString is NOT A PALINDROME !!");

}

OUTPUT :

A screenshot of a computer screen

Description automatically generated with medium confidence

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO PERFORM SORTING ON A LINKED LIST

\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct node {

int data;

struct node\* link;

} mark;

mark\* head = NULL;

void display() {

mark\* temp = head;

printf("\n");

while(temp != NULL) {

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

temp = temp->link;

}

printf("\n");

}

void sort() {

mark\* temp = head;

int swapped = 0;

do {

temp = head;

swapped = 0;

while(temp->link != NULL) {

if(temp->data < temp->link->data) {

int t = temp->data;

temp->data = temp->link->data;

temp->link->data = t;

swapped = 1;

}

temp = temp->link;

}

}while(swapped);

}

void inputmarks(int n) {

mark\* temp;

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

printf("\nenter mark %d : ", i+1);

mark\* n = (mark\*)malloc(sizeof(mark));

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

n->link = NULL;

if(head == NULL)

head = temp = n;

else {

temp->link = n;

temp = n;

}

}

}

void main() {

int n;

printf("enter number of students : ");

scanf("%d", &n);

inputmarks(n);

printf("\nMarks sorted in descending order : \n");

sort();

display();

}

OUTPUT :

A screenshot of a computer screen

Description automatically generated with medium confidence

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO DELETE DUPLICATE ELEMENTS OF A LINKED LIST

\*/

#include<stdio.h>

#include<stdlib.h>

struct listnode

{

int data;

struct listnode \*link;

};

struct listnode \*head=NULL;

void Push(int value)

{

struct listnode \*node;

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

node->data=value;

node->link=NULL;

struct listnode \*temp=head;

if(head==NULL)

head=node;

else

{ while(temp->link!=NULL)

{ temp=temp->link;

}

temp->link=node;

}

}

void RemoveDuplicate()

{

struct listnode \*temp, \*ptr;

ptr = head;

while(ptr != NULL) {

temp = ptr;

while(temp->link != NULL) {

if(temp->link->data == ptr->data) {

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

continue;

}

temp = temp->link;

}

ptr = ptr->link;

}

}

void Display()

{ struct listnode \*temp;

temp=head;

printf("List:");

while(temp!=NULL)

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

temp=temp->link;

}

printf("\n");

}

void main()

{ int N,value;

printf("Enter number of numbers: ");

scanf("%d",&N);

printf("Enter data : \n");

while(N--)

{

scanf("%d",&value);

Push(value);

}

printf("Initial List: ");

Display();

RemoveDuplicate();

printf("Final List: ");

Display();

printf("\n");

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO CONVERT INFIX TO POSTFIX

\*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

/\*

operator from the stack should be popped if its' priority is >= the priority of the scanned character from the expression.

\*/

char stack[100];

int size = -1;

char postexp[100];

// function to display the final postfix expression

void displayPostfix() {

printf("\nPostfix Expression : \n%s\n", postexp);

printf("\n");

}

// function to add required specific characters into the postfix expression

void addToPostExp(char ch) {

postexp[strlen(postexp)] = ch;

//displayPostfix();

}

// function to pop an operator from the operators stack

char popFromStack() {

size--;

return stack[size+1];

}

// function to push operators to the stack

void pushToStack(char ch) {

size++;

stack[size] = ch;

}

// function that returns priority of different characters in an expression

int getPriority(char ch) {

if(ch == '^')

return 3;

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

return 2;

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

return 1;

else // all operands comes under this condition

return 0;

}

OUTPUT:

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO CONVERT INFIX TO PREFIX

\*/

#include<stdio.h>

#include<string.h>

/\*

The infix expression is to be reversed. Then converted to postfix expression

with a slight change in the algorithm. The so obtained postfix expression is

reversed to get the prefix expression.

\*/

/\*

while converting to postfix here, we have to pop all characters from the stack

which are higher than the scanned operator. But in case of '^', we pop operators

from stack which are greater or equal to the scanned operator.

