1. singly linked list to insert at beginning and any position

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

class SinglyLinkedList {

public:

Node\* head;

SinglyLinkedList() : head(nullptr) {}

void insertAtBeginning(int data) {

Node\* newNode = new Node{data, head};

head = newNode;

}

void insertAtPosition(int data, int position) {

Node\* newNode = new Node{data, nullptr};

if (position == 1) {

insertAtBeginning(data);

return;

}

Node\* temp = head;

for (int i = 1; i < position - 1 && temp != nullptr; i++) {

temp = temp->next;

}

if (temp != nullptr) {

newNode->next = temp->next;

temp->next = newNode;

} else {

cout << "Position out of bounds" << endl;

}

}

void display() {

Node\* temp = head;

while (temp != nullptr) {

cout << temp->data << " -> ";

temp = temp->next;

}

cout << "NULL" << endl;

}

};

int main() {

SinglyLinkedList list;

list.insertAtBeginning(10);

list.insertAtBeginning(20);

list.insertAtPosition(30, 2);

list.insertAtPosition(40, 3);

list.display();

return 0;

}

1. singly linked list to insert at end and any position

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

class SinglyLinkedList {

public:

Node\* head;

SinglyLinkedList() : head(nullptr) {}

void insertAtEnd(int data) {

Node\* newNode = new Node{data, nullptr};

if (head == nullptr) {

head = newNode;

} else {

Node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

}

}

void insertAtPosition(int data, int position) {

Node\* newNode = new Node{data, nullptr};

if (position == 1) {

newNode->next = head;

head = newNode;

return;

}

Node\* temp = head;

for (int i = 1; i < position - 1 && temp != nullptr; i++) {

temp = temp->next;

}

if (temp != nullptr) {

newNode->next = temp->next;

temp->next = newNode;

} else {

cout << "Position out of bounds" << endl;

}

}

void display() {

Node\* temp = head;

while (temp != nullptr) {

cout << temp->data << " -> ";

temp = temp->next;

}

cout << "NULL" << endl;

}

};

int main() {

SinglyLinkedList list;

list.insertAtEnd(10);

list.insertAtEnd(20);

list.insertAtPosition(30, 2);

list.insertAtPosition(40, 3);

list.display();

return 0;

}

1. doubly linked list deletion at the end and any position

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node\* prev;

};

class DoublyLinkedList {

public:

Node\* head;

DoublyLinkedList() : head(nullptr) {}

void insertAtEnd(int data) {

Node\* newNode = new Node{data, nullptr, nullptr};

if (head == nullptr) {

head = newNode;

} else {

Node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

}

void deleteAtBeginning() {

if (head == nullptr) return;

Node\* temp = head;

head = head->next;

if (head != nullptr) head->prev = nullptr;

delete temp;

}

void deleteAtPosition(int position) {

if (head == nullptr || position <= 0) return;

if (position == 1) {

deleteAtBeginning();

return;

}

Node\* temp = head;

for (int i = 1; i < position && temp != nullptr; i++) {

temp = temp->next;

}

if (temp == nullptr) return;

if (temp->next != nullptr) temp->next->prev = temp->prev;

if (temp->prev != nullptr) temp->prev->next = temp->next;

delete temp;

}

void display() {

Node\* temp = head;

while (temp != nullptr) {

cout << temp->data << " <-> ";

temp = temp->next;

}

cout << "NULL" << endl;

}

};

int main() {

DoublyLinkedList list;

list.insertAtEnd(10);

list.insertAtEnd(20);

list.insertAtEnd(30);

list.deleteAtBeginning();

list.deleteAtPosition(8);

list.display();

return 0;

}

1. stack operations

( push, pop, display)

#include <iostream>

using namespace std;

int stack[100], n=100, top=-1;

void Push(){

int val;

if (top>=n-1)

cout<<"stack overflow"<<endl;

else{

//if(front==-1)

//front=0;

cout<<"enter value:"<<endl;

cin>>val;

top++;

stack[top]=val;

}

}

void Pop(){

if (top==-1){

cout<<"stack underflow"<<endl;

return;

}

else{

cout<<"element popped:"<<stack[top]<<endl;

top--;

}

}

void Display(){

if(top==-1)

cout<<"stack is empty"<<endl;

else{

cout<<"elements in the stack are:";

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

cout<<stack[i]<<" ";

cout<<endl;

