DSA LAB CAT

<https://paste.sh/XLLEDxRR#SikeW39gGnUy0rXmfsiPGHMU>

LINKED LIST POLYNOMIAL ADDITION

LINKED LIST POLYNOMIAL ADDITION

LINKED LIST POLYNOMIAL ADDITION

LINKED LIST POLYNOMIAL ADDITION

LINKED LIST POLYNOMIAL ADDITION

LINKED LIST POLYNOMIAL ADDITION

LINKED LIST POLYNOMIAL ADDITION

LINKED LIST POLYNOMIAL ADDITION

--------------------------------

#include<stdio.h>

#include<stdlib.h>

typedef struct node {

int coeff,expo;

struct node \*next;

}N;

N \*input();

N \*p\_addition(N \*,N \*);

N \*getnode();

N \*getnode() {

N \*new;

new=(N\*)malloc(sizeof(N));

new->next=NULL;

}

N \*p\_addition(N \*ptr,N \*qtr) {

N \*rptr=getnode(),\*r1ptr=rptr,\*aptr=NULL;

int s;

printf("\n...................addition starts...................\n");

while(ptr!=NULL&&qtr!=NULL) {

if(ptr->expo==qtr->expo) {

s=ptr->coeff+qtr->coeff;

if(s!=0) {

aptr=getnode();

aptr->coeff=s;

aptr->expo=ptr->expo;

r1ptr->next=aptr;

r1ptr=aptr;

}

ptr=ptr->next;

qtr=qtr->next;

}

else if(ptr->expo>qtr->expo) {

aptr=getnode();

aptr->coeff=ptr->coeff;

aptr->expo=ptr->expo;

ptr=ptr->next;

r1ptr->next=aptr;

r1ptr=aptr;

}

else {

aptr=getnode();

aptr->coeff=qtr->coeff;

aptr->expo=qtr->expo;

qtr=qtr->next;

r1ptr->next=aptr;

r1ptr=aptr;

}

}

while(ptr!=NULL) {

aptr=getnode();

aptr->coeff=ptr->coeff;

aptr->expo=ptr->expo;

ptr=ptr->next;

r1ptr->next=aptr;

r1ptr=aptr;

}

while(qtr!=NULL) {

aptr=getnode();

aptr->coeff=qtr->coeff;

aptr->expo=qtr->expo;

qtr=qtr->next;

r1ptr->next=aptr;

r1ptr=aptr;

}

return (rptr->next);

}

N \*input()

{

N \*new,\*qtr,\*ptr;

int i,n;

new=(N \*)malloc(sizeof(N));

new->next=NULL;

qtr=new;

printf("Enter number of nodes");

scanf("%d",&n);

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

{

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

printf("Enter Coeff & Expo respectively");

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

ptr->next=NULL;

qtr->next=ptr;

qtr=qtr->next;

}

return(new->next);

}

int main()

{

N \*ptr,\*qtr,\*add;

ptr=input();

qtr=input();

add=p\_addition(ptr,qtr);

while(add!=NULL) {

printf("%dX^%d+",add->coeff,add->expo);

add=add->next;

}

}

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Evaluation of postfix expression using stack

Evaluation of postfix expression using stack

Evaluation of postfix expression using stack

Evaluation of postfix expression using stack

Evaluation of postfix expression using stack

Evaluation of postfix expression using stack

Evaluation of postfix expression using stack

Evaluation of postfix expression using stack

#include <stdio.h>

#include <ctype.h>

# define MAXSTACK 100

# define POSTFIXSIZE 100

int stack[MAXSTACK];

int top = -1 ;

void push(int item)

{

if(top >= MAXSTACK -1)

{

printf("stack over flow");

return;

}

else

{

top = top + 1 ;

stack[top]= item;

printf("the value pushed into stack : %d\n",item);

}

}

int pop()

{

int item;

if(top <0)

{

printf("stack under flow");

}

else

{

item = stack[top];

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

top = top - 1;

return item;

}

}

void EvalPostfix(char postfix[])

{

int i ;

char ch;

int val;

int A, B ;

for (i = 0 ; postfix[i] != ')'; i++)

{

ch = postfix[i];

if (isdigit(ch))

{

push(ch - '0');

}else if (ch == '+' || ch == '-' || ch == '\*' || ch == '/')

{

A = pop();

B = pop();

switch (ch) /\* ch is an operator \*/

{

case '\*':

val = B \* A;

printf("the value after multiplication : %d\n",val);

break;

case '/':

val = B / A;

printf("the value after division : %d\n",val);

break;

case '+':

val = B + A;

printf("the value after addition : %d\n",val);

break;

case '-':

val = B - A;

printf("the value after subraction : %d\n",val);

break;

}

push(val);

}

}

printf( " \n Result of expression evaluation : %d \n", pop()) ;

}

int main()

{

int i ;

char postfix[POSTFIXSIZE];

printf( " \nEnter postfix expression,press right parenthesis ')' for end expression \n: ");

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

{

scanf("%c", &postfix[i]);

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

{ break; }

}

EvalPostfix(postfix);

return 0;