\*/

char preexp[100];

char stack[100];

int size = -1;

void displayPrefixExp() {

printf("\nPrefix expression : \n%s\n", preexp);

printf("\n");

}

void addToPreExp(char ch) {

preexp[strlen(preexp)] = ch;

//displayPostfix();

}

void reverseExpression(char\* exp) {

char temp;

for(int i=0; i<(strlen(exp)/2); i++) {

temp = exp[i];

exp[i] = exp[strlen(exp)-1-i];

exp[strlen(exp)-1-i] = temp;

}

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

if(exp[i] == '(')

exp[i] = ')';

else if(exp[i] == ')')

exp[i] = '(';

}

}

char popFromStack() {

size--;

return stack[size+1];

}

void pushToStack(char ch) {

size++;

stack[size] = ch;

}

int getPriority(char ch) {

if(ch == '^')

return 3;

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

return 2;

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

return 1;

else // all operands comes under this condition

return 0;

}

void convertToPrefix(char\* exp) {

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

if(exp[i] == '(') // fist checks if it is opening bracket

pushToStack(exp[i]);

else if(exp[i] == ')') { // then checks if it's the closing bracket

char stackoperator;

while(1) {

stackoperator = popFromStack();

if( stackoperator == '(')

break;

addToPreExp(stackoperator);

}

}

else if(getPriority(exp[i]) == 0) // if it is operand

addToPreExp(exp[i]);

else if(getPriority(exp[i]) == 3) {

while(getPriority(stack[size]) >= getPriority(exp[i]))

addToPreExp(popFromStack());

pushToStack(exp[i]);

}

else { // if operator

while(getPriority(stack[size]) > getPriority(exp[i]))

addToPreExp(popFromStack());

pushToStack(exp[i]);

}

}

for(int i=size; i>=0; i--) // adding the remaining operators in stack if any

addToPreExp(popFromStack());

reverseExpression(preexp);

}

void main() {

char inexp[100];

printf("\n");

printf("Enter the Infix Expression : \n");

scanf("%[^\n]", inexp);

reverseExpression(inexp);

convertToPrefix(inexp);

displayPrefixExp();

}

OUTPUT:

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO IMPLEMENT A BINARY SEARCH TREE

\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

int data;

struct node \*left;

struct node \*right;

}treeptr;

treeptr node;

treeptr \*newNode(int value){

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

node->data = value;

node->left = NULL;

node->right = NULL;

return node;

}

treeptr \*insert(treeptr \*node,int value){

if(node==NULL){ //tree empty condition

return newNode(value);

}

if(value<node->data){

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

}

else if(value>node->data){

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

}

return node;

}

treeptr \*search(treeptr \*node,int value){

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

return node;

}

if(node->data>value){

return search(node->left,value);

}

if(node->data<value){

return search(node->right,value);

}

}

treeptr \*findMin(treeptr\* node)

{

while(node->left != NULL)

node = node->left;

return node;

}

treeptr \*delete(treeptr \*node, int data) {

if(node == NULL)

return node;

else if(data < node->data)

node->left = delete(node->left,data);

else if (data > node->data)

node->right = delete(node->right,data);

else {

if(node->left == NULL && node->right == NULL) {

free(node);

node = NULL;

}

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

treeptr \*temp = node;

node = node->right;

free(temp);

}

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

treeptr \*temp = node;

node = node->left;

free(temp);

}

else {

treeptr \*temp = findMin(node->right);

node->data = temp->data;

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

}

}

return node;

}

void inorderTrav(treeptr \*node){

if(!node) return;

inorderTrav(node->left);

printf("%d ",node->data);

inorderTrav(node->right);

}

void preorderTrav(treeptr \*node){

if(!node) return;

printf("%d ",node->data);

preorderTrav(node->left);

preorderTrav(node->right);

}

void postorderTrav(treeptr \*node){

if(!node) return;

postorderTrav(node->left);

postorderTrav(node->right);

printf("%d ",node->data);

}

void main(){

treeptr \*root = NULL;

treeptr \*srch = NULL;

int choice,c=1;

int ele;

inorderTrav(root);