}

}

int main(){

int ch;

cout<<"1)Push"<<endl;

cout<<"2)Pop"<<endl;

cout<<"3)Display"<<endl;

cout<<"4)Exit"<<endl;

do{

cout<<"enter your choice:"<<endl;

cin>>ch;

switch(ch){

case 1: Push();

break;

case 2: Pop();

break;

case 3: Display();

break;

case 4: cout<<"exit"<<endl;

break;

default: cout<<"invalid choice"<<endl;

}

}

while(ch!=4);

return 0;

}

1. queue operations

(push, pop, display)

#include <iostream>

using namespace std;

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

void Enqueue(){

int val;

if (rear==n-1)

cout<<"queue overflow"<<endl;

else{

if(front==-1)

front=0;

cout<<"insert"<<endl;

cin>>val;

rear++;

queue[rear]=val;

}

}

void Dequeue(){

int val;

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

cout<<"queue underflow"<<endl;

return;

}

else{

cout<<"element deleted:"<<queue[front]<<endl;

front++;

}

}

void Display(){

if(front==-1)

cout<<"queue is empty"<<endl;

else{

cout<<"elements in the queue are:";

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

cout<<queue[i]<<" ";

cout<<endl;

}

}

int main(){

int ch;

cout<<"1)Enqueue"<<endl;

cout<<"2)Dequeue"<<endl;

cout<<"3)Display"<<endl;

cout<<"4)Exit"<<endl;

do{

cout<<"enter your choice:"<<endl;

cin>>ch;

switch(ch){

case 1: Enqueue();

break;

case 2: Dequeue();

break;

case 3: Display();

break;

case 4: cout<<"exit"<<endl;

break;

default: cout<<"invalid choice"<<endl;

}

}

while(ch!=4);

return 0;

}

1. infix to postfix expression

#include <iostream>

#include <stack>

#include <string>

using namespace std;

// Function to return precedence of operators

int precedence(char op) {

if (op == '+' || op == '-') return 1;

if (op == '\*' || op == '/') return 2;

if (op == '^') return 3;

return 0;

}

// Function to convert infix expression to postfix

string infixToPostfix(const string& infix) {

stack<char> operators;

string postfix;

for (char ch : infix) {

// If the character is an operand, add it to the output

if (isalnum(ch)) {

postfix += ch;

}

// If the character is '(', push it to the stack

else if (ch == '(') {

operators.push(ch);

}

// If the character is ')', pop and output from the stack

// until an '(' is encountered

else if (ch == ')') {

while (!operators.empty() && operators.top() != '(') {

postfix += operators.top();

operators.pop();

}

operators.pop(); // pop '('

}

// If an operator is encountered

else {

while (!operators.empty() && precedence(operators.top()) >= precedence(ch)) {

postfix += operators.top();

operators.pop();

}

operators.push(ch);

}

}

// Pop all remaining operators from the stack

while (!operators.empty()) {

postfix += operators.top();

operators.pop();

}

return postfix;

}

int main() {

string infix;

cout << "Enter an infix expression: ";

cin >> infix;

string postfix = infixToPostfix(infix);

cout << "Postfix expression: " << postfix << endl;

return 0;

}

1. infix to prefix expression

#include <iostream>

#include <stack>

#include <string>

#include <algorithm>

using namespace std;

// Function to return precedence of operators

int precedence(char op) {

if (op == '+' || op == '-') return 1;

if (op == '\*' || op == '/') return 2;

if (op == '^') return 3;

return 0;

}

// Function to convert infix expression to prefix

string infixToPrefix(const string& infix) {

string reversedInfix = infix;

reverse(reversedInfix.begin(), reversedInfix.end());

// Reverse the brackets

for (char &ch : reversedInfix) {

if (ch == '(') ch = ')';

else if (ch == ')') ch = '(';

}

// Convert reversed infix to postfix

stack<char> operators;

string postfix;

for (char ch : reversedInfix) {

if (isalnum(ch)) {

postfix += ch;

} else if (ch == '(') {

operators.push(ch);

} else if (ch == ')') {

while (!operators.empty() && operators.top() != '(') {

postfix += operators.top();

operators.pop();

}

operators.pop();

} else {

while (!operators.empty() && precedence(operators.top()) > precedence(ch)) {

postfix += operators.top();

operators.pop();

}

operators.push(ch);

}

}

while (!operators.empty()) {

postfix += operators.top();

operators.pop();

}

// Reverse the postfix result to get prefix

reverse(postfix.begin(), postfix.end());

return postfix;