}

OR

OR

OR

OR

OR

OR

OR

#include<stdio.h> int stack[20]; int top = -1;

void push(int x) { stack[++top] = x; }

int pop() { return stack[top--]; }

int main() { char exp[20]; char \*e; int n1,n2,n3,num; printf("Enter the expression :: "); scanf("%s",exp);

e = exp; while(\*e != '\0') { if(isdigit(\*e)) {

} else {

num = \*e - 48; push(num);

n1 = pop(); n2 = pop(); switch(\*e) { case '+': {

break; }

n3 = n1 + n2;

case '-': { n3 = n2 - n1; break; } case '\*': { n3 = n1 \* n2; break; } case '/': { n3 = n2 / n1; break; }

} push(n3);

}

e++;

} printf("\nThe result of expression %s = %d\n\n",exp,pop()); return 0;

}

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Functionalaties of circular queue

Functionalaties of circular queue

Functionalaties of circular queue

Functionalaties of circular queue

Functionalaties of circular queue

Functionalaties of circular queue

code : #include <stdio.h>

#define SIZE 5

int

items[SIZE]; int front = -1, rear =-1;

int isFull() { if( (front == rear + 1) || (front == 0 && rear == SIZE-1)) return 1; return 0; }

int isEmpty() { if(front == -1) return 1; return 0; }

void enQueue(int element) { if(isFull()) printf("\n Queue is full!! \n"); else { if(front == -1) front = 0; rear = (rear + 1) % SIZE; items[rear] = element; printf("\n Inserted -> %d", element); } }

int deQueue() { int element; if(isEmpty()) { printf("\n Queue is empty !! \n"); return(-1); } else { element = items[front]; if (front == rear){ front = -1; rear = -1; } else {

front = (front + 1) % SIZE;

} printf("\n Deleted element -> %d \n", element); return(element);

}

}

void display() { int i; if(isEmpty()) printf(" \n Empty Queue\n"); else { printf("\n Front -> %d ",front); printf("\n Items -> "); for( i = front; i!=rear; i=(i+1)%SIZE) { printf("%d ",items[i]); } printf("%d ",items[i]); printf("\n Rear -> %d \n",rear); } }

int main() { deQueue();

enQueue(1); enQueue(2); enQueue(3); enQueue(4); enQueue(5); enQueue(6); display(); deQueue();displ ay(); enQueue(7);

display(); enQueue(8); return 0;

}

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

TREE TRAVERSAL

TREE TRAVERSAL

TREE TRAVERSAL

TREE TRAVERSAL

TREE TRAVERSAL

TREE TRAVERSAL

TREE TRAVERSAL

TREE TRAVERSAL

TREE TRAVERSAL

TREE TRAVERSAL

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

using namespace std;

struct node

{

int value;

node\* left;

node\* right;

};

struct node\* root;

struct node\* insert(struct node\* r, int data);

void inOrder(struct node\* r);

void preOrder(struct node\* r);

void postOrder(struct node\* r);

int main()

{

system("color E");

root = NULL;

int j,n, v;

cout<<"\nHOW MANY DATA'S DO YOU WANT TO INSERT FOR TREE TRAVERSAL?\n";

cin>>n;

int i;

char a[n];

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

cin>>a[i];

}

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

v=a[k];

root = insert(root, v);

}

cout<<"\nINORDER TRAVERSAL: ";

inOrder(root);

cout<<"\n";

cout<<"PREORDER TRAVERSAL: ";

preOrder(root);

cout<<"\n";

cout<<"POSTORDER TRAVERSAL: ";

postOrder(root);

cout<<"\n";

return 0;

}

struct node\* insert(struct node\* r, int data)

{

if(r==NULL)

{

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

r->value = data;

r->left = NULL;

r->right = NULL;

}

else if(data < r->value){

r->left = insert(r->left, data);

}

else {

r->right = insert(r->right, data);

}

return r;

}

void inOrder(struct node\* r)

{

if(r!=NULL){

inOrder(r->left);

cout<<(char)r->value;

inOrder(r->right);

}

}

void preOrder(struct node\* r)

{

if(r!=NULL){

cout<<(char)r->value;

preOrder(r->left);

preOrder(r->right);

}

}

void postOrder(struct node\* r)

{

if(r!=NULL){

postOrder(r->left);

postOrder(r->right);

cout<<(char)r->value;

}

}

==========================================================================================================================================================================================================

HEAP SORT

#include<stdio.h>

void create(int []);

void down\_adjust(int [],int);

void main()

{

int heap[30],n,i,last,temp;

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

scanf("%d",&n);

printf("\nEnter elements:");

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

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

//create a heap

heap[0]=n;

create(heap);

//sorting

while(heap[0] > 1)

{

//swap heap[1] and heap[last]

last=heap[0];

temp=heap[1];

heap[1]=heap[last];

heap[last]=temp;

heap[0]--;

down\_adjust(heap,1);

}

//print sorted data

printf("\nArray after sorting:\n");

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

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

}

void create(int heap[])

{

int i,n;

n=heap[0]; //no. of elements

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

down\_adjust(heap,i);

}

void down\_adjust(int heap[],int i)

{

int j,temp,n,flag=1;

n=heap[0];

while(2\*i<=n && flag==1)

{

j=2\*i; //j points to left child

if(j+1<=n && heap[j+1] > heap[j])

j=j+1;

if(heap[i] > heap[j])

flag=0;

else

{

temp=heap[i];

heap[i]=heap[j];

heap[j]=temp;

i=j;

}

}

}