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

printf("1.Insert Node\n");

printf("2.Preorder Traversal\n");

printf("3.Inorder Traversal\n");

printf("4.Postorder Traversal\n");

printf("5.Delete Node\n");

printf("6.Search for Element\n");

printf("7.Exit\n");

while(c>=1){

printf("\nEnter the choice : ");

scanf("%d",&choice);

switch(choice){

case 1: printf("Enter the element to be inserted : ");

scanf("%d",&ele);

if(c==1){root = insert(root,ele);c++;}

else{

insert(root,ele);

}

break;

case 2: printf("\nPreorder :\n");

preorderTrav(root);

break;

case 3: printf("\nInorder :\n");

inorderTrav(root);

break;

case 4: printf("\nPostorder :\n");

postorderTrav(root);

break;

case 5: printf("\nEnter the element to be Deleted : ");

scanf("%d",&ele);

delete(root,ele);

printf("\nDeleted !!");

break;

case 6: printf("\nEnter the element to be searched : ");

scanf("%d",&ele);

srch = search(root,ele);

if(srch == NULL){

printf("\nElement Not Found !!!");

}

else{

printf("\nElement Found !!!");

}

break;

case 7: return;

default: printf("\nEnter a Valid Choice !!!");

}

}

}

OUTPUT:

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO COUNT LEAF NODES OF A BST

\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

int data;

struct node \*left;

struct node \*right;

}treeptr;

treeptr node;

treeptr \*newNode(int value){

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

node->data = value;

node->left = NULL;

node->right = NULL;

return node;

}

treeptr \*insert(treeptr \*node,int value){

if(node==NULL){ //tree empty condition

return newNode(value);

}

if(value<node->data){

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

}

else if(value>node->data){

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

}

return node;

}

treeptr \*findMin(treeptr\* node)

{

while(node->left != NULL)

node = node->left;

return node;

}

treeptr \*delete(treeptr \*node, int data) {

if(node == NULL)

return node;

else if(data < node->data)

node->left = delete(node->left,data);

else if (data > node->data)

node->right = delete(node->right,data);

else {

if(node->left == NULL && node->right == NULL) {

free(node);

node = NULL;

}

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

treeptr \*temp = node;

node = node->right;

free(temp);

}

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

treeptr \*temp = node;

node = node->left;

free(temp);

}

else {

treeptr \*temp = findMin(node->right);

node->data = temp->data;

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

}

}

return node;

}

void inorderTrav(treeptr \*node){

if(!node) return;

inorderTrav(node->left);

printf("%d ",node->data);

inorderTrav(node->right);

}

int no\_leafnode(treeptr \*node){

if(node == NULL)

return 0;

if(node->left == NULL && node->right == NULL)

return 1;

else

return no\_leafnode(node->left) + no\_leafnode(node->right);

}

void main(){

treeptr \*root = NULL;

treeptr \*srch = NULL;

int choice,c=1;

int ele,num;

inorderTrav(root);

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

printf("1.Insert Node\n");

printf("2.Inorder Traversal\n");

printf("3.Delete Node\n");

printf("4.Number of Leaf Nodes\n");

printf("5.Exit\n");

while(c>=1){

printf("\nEnter the choice : ");

scanf("%d",&choice);

switch(choice){

case 1: printf("Enter the element to be inserted : ");

scanf("%d",&ele);

if(c==1){root = insert(root,ele);c++;}

else{

insert(root,ele);

}

break;

case 2: printf("\nInorder :\n");

inorderTrav(root);

break;

case 3: printf("\nEnter the element to be Deleted : ");

scanf("%d",&ele);

delete(root,ele);

printf("\nDeleted !!");

break;

case 4: num = no\_leafnode(root);

printf("The number of Leaf Nodes is -> %d !!",num);

break;

case 5: return;

default: printf("\nEnter a Valid Choice !!!");

}

}

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO FIND NUMBER OF NODES,EDGES AND HEIGHT OF A BST

\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

int data;

struct node \*left;

struct node \*right;

}treeptr;

treeptr node;

treeptr \*newNode(int value){

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

node->data = value;

node->left = NULL;

node->right = NULL;

return node;

}

treeptr \*insert(treeptr \*node,int value){

if(node==NULL){

return newNode(value);