}

int main() {

string infix;

cout << "Enter an infix expression: ";

cin >> infix;

string prefix = infixToPrefix(infix);

cout << "Prefix expression: " << prefix << endl;

return 0;

}

1. (a) linear search

#include <iostream>

using namespace std;

int linearSearch(int arr[], int n, int key){

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

if (arr[i]==key)

return i;

}

return -1;

}

int main(){

int n,key;

cout<<"enter the no.of elements:";

cin>>n;

int \*arr= new int[n];

cout<<"enter the elements:";

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

cin>>arr[i];

}

cout<<"enter the element to search:";

cin>>key;

int index= linearSearch(arr,n,key);

if(index!=-1){

cout<<"element found at index:"<<index<<endl;

}

else{

cout<<"element not found";

}

delete[] arr;

return 0;

}

8.) (b) binary search

#include <iostream>

#include <algorithm>

using namespace std;

int binarySearch(int arr[], int left, int right, int key){

while (left<right){

int mid= left+ (right-left)/2;

if(arr[mid]==key)

return mid;

if(arr[mid]<key)

left= mid+1;

else

right= mid-1;

}

return -1;

}

int main(){

int n, key;

cout<<"enter the no.of elements:";

cin>>n;

int \*arr= new int[n];

cout<<"enter the elements:";

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

cin>>arr[i];

}

cout<<"enter the element to search:";

cin>>key;

sort(arr,arr+n);

int index= binarySearch(arr,0,n-1,key);

if(index!=-1)

cout<<"element found at index:"<<index<<endl;

else

cout<<"element not found";

delete[] arr;

return 0;

}

1. bubble sort

#include <iostream>

using namespace std;

void bubbleSort(int arr[], int n){

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

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

if(arr[j]>arr[j+1]){

swap(arr[j],arr[j+1]);

}

}

}

}

void displayArray(int arr[], int n){

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

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

}

cout<<endl;

}

int main(){

int n;

cout<<"enter the no.of elements:";

cin>>n;

int arr[100];

cout<<"enter the elements:";

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

cin>>arr[i];

}

bubbleSort(arr,n);

cout<<"sorted array:";

displayArray(arr, n);

return 0;

}

1. selection sort

#include <iostream>

using namespace std;

void selectionSort(int arr[], int n){

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

int min\_ind=i;

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

if(arr[j]<arr[min\_ind]){

min\_ind=j;

}

}

if (min\_ind!=i){

swap(arr[i], arr[min\_ind]);

}

}

}

void displayArray(int arr[], int n){

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

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

}

cout<<endl;

}

int main(){

int n;

cout<<"enter the no.of elements:";

cin>>n;

int arr[n];

cout<<"enter the elements:";

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

cin>>arr[i];

}

selectionSort(arr,n);

cout<<"sorted array:";

displayArray(arr, n);

return 0;

}

1. insertion sort

#include <iostream>

using namespace std;

void insertionSort(int arr[], int n){

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

int key= arr[i];

int j= i-1;

while(j>=0 && arr[j]>key){

arr[j+1]=arr[j];

j--;

}

arr[j+1]=key;

}

}

void displayArray(int arr[], int n){

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

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

}

cout<<endl;

}

int main(){

int n;

cout<<"enter the no.of elements:";

cin>>n;

int arr[n];

cout<<"enter the elements:";

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

cin>>arr[i];

}

insertionSort(arr,n);

cout<<"sorted array:";

displayArray(arr, n);

return 0;

}

1. merge sort

#include<iostream>

using namespace std;

void merge(int arr[],int low, int high, int mid){

int i,j,k;

int n1=mid-low+1;

int n2=high-mid;

int left\_array[n1],right\_array[n2];

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

left\_array[i]=arr[low+i];

}

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

right\_array[j]=arr[mid+1+j];

}

i=0;

j=0;

k=low;

while(i<n1 && j<n2){

if(left\_array[i]<=right\_array[j]){

arr[k]=left\_array[i];

i++;

}

else{

arr[k]=right\_array[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = left\_array[i];

i++;

k++;

}

while (j < n2) {

arr[k] = right\_array[j];

j++;

k++;

}

}

void merge\_sort(int arr[],int low,int high){

if(low<high){

int mid=(low+high)/2;

merge\_sort(arr,low,mid);

merge\_sort(arr,mid+1,high);

merge(arr,low,high,mid);

}

}

void print\_array(int arr[],int n){

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

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

}

cout<<endl;

}

int main(){

int n;

cout<<"Enter size of array: ";

cin>>n;

int arr[n];

cout<<"Enter Elements:"<<" ";