}

if(value<node->data){

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

}

else if(value>node->data){

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

}

return node;

}

treeptr \*findMin(treeptr\* node)

{

while(node->left != NULL)

node = node->left;

return node;

}

treeptr \*delete(treeptr \*node, int data) {

if(node == NULL)

return node;

else if(data < node->data)

node->left = delete(node->left,data);

else if (data > node->data)

node->right = delete(node->right,data);

else {

if(node->left == NULL && node->right == NULL) {

free(node);

node = NULL;

}

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

treeptr \*temp = node;

node = node->right;

free(temp);

}

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

treeptr \*temp = node;

node = node->left;

free(temp);

}

else {

treeptr \*temp = findMin(node->right);

node->data = temp->data;

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

}

}

return node;

}

void inorderTrav(treeptr \*node){

if(!node) return;

inorderTrav(node->left);

printf("%d ",node->data);

inorderTrav(node->right);

}

int no\_leafnode(treeptr \*node){

if(node == NULL)

return 0;

if(node->left == NULL && node->right == NULL)

return 1;

else

return no\_leafnode(node->left) + no\_leafnode(node->right);

}

int no\_node(treeptr \*node){

if(node == NULL)

return 0;

else

return 1 + no\_node(node->left) + no\_node(node->right);

}

int height(treeptr \*node){

if(node == NULL)

return -1;

else{

int l\_height = height(node->left);

int r\_height = height(node->right);

if(l\_height>r\_height)

return l\_height+1;

else

return r\_height+1;

}

}

void main(){

treeptr \*root = NULL;

treeptr \*srch = NULL;

int choice,c=1;

int ele,num;

inorderTrav(root);

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

printf("1.Insert Node\n");

printf("2.Inorder Traversal\n");

printf("3.Delete Node\n");

printf("4.Number of Leaf Nodes\n");

printf("5.Number of Nodes\n");

printf("6.Number of Edges\n");

printf("7.Height of BST\n");

printf("8.Exit\n");

while(c>=1){

printf("\nEnter the choice : ");

scanf("%d",&choice);

switch(choice){

case 1: printf("Enter the element to be inserted : ");

scanf("%d",&ele);

if(c==1){root = insert(root,ele);c++;}

else{

insert(root,ele);

}

break;

case 2: printf("\nInorder :\n");

inorderTrav(root);

break;

case 3: printf("\nEnter the element to be Deleted : ");

scanf("%d",&ele);

delete(root,ele);

printf("\nDeleted !!");

break;

case 4: num = no\_leafnode(root);

printf("The number of Leaf Nodes is -> %d !!",num);

break;

case 5: num = no\_node(root);

printf("The number of Nodes is -> %d !!",num);

break;

case 6: num = no\_node(root);

printf("The number of Edges is -> %d !!",num-1);

break;

case 7: num = height(root) + 1; // taking into consideration root node

printf("The height of BST is -> %d !!",num);

break;

case 8: return;

default: printf("\nEnter a Valid Choice !!!");

}

}

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO PERFORM DFS ON A GRAPH

\*/

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

int ADJ\_MAT[MAX\_SIZE][MAX\_SIZE]={{0}};

int isVisited[MAX\_SIZE] = {0};

void addEdge(int v1,int v2){

ADJ\_MAT[v1][v2] = 1;

ADJ\_MAT[v2][v1] = 1;

}

void DFS(int v,int n){

isVisited[v] = 1;

printf("%d ",v);

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

if (ADJ\_MAT[v][i] && !isVisited[i]){

DFS(i,n);

}

}

}

void main(){

int v,e;

printf("Enter the number of vertices in the graph : ");

scanf("%d",&v);

int v1,v2;

printf("Enter the number of edges in the graph : ");

scanf("%d",&e);

printf("enter v1 v2\n");

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

scanf("%d %d",&v1,&v2);

addEdge(v1,v2);

}

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

if (!isVisited[i]){

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

DFS(i,v);

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

}

}

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO PERFROM BFS ON A GRAPH

\*/

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

typedef struct vertex{

int data;

struct vertex \*next;

}\*vertex;

vertex adjList[MAX\_SIZE];

int queue[MAX\_SIZE];

int isVisited[MAX\_SIZE] = {0};

int front = -1, rear = -1;

vertex createItem(int data){

vertex newVertex = (vertex)malloc(MAX\_SIZE\*sizeof(int));

newVertex->data = data;

newVertex->next = NULL;

return newVertex;

}

void addEdge(int v1,int v2){

if (!adjList[v1]){

adjList[v1] = createItem(v2);

} else{

vertex curr = adjList[v1];

while (curr->next){

curr = curr->next;

}

curr->next = createItem(v2);

}

if (!adjList[v2]){

adjList[v2] = createItem(v1);

} else{

vertex curr = adjList[v2];

while (curr->next){

curr = curr->next;

}

curr->next = createItem(v1);

}

}

void enQueue(int element){

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

printf("Queue is full\n");

return;

}

if (front==-1){

front++;

}

queue[++rear] = element;

}

int deQueue(){

if (front == -1){

printf("Queue empty\n");

return -1;

}

int data = queue[front];

if (front == rear)

{

front = -1;rear = -1;

}

else{

if (front == rear){

front = -1;rear = -1;

}

else{

front++;

}

}

return data;

}

int isEmpty(){

if (front == -1){

return 1;

}

return 0;

}

void bfs(int v){

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

if (!isVisited[i]){

printf("\n");

enQueue(i);

isVisited[i] = 1;

int i;

while (!isEmpty()){

i = deQueue();

printf("%d ",i);

vertex curr = adjList[i];

while (curr){

if (!isVisited[curr->data]){

enQueue(curr->data);

isVisited[curr->data] = 1;

}

curr = curr->next;

}

}

printf("\n");

}

}

}

void main()

{

int v,e;

printf("Enter the number of vertices in the graph : ");

scanf("%d",&v);

int v1,v2;

printf("Enter the number of edges in the graph : ");

scanf("%d",&e);

printf("enter v1 v2\n");

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

{

scanf("%d %d",&v1,&v2);

addEdge(v1,v2);

}

bfs(v);

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO PERFORM SELECTION SORT

\*/

#include <stdio.h>

#define MAX\_SIZE 101

void selection(int arr[], int n)

{

int i, j, small;

for (i = 0; i < n-1; i++)

{

small = i;

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

if (arr[j] < arr[small])

small = j;

int temp = arr[small];

arr[small] = arr[i];

arr[i] = temp;

}

}

void display(int a[], int n)

{

int i;

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

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

}

int main()

{

int n,a[MAX\_SIZE];

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

scanf("%d",&n);

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

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

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

}

printf("Before sorting array elements are - \n");

display(a, n);

selection(a, n);

printf("\nAfter sorting array elements are - \n");

display(a, n);

printf("\n");

return 0;

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO PERFORM INSERTION SORT

\*/

#include <stdio.h>

#define MAX\_SIZE 101

void insertion(int a[], int n)

{

int i, j, temp;

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

temp = a[i];

j = i - 1;

while(j>=0 && temp <= a[j])

{

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

j = j-1;

}

a[j+1] = temp;

}

}

void display(int a[], int n)

{

int i;

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

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

}

int main()

{

int n,a[MAX\_SIZE];

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

scanf("%d",&n);

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

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

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

}

printf("Before sorting array elements are - \n");

display(a, n);

insertion(a, n);

printf("\nAfter sorting array elements are - \n");

display(a, n);

printf("\n");

return 0;

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO PERFORM QUICK SORT

\*/

#include <stdio.h>

#define MAX\_SIZE 101

int partition (int a[], int start, int end)

{

int pivot = a[end];

int i = (start - 1);

for (int j = start; j <= end - 1; j++)

{

if (a[j] < pivot)

{

i++;

int t = a[i];

a[i] = a[j];

a[j] = t;

}

}

int t = a[i+1];

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

a[end] = t;

return (i + 1);

}

void quick(int a[], int start, int end)

{

if (start < end)

{

int p = partition(a, start, end);

quick(a, start, p - 1);

quick(a, p + 1, end);

}

}

void display(int a[], int n)

{

int i;

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

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

}

int main()