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

cin>>arr[i];

}

merge\_sort(arr,0,n-1);

cout<<"Sorted Array is:"<< " ";

print\_array(arr,n);

return 0;

}

1. quick sort

#include <iostream>

using namespace std;

int partition(int arr[], int l, int h) {

int pivot = arr[h];

int i = (l-1);

for (int j = l; j <= h - 1; j++) {

if (arr[j] <= pivot) {

i++;

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

}

}

swap(arr[i + 1], arr[h]);

return (i + 1);

}

void quick\_sort(int arr[], int l, int h) {

if (l< h) {

int pi = partition(arr, l, h);

quick\_sort(arr, l, pi - 1);

quick\_sort(arr, pi + 1, h);

}

}

void print\_array(int arr[], int n) {

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

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

}

cout << endl;

}

int main() {

int n;

cout << "Enter size of array: ";

cin >> n;

int arr[n];

cout << "Enter elements: ";

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

cin >> arr[i];

}

quick\_sort(arr, 0, n - 1);

cout << "Sorted Array is: ";

print\_array(arr, n);

return 0;

}

1. insert an element in BST

#include<iostream>

using namespace std;

struct Node{

int data;

Node\* left;

Node\* right;

Node(int val){

data=val;

left=NULL;

right=NULL;

}

};

Node\* insertBST(Node\* root, int val){

if(root==NULL){

return new Node(val);

}

if(val<root->data){

root->left=insertBST(root->left,val);

}

else{

root->right=insertBST(root->right,val);

}

return root;

}

void inorder(Node\* root){

if(root==NULL){

return;

}

inorder(root->left);

cout<<root->data<<" ";

inorder(root->right);

}

int main(){

Node\* root=NULL;

root=insertBST(root,5);

insertBST(root,1);

insertBST(root,3);

insertBST(root,4);

insertBST(root,2);

insertBST(root,7);

inorder(root);

cout<<endl;

return 0;

}

1. search an element in BST

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int val) {

data= val;

left= NULL;

right= NULL;

}

};

void inorder(Node\* root){

if(root==NULL){

return;

}

inorder(root->left);

cout<<root->data<<" ";

inorder(root->right);

}

Node\* searchinBST(Node\* root, int key){

if(root==NULL){

return NULL;

}

if(root->data==key){

return root;

}

if(root->data>key){

return searchinBST(root->left,key);

}

else{

return searchinBST(root->right,key);

}

}

int main()

{

Node\* root=new Node(4);

root->left=new Node(2);

root->right=new Node(5);

root->left->left=new Node(1);

root->left->right=new Node(3);

root->right->right=new Node(6);

int data;

cout<<"data:";

cin>>data;

if(searchinBST(root, data)==NULL){

cout<<"doesnt exists";

}

else{

cout<<"does exists";

}

return 0;

}

1. preorder traversal in BST

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int val) {

data= val;

left= NULL;

right= NULL;

}

};

void preorder(struct Node\* root) {

if(root==NULL) {

return;

}

cout<<root->data<<" ";

preorder(root->left);

preorder(root->right);

}

int main()

{

struct Node\* root=new Node(1);

root->left=new Node(2);

root->right=new Node(3);

root->left->left=new Node(4);

root->left->right=new Node(5);

root->right->left=new Node(6);

root->right->right=new Node(7);

preorder(root);

return 0;

}

1. inorder traversal in BST

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int val) {

data= val;

left= NULL;

right= NULL;

}

};

void inorder(struct Node\* root){

if(root==NULL){

return;

}

inorder(root->left);

cout<<root->data<<" ";

inorder(root->right);

}

int main()

{

struct Node\* root=new Node(1);

root->left=new Node(2);

root->right=new Node(3);

root->left->left=new Node(4);

root->left->right=new Node(5);

root->right->left=new Node(6);

root->right->right=new Node(7);

inorder(root);

return 0;

}

1. postorder traversal in BST

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int val) {

data= val;

left= NULL;

right= NULL;

}

};

void postorder(struct Node\* root){

if(root==NULL) {

return;

}

postorder(root->left);

postorder(root->right);

cout<<root->data<<" ";

}

int main()

{

struct Node\* root=new Node(1);

root->left=new Node(2);

root->right=new Node(3);

root->left->left=new Node(4);

root->left->right=new Node(5);

root->right->left=new Node(6);

root->right->right=new Node(7);

postorder(root);

return 0;

}