{

int n,a[MAX\_SIZE];

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

scanf("%d",&n);

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

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

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

}

printf("Before sorting array elements are - \n");

display(a, n);

quick(a,0,n-1);

printf("\nAfter sorting array elements are - \n");

display(a, n);

printf("\n");

return 0;

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO PERFORM MERGE SORT

\*/

#include <stdio.h>

#define MAX\_SIZE 101

void merge(int a[], int beg, int mid, int end)

{

int i, j, k;

int n1 = mid - beg + 1;

int n2 = end - mid;

int LeftArray[n1], RightArray[n2];

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

LeftArray[i] = a[beg + i];

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

RightArray[j] = a[mid + 1 + j];

i = 0;

j = 0;

k = beg;

while (i < n1 && j < n2)

{

if(LeftArray[i] <= RightArray[j])

{

a[k] = LeftArray[i];

i++;

}

else

{

a[k] = RightArray[j];

j++;

}

k++;

}

while (i<n1)

{

a[k] = LeftArray[i];

i++;

k++;

}

while (j<n2)

{

a[k] = RightArray[j];

j++;

k++;

}

}

void mergeSort(int a[], int beg, int end)

{

if (beg < end)

{

int mid = (beg + end) / 2;

mergeSort(a, beg, mid);

mergeSort(a, mid + 1, end);

merge(a, beg, mid, end);

}

}

void display(int a[], int n)

{

int i;

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

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

}

int main()

{

int n,a[MAX\_SIZE];

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

scanf("%d",&n);

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

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

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

}

printf("Before sorting array elements are - \n");

display(a, n);

mergeSort(a,0,n-1);

printf("\nAfter sorting array elements are - \n");

display(a, n);

printf("\n");

return 0;

}

OUTPUT :

Text

Description automatically generated

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO PERFORM HEAP SORT

\*/

#include <stdio.h>

#define MAX\_SIZE 101

void heapify(int a[], int n, int i)

{

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && a[left] > a[largest])

largest = left;

if (right < n && a[right] > a[largest])

largest = right;

if (largest != i) {

int temp = a[i];

a[i] = a[largest];

a[largest] = temp;

heapify(a, n, largest);

}

}

void heapSort(int a[], int n)

{

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

heapify(a, n, i);

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

int temp = a[0];

a[0] = a[i];

a[i] = temp;

heapify(a, i, 0);

}

}

void display(int a[], int n)

{

int i;

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

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

}

int main()

{

int n,a[MAX\_SIZE];

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

scanf("%d",&n);

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

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

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

}

printf("Before sorting array elements are - \n");

display(a, n);

heapSort(a,n);

printf("\nAfter sorting array elements are - \n");

display(a, n);

printf("\n");

return 0;

}

OUTPUT :

A screenshot of a computer

Description automatically generated with medium confidence

/\*

AMAL GOVIND S

CS3B 10

MDL20CS021

#PROGRAM TO PERFORM HASHING

\*/

#include <stdio.h>

#define SIZE 11

int hashTable[SIZE] = {0};

void performHash(int inputArr[], int inputSize){

int key;

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

key = inputArr[i] % SIZE;

if (hashTable[key] == 0)

hashTable[key] = inputArr[i];

else{

int j;

j = (key + 1) % SIZE;

while (1){

if (hashTable[j] == 0){

hashTable[j] = inputArr[i];

break;

}

else{

j = (j + 1) % SIZE;

}

}

}

}

}

void printHashTable();

int main(){

int inputArr[SIZE];

int inputSize;

printf("Enter size of input array\n");

scanf("%d", &inputSize);

if (inputSize > SIZE){

printf("\nSorry, Max size exceeded");

return 0;

}

printf("Enter elements\n");

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

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

performHash(inputArr, inputSize);

printHashTable();

return 0;

}

void printHashTable(){

printf("\nHash table");

printf("\nIndex:\t");

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

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

}

printf("\nValue:\t");

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

hashTable[i] != 0 ? printf("%d\t", hashTable[i]) : printf("\_\_\t");

}

printf("\n");

}

OUTPUT :

Graphical user interface, application

Description automatically generated